ISBN 978-89-5884-861-5 93560

2006년도 HPC 기술서

코드 최적화/병렬화

2006년 12월

한국과학기술정보연구원

목 차

► CHAPTER I. GDAS 코드

► CHAPTER II. RCAV 코드

► CHAPTER III. KAIST PipeFlow 코드

► CHAPTER IV. 조선대 코드

그림 목차

표 목차

CHAPTER I. GDAS 코드

1. Makefile

Original 코드 및 최적화 코드의 Makefile(gdas /libs/opt_libs/options-ibmsp-thread)

```
F77=xlf_r
FORT_FLAGS=" -O3 -qfixed -qrealsize=8 -qnosave -qmaxmem=-1 -q64 ₩
                  -qarch=auto -qsmp=noauto"
LOAD_FLAGS=" -O3 -qrealsize=8 -qnosave -qmaxmem=-1 -q64 ₩
                  -qarch=auto -qsmp=noauto"
EXTRA_LIBS=
#
#   UTILS is normally same as the model options
#
UTIL_F77=xlf
UTIL_FORT_FLAGS=" -O3 -qrealsize=8 -qnosave -qmaxmem=-1 -q64 -qarch=auto"
UTIL_LOAD_FLAGS=" -O3 -qrealsize=8 -qnosave -qmaxmem=-1 -q64 -qarch=auto"
#
W3LIB_F77=xlf
W3LIB_FORT_FLAGS="-q64 -qarch=auto"
W3LIB_LOAD_FLAGS="-g -q64 -qarch=auto"
#
FFT99M=fft99m.o
#
BFLIB_F77=xlf
BFLIB_FORT_FLAGS=" -O3 -qrealsize=8 -q64 -qarch=auto"
BFLIB_LOAD_FLAGS=" -O3 -qrealsize=8 -q64 -qarch=auto"
```

2. Profiling 결과

gdas 프로그램은 여러 개의 실행 파일을 가지며, 이 실행파일이 의존적인 관계에서 순차적으로 돌아가는 스크립트에 의해서 실행이 되고 있다. 여러 개의 실행파일들 중 가장 많은 시간을 차지하는 두 개의 실행파일 oiqc.x와 ssi.x에 대해 최적화, 병렬화가 시도되었다.

2.1 oiqc.x 실행파일의 profiling 결과

<Original 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 240.72 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
28.3	68.08	68.08	158967	0.43	0.93	.searchs [4]
17.1	109.26	41.17				.__mcount [5]
11.0	135.63	26.37	225150780	0.00	0.00	._exp [10]
7.9	154.62	19.00	539253630	0.00	0.00	._cos [11]
6.7	170.81	16.18	137895620	0.00	0.00	._log [13]
3.5	179.23	8.43	108713120	0.00	0.00	._atan [14]
2.7	185.67	6.43	65846	0.10	0.40	.chdist [9]
2.4	191.50	5.83	134659157	0.00	0.00	.foi [15]
1.9	196.11	4.61	18848	0.24	0.94	.coor1 [12]
1.9	200.67	4.56	6545	0.70	4.70	.qcoi [7]
1.8	205.09	4.42	121199269	0.00	0.00	._sin [17]
1.4	208.56	3.47	85763127	0.00	0.00	.fnl [18]
1.4	212.01	3.45				._clc [19]
1.0	214.39	2.38	85773873	0.00	0.00	.flv [21]
0.9	216.60	2.21	86345701	0.00	0.00	.fkx [22]
0.9	218.75	2.14				.qincrement1 [23]
0.7	220.32	1.58				._pxldmod [25]
0.6	221.71	1.38				.qincrement [26]
0.5	222.96	1.25	18848	0.07	0.07	.vsolve [27]
0.5	224.17	1.21				.upbb [28]
0.5	225.37	1.20				.rcstpl [29]
0.5	226.51	1.14	139975582	0.00	0.00	.fln [30]
0.5	227.64	1.13	18848	0.06	0.25	.coor2 [16]
0.4	228.59	0.95				.__stack_pointer [33]
0.4	229.48	0.89	37757	0.02	0.02	.srtprs [34]

0.3	230.32	0.84	130437562	0.00	0.00	.flt [35]
0.3	231.08	0.76	1	760.00	978.33	.makeamap [32]
0.3	231.80	0.72				.pkb [37]
0.3	232.52	0.72				.rdtree [38]
0.3	233.18	0.66	6545	0.10	0.16	.seldat [31]
0.2	233.77	0.59	136269043	0.00	0.00	.fpr [39]
0.2	234.30	0.53	145608471	0.00	0.00	.fql [40]
0.2	234.68	0.38	17985763	0.00	0.00	.foe [41]
0.2	235.05	0.37				.call_pthread_init [42]
0.1	235.41	0.36				.irev [43]
0.1	235.76	0.36				.upb [44]
0.1	236.10	0.34	18848	0.02	0.08	.drctsl [24]
0.1	236.43	0.33				._xljjpow [46]
0.1	236.71	0.28	938087	0.00	0.00	.foc [48]
0.1	236.98	0.27				.invwin [50]
0.1	237.25	0.27				.wrtree [51]

<최적화 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 78.45 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
51.9	40.72	40.72	158967	0.26	0.29	.searchs [3]
7.4	46.54	5.82	65846	0.09	0.09	.chdist [7]
6.3	51.52	4.98				.atan [8]
4.6	55.10	3.58	18848	0.19	0.19	.coor1 [9]
3.3	57.71	2.61	6545	0.40	1.31	.qcoi_2_1 [5]
2.6	59.75	2.04				._clc [10]
2.0	61.34	1.59				.vcos_ [11]
2.0	62.93	1.59				vsincos [12]
2.0	64.47	1.54				.vexp [13]
1.9	65.95	1.48				._atan2 [14]
1.9	67.42	1.47				._pxldmod [15]
1.5	68.63	1.21				.rcstpl [17]
1.4	69.73	1.10	18848	0.06	0.06	.coor2 [18]
1.4	70.80	1.07	18848	0.06	0.06	.vsolve [19]
1.2	71.76	0.96				.upbb [20]
1.0	72.55	0.79				vlog [22]
0.9	73.23	0.68				.pkb [23]
0.8	73.89	0.66				.rdtree [24]
0.6	74.33	0.45				.irev [25]
0.6	74.77	0.44				.__mcount [26]

0.5	75.17	0.40				.upb [27]
0.4	75.49	0.32				.invwin [30]
0.4	75.77	0.28				.wrtree [31]
0.3	76.03	0.26				._xljjpow [32]
0.3	76.24	0.21	18848	0.01	0.07	.drctsl [16]
0.2	76.42	0.18	5940	0.03	0.03	.output [33]
0.2	76.57	0.15	1	150.00	150.00	.makeamap [34]
0.2	76.72	0.15				_log [35]
0.2	76.86	0.14				.IORead [37]
0.1	76.97	0.11	6545	0.02	0.02	.seldat [38]
0.1	77.07	0.09				.iupm [40]
0.1	77.16	0.09	1114094	0.00	0.00	.pilnlnp [41]
0.1	77.24	0.08				._moveeq [42]
0.1	77.32	0.08				.lstrpc [43]
0.1	77.39	0.07				.ufbrw [44]
0.1	77.45	0.06				.ReadUnit [45]
0.1	77.51	0.06				._xlfReadUfmt [46]
0.1	77.57	0.06				.ufbcpy [47]
0.1	77.62	0.05				.ipkm [48]
0.1	77.67	0.05				.status [49]
0.1	77.72	0.05				.string [50]
0.1	77.77	0.05				.usrtpl [51]

sin, cos, exp, log 등의 고유함수와 작은 크기의 사용자 함수들이 많이 호출되어 코드의 실행시간을 비효율적으로 소비하고 있다. 최적화 코드에서는 고유함수 호출을 대신해 MASS 벡터라이브러리를 사용해 코드 성능을 높이고 작은 크기의 사용자 함수 호출은 인라이닝 시켜 코드 효율성을 높이고 있다.

2.2 ssi.x 실행파일의 profiling 결과

<Original 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 1971.51 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
32.1	632.91	632.91	100	6329.10	7397.09	.qtoper [5]
30.0	1224.57	591.66	100	5916.60	6941.35	.qoper [6]

4.9	1320.30	95.73	11585	8.26	10.61	.s2g0 [9]
4.7	1412.80	92.50	11413	8.10	10.46	.ts2g0 [11]
3.6	1484.18	71.38	2856	24.99	29.70	.s2gvec [12]
3.5	1552.57	68.39	206732	0.33	0.33	.vpassm [16]
3.4	1619.66	67.09	2828	23.72	28.43	.ts2gvec [13]
3.0	1677.89	58.23	2800	20.80	25.50	.tgrad2s [14]
2.8	1733.48	55.59	2828	19.66	24.36	.grad2s [15]
2.0	1772.40	38.92	100	389.20	2151.20	.hoper [7]
1.7	1805.10	32.70	51683	0.63	2.35	.fft99m [10]
1.5	1833.89	28.79	2800	10.28	12.63	.tg2s0 [19]
1.4	1862.11	28.22	2857	9.88	12.23	.g2s0 [20]
1.3	1888.01	25.90	101	256.44	1968.63	.htoper [8]
0.7	1902.68	14.67	100	146.70	146.70	.intw [21]
0.7	1916.80	14.12	1	14120.00	1957875.51	.pcgsoi [4]
0.5	1927.05	10.25	25899	0.40	0.40	.fft99a [22]
0.5	1937.24	10.19	25784	0.40	0.40	.fft99b [23]
0.3	1942.75	5.51	100	55.10	55.10	.intt [25]
0.3	1948.18	5.43	100	54.30	646.07	.satop4 [17]
0.2	1952.26	4.08	100	40.80	40.80	.intqpw [26]
0.1	1954.64	2.38	101	23.56	28.27	.s2grad [28]
0.1	1956.94	2.30	101	22.77	27.48	.ts2grad [29]
0.1	1959.15	2.21	4	552.50	552.50	.rdgesc [32]
0.1	1961.20	2.05	402	5.10	5.10	.sdot [33]
0.1	1963.07	1.87	1	1870.00	2894.62	.fulldivt [27]
0.1	1964.10	1.03	8547264	0.00	0.00	._exp [37]

<최적화 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 268.10 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
13.1	35.16	35.16	100	351.60	351.60	.qtoper@OL@3 [4]
10.6	63.61	28.45	100	284.50	284.50	.qoper@OL@1 [5]
7.3	83.28	19.67	100	196.70	451.78	.hoper@OL@1 [2]
6.2	99.83	16.55	103366	0.16	0.16	.transpose [6]
6.2	116.38	16.55	101	163.86	406.76	.htoper@OL@2 [3]
5.5	131.25	14.87	102616	0.14	0.14	.s2g1 [7]
4.8	144.07	12.82	12700	1.01	1.01	.s2gvec1 [10]
4.8	156.86	12.79	102616	0.12	0.12	.ts2g1 [11]
3.4	166.01	9.15	12700	0.72	0.72	.tgrad2s [13]
3.1	174.29	8.28				.dcr241f [14]
3.0	182.44	8.15	12700	0.64	0.64	.grad2s1 [15]

3.0	190.59	8.15				.drc1633pl [16]
2.4	196.90	6.31	25654	0.25	0.25	.ts2gvec11 [18]
2.2	202.80	5.90	25654	0.23	0.23	.ts2gvec12 [19]
1.4	206.54	3.74	100	37.40	37.40	.intw@OL@2 [26]
1.3	210.15	3.61	25400	0.14	0.14	.tg2s0 [27]
1.3	213.66	3.51	25400	0.14	0.14	.g2s1 [28]
1.3	217.14	3.48				.drc241f [29]
1.2	220.49	3.35				.dcr1613nl [30]
1.2	223.62	3.13				.dcr1633nl [31]
1.0	226.39	2.77	100	27.70	27.70	.intqpw@OL@3 [33]
1.0	229.10	2.71	100	27.10	27.10	.satop4@OL@1 [34]
1.0	231.79	2.69	100	26.90	26.90	.intt@OL@2 [35]
0.9	234.10	2.31				.drc243f [37]
0.8	236.22	2.12	100	21.20	21.20	.satop4@OL@2 [38]
0.7	238.07	1.85	100	18.50	18.50	.intw [39]
0.7	239.91	1.84	100	18.40	18.40	.pcgsoi@OL@3 [40]
0.7	241.68	1.77	1400	1.26	1.26	.intw@OL@1 [41]
0.6	243.39	1.71	273	6.26	6.26	.s2g0 [42]
0.6	244.94	1.55	100	15.50	15.50	.pcgsoi@OL@6 [43]
0.5	246.34	1.40	1	1400.00	4065.41	.pcgsoi [25]
0.4	247.47	1.13	101	11.19	11.19	.ts2grad [44]
0.4	248.51	1.04				.dcr242f [47]
0.4	249.54	1.03	101	10.20	10.20	.s2grad [48]
0.4	250.53	0.99	12827	0.08	1.03	.ts2gvec1 [8]
0.3	251.43	0.90	100	9.00	9.00	.pcgsoi@OL@1 [50]
0.3	252.27	0.84	100	8.40	8.40	.pcgsoi@OL@5 [52]
0.3	253.10	0.83	100	8.30	8.30	.pcgsoi@OL@4 [53]
0.3	253.78	0.68	56	12.14	12.14	.s2gvec [55]
0.2	254.44	0.66	100	6.60	52.39	.hoper@OL@2 [20]
0.2	255.08	0.64				.dcr240f [56]
0.2	255.70	0.62				.upbb [57]

ssi.x 프로그램에서는 단순히 지시어 삽입에 의한 루프 병렬화를 시도한 것이 아니라 명시적인 데이터 블록분할을 통해 각 스레드에 작업을 할당하고 있으며 이 과정에서 original 코드의 자료 구조를 변경해 사용하고 있다. 이에 순차코드의 최적화 과정과 병렬화 과정을 분리하기 어려워 이곳에서는 최적화/병렬화 과정을 한번에 다루었다.

3. Hpmcount 결과

hpmcount는 사용자가 작성한 프로그램의 실제 실행 시간과 하드웨어 카운터에 의해 수집되는 정보, 사용자원 등의 전반적인 성능을 제공하는 커맨드라인 유틸리티이다. 아래는 original 코드와 최적화된 코드에 대해 hpmcount를 실행시킨 결과이다.

3.1 oiqc.x 실행파일의 hpmcount 결과

<Original 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

 Total execution time (wall clock time): 243.823715 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode            : 240.580000 seconds
 Total amount of time in system mode          : 0.440000 seconds
 Maximum resident set size                    : 42260 Kbytes
 Average shared memory use in text segment    : 76615 Kbytes*sec
 Average unshared memory use in data segment  : 8933904 Kbytes*sec
 Number of page faults without I/O activity   : 14162
 Number of page faults with I/O activity      : 29
 Number of times process was swapped out      : 0
 Number of times file system performed INPUT  : 0
 Number of times file system performed OUTPUT : 0
 Number of IPC messages sent                  : 0
 Number of IPC messages received              : 0
 Number of signals delivered                  : 0
 Number of voluntary context switches         : 147
 Number of involuntary context switches       : 371

 #######  End of Resource Statistics  ########

  PM_FPU_FDIV (FPU executed FDIV instruction)           :         639309323
  PM_FPU_FMA (FPU executed multiply-add instruction)    :       11134339010
  PM_FPU0_FIN (FPU0 produced a result)                  :       23939775390
  PM_FPU1_FIN (FPU1 produced a result)                  :       21879945242
  PM_CYC (Processor cycles)                             :      407246471244
```

```
PM_FPU_STF (FPU executed store instruction)            :        8238127803
PM_INST_CMPL (Instructions completed)                  :      368180403683
PM_LSU_LDF (LSU executed Floating Point load instruction) :      26529010346

Utilization rate                                :            98.019 %
Load and store operations                       :         34767.138 M
MIPS                                            :          1510.027
Instructions per cycle                          :             0.904
HW Float points instructions per Cycle          :             0.113
Floating point instructions + FMAs              :         48715.932 M
Float point instructions + FMA rate             :           199.800 Mflip/s
FMA percentage                                  :            45.711 %
Computation intensity                           :             1.401
```

<최적화 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

 Total execution time (wall clock time): 85.560483 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode                 : 84.070000 seconds
 Total amount of time in system mode               : 0.380000 seconds
 Maximum resident set size                         : 42316 Kbytes
 Average shared memory use in text segment         : 26074 Kbytes*sec
 Average unshared memory use in data segment       : 3116484 Kbytes*sec
 Number of page faults without I/O activity        : 14185
 Number of page faults with I/O activity           : 35
 Number of times process was swapped out           : 0
 Number of times file system performed INPUT       : 0
 Number of times file system performed OUTPUT      : 0
 Number of IPC messages sent                       : 0
 Number of IPC messages received                   : 0
 Number of signals delivered                       : 0
 Number of voluntary context switches              : 123
 Number of involuntary context switches            : 98

 #######  End of Resource Statistics  ########

  PM_FPU_FDIV (FPU executed FDIV instruction)                 :        637990992
  PM_FPU_FMA (FPU executed multiply-add instruction)          :      12953241741
  PM_FPU0_FIN (FPU0 produced a result)                        :      14656215540
```

```
PM_FPU1_FIN (FPU1 produced a result)                          :      14648092669
PM_CYC (Processor cycles)                                     :     143012696132
PM_FPU_STF (FPU executed store instruction)                   :       5568021302
PM_INST_CMPL (Instructions completed)                         :     179435135511
PM_LSU_LDF (LSU executed Floating Point load instruction)     :      11701862066

Utilization rate                          :        98.092 %
Load and store operations                 :     17269.883 M
MIPS                                      :      2097.173
Instructions per cycle                    :         1.255
HW Float points instructions per Cycle    :         0.205
Floating point instructions + FMAs        :     36689.529 M
Float point instructions + FMA rate       :      428.814 Mflip/s
FMA percentage                            :        70.610 %
Computation intensity                     :         2.124
```

Original 코드와 최적화된 순차코드의 hpmcount 결과에서 단위시간당 부동소수 연산회수를 나타내는 Float point instructions + FMA rate을 비교해 보면 oiqc.x 프로그램은 199.8 Mflip/s에서 428.8 Mflip/s로 변화하였다. 이를 통해 최적화된 코드에서 단위시간당 더 많은 부동소수 연산을 수행해 코드의 효율 및 성능이 더 나아졌음을 확인할 수 있다.

3.2 ssi.x 실행파일의 hpmcount 결과

<Original 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 2102.100262 seconds

########  Resource Usage Statistics  ########

Total amount of time in user mode                 : 2097.870000 seconds
Total amount of time in system mode               : 1.420000 seconds
Maximum resident set size                         : 341448 Kbytes
Average shared memory use in text segment         : 1063784 Kbytes*sec
Average unshared memory use in data segment       : 712712803 Kbytes*sec
Number of page faults without I/O activity        : 88664
Number of page faults with I/O activity           : 30
Number of times process was swapped out           : 0
Number of times file system performed INPUT       : 0
```

```
Number of times file system performed OUTPUT : 0
Number of IPC messages sent                  : 0
Number of IPC messages received              : 0
Number of signals delivered                  : 0
Number of voluntary context switches         : 278
Number of involuntary context switches       : 2839

#######   End of Resource Statistics   ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                 :          16995852
 PM_FPU_FMA (FPU executed multiply-add instruction)          :      145186451092
 PM_FPU0_FIN (FPU0 produced a result)                        :      167511434161
 PM_FPU1_FIN (FPU1 produced a result)                        :      195125509000
 PM_CYC (Processor cycles)                                   :     3545738040849
 PM_FPU_STF (FPU executed store instruction)                 :      162570250539
 PM_INST_CMPL (Instructions completed)                       :      912020895612
 PM_LSU_LDF (LSU executed Floating Point load instruction) :      371301601969

 Utilization rate                          :            98.988 %
 Load and store operations                 :       533871.853 M
 MIPS                                      :           433.862
 Instructions per cycle                    :             0.257
 HW Float points instructions per Cycle    :             0.102
 Floating point instructions + FMAs        :       345253.144 M
 Float point instructions + FMA rate       :       164.242 Mflip/s
 FMA percentage                            :            84.104 %
 Computation intensity                     :             0.647
```

<최적화 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

 Total execution time (wall clock time): 269.02648 seconds

 ########   Resource Usage Statistics   ########

 Total amount of time in user mode                 : 267.340000 seconds
 Total amount of time in system mode               : 0.980000 seconds
 Maximum resident set size                         : 500920 Kbytes
 Average shared memory use in text segment         : 265997 Kbytes*sec
 Average unshared memory use in data segment       : 131207155 Kbytes*sec
 Number of page faults without I/O activity        : 128387
 Number of page faults with I/O activity           : 0
 Number of times process was swapped out           : 0
```

```
Number of times file system performed INPUT   : 0
Number of times file system performed OUTPUT : 0
Number of IPC messages sent                   : 0
Number of IPC messages received               : 0
Number of signals delivered                   : 0
Number of voluntary context switches          : 277
Number of involuntary context switches        : 269

#######   End of Resource Statistics   ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                 :          18917544
 PM_FPU_FMA (FPU executed multiply-add instruction)          :      147849235345
 PM_FPU0_FIN (FPU0 produced a result)                        :      121131444648
 PM_FPU1_FIN (FPU1 produced a result)                        :      116313924982
 PM_CYC (Processor cycles)                                   :      454857385650
 PM_FPU_STF (FPU executed store instruction)                 :       34699875338
 PM_INST_CMPL (Instructions completed)                       :      469318294724
 PM_LSU_LDF (LSU executed Floating Point load instruction) :      167736794834

 Utilization rate                          :          99.223 %
 Load and store operations                 :       202436.670 M
 MIPS                                      :         1744.506
 Instructions per cycle                    :            1.032
 HW Float points instructions per Cycle    :            0.522
 Floating point instructions + FMAs        :       350594.730 M
 Float point instructions + FMA rate       :      1303.198 Mflip/s
 FMA percentage                            :          84.342 %
 Computation intensity                     :            1.732
```

ssi.x 프로그램에서는 단위시간당 부동소수 연산회수가 164.2 Mflip/s에서 1303.2 Mflip/s로 증가해 코드 성능과 효율이 상당 부분 개선되었음을 확인할 수 있다.

4. oiqc.x 코드 최적화/병렬화 비교 및 분석

프로파일링 정보를 참조하여, cpu 시간을 가장 많이 차지하는 루틴별로 정리를 하였다.

4.1 search.f

서브루틴 search는 그 자체가 프로파일 정보에서 가장 상위 자리를 차지하는 루틴은 아니다. 다만, 프로파일 정보의 최상단에 위치하는 루틴 searchs를 호출해 사용하는 루틴이므로 우선적으로 알아본다.

< Original 코드 >

```
18                      do nptr=1,nrep
19                      call searchs(nptr,itw,ipf)
21          1           ilv = flv(nptr)-1
22                      nlv = fnl(nptr)
23                      do k=1,2
24          1           do j=1,max(mtoth,mtotv)
25          1           do l=1,nlv
26          4           if(j.le.mtoth) pack = wtw(ilv+l,j,k,itw(l,j,k))
27          4           if(j.le.mtotv) pack = wpf(ilv+l,j,k,ipf(l,j,k))
28          4           enddo
29                      enddo
30                      enddo
32                      enddo
```

< 최적화/병렬화 코드>

```
32                  !$OMP PARALLEL DO PRIVATE(itw,ipf,ilv,nlv)
33                      do nptr=1,nrep
34                      call searchs(nptr,itw,ipf)
37                      ilv = iar1(nptr)-1
38                      nlv = iar7(nptr)
40                      do k=1,2
41                      do j=1,mtoth
42                          do l=1,nlv
43                              iar21(ilv+l,j,k)=itw(l,j,k)
```

```
44                              enddo
45                          enddo
46                          do j=1,mtotv
47                              do l=1,nlv
48                                  iar22(ilv+l,j,k)=ipf(l,j,k)
49                              enddo
50                          enddo
53                          enddo
54                          enddo
```

① Original 코드에서 호출해 사용하는 flv는 다음과 같이 단순히 배열 iar(i) 값을 함수 flv(i) 값으로 리턴해 주는 매우 간단한 함수이다. 최적화 코드에서는 함수를 호출하는 대신 코드에서 직접 iar(i) 값을 읽어 들여 함수 호출에 의한 부하를 감소시키고 있다. 유사한 함수 fnl도 호출해 사용하지 않고 이와 같은 함수 인라이닝을 시켜 최적화 하였다.

```
Function flv(i)
integer maxrep_
parameter(maxrep_=80000)
parameter (im = maxrep_)
common /stor1/iar(im)
integer iar
flv = iar(i)
return
end
```

② 함수 wtw 와 wpf 역시 인라이닝시키고 있으며, 루프를 분리해 if 문과 함수 max(mtoth, mtotv)를 사용하지 않도록 하고 있다.

```
Function wtw(i,j,k,v)
integer maxlev_
parameter(maxlev_=200000)
parameter (im = maxlev_)
parameter (jm = 10)
parameter (km = 2)
common /stor21/iar(im,jm,km)
integer iar,v
iar(i,j,k) = v
wtw = 0
return
end
```

4.2 searchs.f

프로파일 정보에서 최상단의 자리를 차지하는 서브루틴이다.

< Original 코드 >

```
42                          do i=1,maxlv*mtoth*2
43              55          itw(i,1,1) = 0
44             126          ipf(i,1,1) = 0
45                          enddo
46                          if(fql(nptr).le.0) return
```

< 최적화 코드>

```
67                          do i=1,maxlv*mtoth*2
68              77          itw(i,1,1) = 0
69                          enddo
70                          do i=1,maxlv*mtotv*2
71              36          ipf(i,1,1) = 0
72                          enddo
74                          if(iar8(nptr).le.0) return
```

① 각 배열을 초기화 하는 과정으로 maxlv = 255 , mtoth = 10, mtotv = 5 이다. itw(maxlv,mtoth,2)이고 ipf(maxlv,mtotv,2)이므로 루프를 분리해 배열 ipf에 대한 불필요한 메모리 접근이 없도록 하고 있다.

② iar8(nptr)을 리턴하는 함수 fql(nptr)를 호출하는 대신 iar8배열을 직접 읽어 들이 도록 수정하였다

< Original 코드 >

```
89                          do 10 j=j1,j2
90               9          do 10 n=imap(ia,j),imap(ib,j)
91              34          nn   = inob(n)
92             370          qla = fql(nn)
```

```
93          6              if(qla.ge.0. .and. qla.le.binq) then
94          7                 nprf = nprf+1
95        354                 nind(nprf) = nn
96        290                 rlon(nprf) = fln(nn)
97          7                 rlat(nprf) = flt(nn)
98                          endif
99         21      10       continue
```

< 최적화 코드>

```
122                         do 10 j=j1,j2
123         11              do 10 n=imap(ia,j),imap(ib,j)
124                         nn    = inob(n)
126                         qla = iar8(nn)
127        185              if(qla.ge.0. .and. qla.le.binq) then
128          5                 nprf = nprf+1
129         28                 nind(nprf) = nn
132          6                 rlon(nprf) = iar4(nn)
133        180                 rlat(nprf) = iar3(nn)
134                         endif
135          1      10       continue
```

③ 간단한 함수 flq(nn), fln(nn), flt(nn)을 인라인시켜 배열 iar8, iar4, iar3을 직접 읽어 들이고 있다.

< Original 코드 >

```
106          1              do 21 n=1,nprf
107         66              same = sid(nptr)(1:nc) .eq. sid(nind(n))(1:nc)
108        142              if(rdis(n).le.rscan .and. .not.same) then
109         63                 ilev    = flv(nind(n))
110        206                 klev    = ilev+fnl(nind(n))-1
111                            kx      = fkx(nind(n))
112        401                 do 20 jlev=ilev,klev
113                            if(foi(jlev).gt.0) then
114         76                    nobs = nobs+1
115        315                    lind(nobs) = jlev
116          6                    pres(nobs) = log(fpr(jlev))
117         52                    dist(nobs) = (rdis(n)/rade)**2
```

```
118          54                    dirn(nobs) = amod(rdir(n)/90.,4.) + 1.01
119                                ltyp(nobs) = kx/100
120                               endif
121           7        20          continue
122           8                    if(nobs.gt.maxcyl) call abt('search - nobs>maxcyl')
123                              endif
124          16        21         continue
```

< 최적화 코드>

```
142           1                   do 21 n=1,nprf
143          30                   same = sid(nptr)(1:nc) .eq. sid(nind(n))(1:nc)
144          11                   if(rdis(n).le.rscan .and. .not.same) then
146         163                       ilev   = iar1(nind(n))
148                                   klev   = ilev+iar7(nind(n))-1
150          83                       kx      = iar6(nind(n))
151           7                       do 20 jlev=ilev,klev
152         128                       if(iar16(jlev).gt.0) then
153           4                           nobs = nobs+1
154          20                           lind(nobs) = jlev
157           7                           pres(nobs) = iar14(jlev)
158         101                           dist(nobs) = (rdis(n)/rade)**2
159          94                           dirn(nobs) = amod(rdir(n)/90.,4.) + 1.01
160          22                           ltyp(nobs) = kx/100
161                                     endif
162           4        20              continue
163           1                     if(nobs.gt.maxcyl) call abt('search - nobs>maxcyl')
164                                 endif
165           7        21          continue
166                                 call vlog(pres,pres,nobs)
```

④ flv , fnl , fkx , foi, fpr 등의 함수를 인라인 하였고, 고유함수 log 대신 MASS 벡터 라이브러리를 사용하였다.

< Original 코드 >

```
143                               bprs = fpr(il)
144                               if(bprs.le.0 .or. bprs.gt.2000) goto 50
145                               iprs = bprs
```

```
146                                 bprs = log(bprs)
147             2                   do 24 n=1,nobs
148                                 chrz      = exp(-chlp(iprs)*(dist(n)))
149                                 cvrt      = 1./(1.+ cvc*(bprs-pres(n))**2)
150                                 ic        = chrz*cvrt*100.+1.
151           985                   icor(n) = catcor(ic)
152            30                   idir(n) = dirn(n)
153                          24     continue
154                                 do 25 j=1,4
155                                 do 25 i=1,4
156             1                   do 25 n=1,nobs
157          1077                   cat(n,i,j) = icor(n).eq.j .and. idir(n).eq.i
158                          25     continue
```

< 최적화 코드 >

```
188                                 bprs = iar14(il)
189                                 if(bprs.le.0 .or. bprs.gt.2000) goto 50
190                                 iprs = bprs
191                                 bprs = log(bprs)
192             1                   do n=1,nobs
193            51                       dist1(n) = -chlp(iprs)*(dist(n))
194                                 enddo
195                                 call vexp(dist1,dist1,nobs)
196             1                   do 24 n=1,nobs
198                                 chrz      = dist1(n)
199                                 cvrt      = 1./(1.+ cvc*(bprs-pres(n))**2)
200                                 ic        = chrz*cvrt*100.+1.
202           234                   icor(n) =icatcor(ic)
203           144                   idir(n) = dirn(n)
204                          24     continue
205                                 do 25 j=1,4
206                                 do 25 i=1,4
207                                 do 25 n=1,nobs
208          1101                   cat(n,i,j) = icor(n).eq.j .and. idir(n).eq.i
209                          25     continue
```

⑤ fpr 함수를 인라인 시켰다.

⑥ -chlp(iprs)*dist(n)의 exponent 계산을 위해 MASS 벡터 라이브러리를 이용하였다. 이를 위해 dist1 배열을 새롭게 정의해

사용하고 있다.

4.3 chdist.f

< Original 코드 >

```
46                      c   compute the distance
47                      c   --------------------
48                          do 10 i=1,np
49                          cosy1 = cos(y1*pi180)
50                          cosy2 = cos(y2(i)*pi180)
51                          cosdx = cos((x1-x2(i))*pi180)
52                          cosdy = cos((y1-y2(i))*pi180)
53                          s = 1.0-cosdy+cosy1*cosy2*(1.0-cosdx)
54                          s = sqrt(2.*s)
55              363         if(s.le..002) s = 0.
56               53         dist(i) = s*rade
57                      10  continue
```

< 최적화 코드>

```
47                      c   compute the distance
48                      c   --------------------
49                          do i=1,np
50              59          cosy2(i) = y2(i)*pi180
51              14          cosdx(i) = (x1-x2(i))*pi180
52              34          cosdy(i) = (y1-y2(i))*pi180
53                          enddo
54                          call vcos(cosy2,cosy2,np)
55                          call vcos(cosdx,cosdx,np)
56                          call vcos(cosdy,cosdy,np)
57                          do 10 i=1,np
58                          cosy1 = cos(y1*pi180)
62                          s = 1.0-cosdy(i)+cosy1*cosy2(i)*(1.0-cosdx(i))
63                          s = sqrt(2.*s)
64              37          if(s.le..002) s = 0.
65              134         dist(i) = s*rade
66                      10  continue
```

① 코사인 함수 계산을 위해 MASS 벡터 라이브러리를 이용하였다.

4.4 coor1.f

< Original 코드 >

```
   60                          do 20 i=1,np
---------------------------------- 생  략 ----------------------------------
   74                       c   compute the matrix of sines and cosines
   75                       c   ---------------------------------------
   76                           cosy1 = cos(y1(i)*pi180)
   77                           cosy2 = cos(y2(i)*pi180)
   78                           siny1 = sin(y1(i)*pi180)
   79                           siny2 = sin(y2(i)*pi180)
   80                           cosdx = cos((x1(i)-x2(i))*pi180)
   81                           cosdy = cos((y1(i)-y2(i))*pi180)
   82                           sindx = sin((x1(i)-x2(i))*pi180)
   83                           sindy = sin((y1(i)-y2(i))*pi180)
---------------------------------- 생  략 ----------------------------------
  140                        20    continue
```

< 최적화 코드 >

```
   65                          do i=1,np
   66              12              argy1(i)=y1(i)*pi180
   67              15              argy2(i)=y2(i)*pi180
   68               7              argdx(i)=(x1(i)-x2(i))*pi180
   69                              argdy(i)=(y1(i)-y2(i))*pi180
   70                          enddo
   71                          call vsincos(siny10,cosy10,argy1,np)
   72                          call vsincos(siny20,cosy20,argy2,np)
   73                          call vsincos(sindx10,cosdx10,argdx,np)
   74                          call vsincos(sindy10,cosdy10,argdy,np)
```

① 사인, 코사인 함수 계산을 위해 MASS 벡터 라이브러리 vsincos 라이브러리를 사용하였다. call vsincos(siny10,cosy10,argy1,np)는 배열 argy의 sin과 cos을 계산해 배열 siny10과 cosy10으로 각각 리턴한다.

< Original 코드 >

```
60                          do 20 i=1,np
 ------------------------------ 생  략 ------------------------------
84                    c   compute the normal angle
85                    c   ----------------------
86                          s = 1.0-cosdy+cosy1*cosy2*(1.0-cosdx)
87                          s = sqrt(2.*s)
88                    c   calculate the first partials with respect to x1 and y1
89                    c   ------------------------------------------------------
90                          si   =   max(s,smin)
91              21          si   =   1./si
92              15          dy1 =    si*(sindy-siny1*cosy2*(1.0-cosdx))
93                          dx1 =    si*(cosy1*cosy2*sindx)
94               4          dy2 = -si*(sindy+cosy1*siny2*(1.0-cosdx))
95                          dx2 = -dx1
96                    c   calculate the mixed partial derivatives
97                    c   ----------------------------------------
98                         dy1y2=   si*(siny1*siny2*(1.0-cosdx)-cosdy-dy1*dy2)
99                          dy1x2=   si*(siny1*cosy2*sindx-dy1*dx2)
100                         dx1x2= -si*(cosy1*cosy2*cosdx+dx1*dx2)
101                         dx1y2= -si*(cosy1*siny2*sindx+dx1*dy2)
102                   c   compute the various correlations
103                   c   --------------------------------
104             20          rho     = exp(-ch(i)*s*s)
105             39          drho    = -2.*ch(i)*s*rho
106              2          d2rho = -2.*ch(i)*(rho+s*drho)
107                         ww0     = 2.*ch(i)
108                         srww0 = sqrt(ww0)
109             70          d(i)    = s
-------------------------------- 생  략 --------------------------------
169                   20    continue
```

< 최적화 코드 >

```
75                          do i=1,np
76                       s1= 1.0-cosdy10(i)+cosy10(i)*cosy20(i)*(1.0-cosdx10(i))
77              73          s(i) = sqrt(2.*s1)
78              24          rho(i) = -ch(i)*s(i)*s(i)
79                          enddo
80                          call vexp(rho,rho,np)
81                          do 20 i=1,np
```

```
 82                          c1(i) = 0.
 83            3             c2(i) = 0.
 95                  c   compute the matrix of sines and cosines
 96                  c   ---------------------------------------
 98                      cosy1 = cosy10(i)
100                      cosy2 = cosy20(i)
102                      siny1 = siny10(i)
104                      siny2 = siny20(i)
109                      cosdx = cosdx10(i)
110                      sindx = sindx10(i)
111                      cosdy = cosdy10(i)
112                      sindy = sindy10(i)
117                 c   calculate the first partials with respect to x1 and y1
118                 c   ------------------------------------------------------
119                      si   =   max(s(i),smin)
120                      si   =   1./si
121            7         dy1 =    si*(sindy-siny1*cosy2*(1.0-cosdx))
122                      dx1 =    si*(cosy1*cosy2*sindx)
123           48         dy2 = -si*(sindy+cosy1*siny2*(1.0-cosdx))
124                      dx2 = -dx1
125                 c   calculate the mixed partial derivatives
126                 c   ---------------------------------------
127                     dy1y2=   si*(siny1*siny2*(1.0-cosdx)-cosdy-dy1*dy2)
128                      dy1x2=   si*(siny1*cosy2*sindx-dy1*dx2)
129                      dx1x2= -si*(cosy1*cosy2*cosdx+dx1*dx2)
130                      dx1y2= -si*(cosy1*siny2*sindx+dx1*dy2)
131                 c   compute the various correlations
132                 c   -------------------------------
134                      drho   = -2.*ch(i)*s(i)*rho(i)
135            1         d2rho = -2.*ch(i)*(rho(i)+s(i)*drho)
136                      ww0    = 2.*ch(i)
137                      srww0 = sqrt(ww0)
138           10         d(i)   = s(i)
-------------------------------- 생략 --------------------------------
169                   20   continue
```

② exponent 함수 계산을 위해 MASS 벡터 라이브러리를 사용하였다. 이를 위해 original 코드에서의 스칼라 s 대신 벡터 s를 정의해 사용하였고 이 과정에서 벡터 s를 정의하기위해 루프를 분리하였다.

4.5 qcoi.f

< Original 코드 >

```
  121                   c   set up vectors with information about each ob to be
checked
  122                   c   ------------------------------------------------------------
  123                         do 15 igrp=1,nxxyy
  124                         ndim = mdim(igrp,iq)
  125                         if(ndim.eq.0) goto 15
  126             2           ilev = mchk(igrp,1)
  127             2           irep = fhd(ilev)
  128                         tzf = ftz(ilev)
  129             1           xxc = fln(irep)
  130             1           yyc = flt(irep)
  131             2           ppc = fpr(ilev)
  132                         kkc = mchk(igrp,2)
  133                         chl = chlp(int(min(ppc,1500.)))
  134             3           fla = fflat(int(abs(yyc)+1.51))
  135                         ppc = log(ppc)
  136                         do 10 iob=1,ndim
  137             3           jlev   = mobs(igrp,iob,iq,1)
  138            12           jtyp   = mobs(igrp,iob,iq,2)
  139            10           jrep   = fhd(jlev)
  140             1           ivmax = ivmax+1
  141                         chls(ivmax) = chl
  142                         flat(ivmax) = fla
  143                         tzfa(ivmax) = tzf
  144                         xxcp(ivmax) = xxc
  145                         yycp(ivmax) = yyc
  146                         kkcp(ivmax) = kkc
  147            17           ppcp(ivmax) = ppc
  148            17           xxob(ivmax) = fln(jrep)
  149            24           yyob(ivmax) = flt(jrep)
  150             2           ppob(ivmax) = log(fpr(jlev))
  151            14           kkob(ivmax) = jtyp
  152            33           oeob(ivmax) = foe(jlev)
  153             2           obob(ivmax) = fob(jlev,ivar(jtyp))
  154                         ngrp(ivmax) = igrp
  155                         niob(ivmax) = iob
  156                   10    continue
  157                   15    continue
```

< 최적화 코드>

```
146                 c   set up vectors with information about each ob to be checked
147                     c   ------------------------------------------------------------
148                             do 15 igrp=1,nxxyy
149                             ndim = mdim(igrp,iq)
150               1             if(ndim.eq.0) goto 15
151                             ilev = mchk(igrp,1)
153                             irep = iar13(ilev)
155                             tzf = iar20(ilev)
158                             xxc = iar4(irep)
159                             yyc = iar3(irep)
161                             ppc = iar14(ilev)
162                             kkc = mchk(igrp,2)
163                             chl = chlp(int(min(ppc,1500.)))
164                             fla = fflat(int(abs(yyc)+1.51))
165                             ppc = log(ppc)
166               3             do 10 iob=1,ndim
167                             jlev   = mobs(igrp,iob,iq,1)
168                             jtyp   = mobs(igrp,iob,iq,2)
170                             jrep   = iar13(jlev)
171                             ivmax = ivmax+1
172               1             chls(ivmax) = chl
173               4             flat(ivmax) = fla
174                             tzfa(ivmax) = tzf
175               5             xxcp(ivmax) = xxc
176               3             yycp(ivmax) = yyc
177                             kkcp(ivmax) = kkc
178               2             ppcp(ivmax) = ppc
182               5             xxob(ivmax) = iar4(jrep)
183               3             yyob(ivmax) = iar3(jrep)
185               5             ppob(ivmax) = iar14(jlev)
186              17             kkob(ivmax) = jtyp
189               2             oeob(ivmax) = iar18(jlev)
190               8             obob(ivmax) = iar17(jlev,ivar(jtyp))
191                             ngrp(ivmax) = igrp
192                             niob(ivmax) = iob
193                         10  continue
194                         15  continue
195                             if(ivmax.eq.0) goto 61
196                             call vlog(ppob,ppob,ivmax)
```

① 함수 fhd, ftz, fln, flt, fpr, foe, fob 등을 인라이닝 하였다.

② 함수 fpr의 로그 값을 계산해 저장하는 배열 ppob의 계산은, 우선 fpr을 인라이닝시켜 배열 iar14의 값을 ppob에 할당하고 루프 밖에서 MASS 벡터 라이브러리를 이용해 배열 ppob의 로그 값을 계산하도록 하였다.

4.6 makeamap.f

< Original 코드 >

```
21                          do 1 n=1,nrep
22                          if(fln(n).eq.360.) pack = wln(n,0.)
23                    1     continue
24                    c   longitude sort
25                    c   --------------
26                          nn = 0
27                          do 10 nx=1,360
28              4           do 10 n=1,nrep
29                          ix = fln(n)+1.
30             23           if(ix.eq.nx) then
31                             nn = nn+1
32              1              indd(nn) = n
33                          endif
34             12     10    continue
35                    c   latitude sort
36                    c   -------------
37                          nob = 0
38                          do 20 ny=1,181
39             14           do 20 n=1,nn
40                          iy = flt(indd(n))+91.
41              7           if(iy.eq.ny) then
42                             nob = nob+1
43                             inob(nob) = indd(n)
44                          endif
45              6     20    continue
```

< 최적화 코드 >

```
26                          do 1 n=1,nrep
28                          if(iar4(n).eq.360.) then
29                             pack = 0.
```

```
30                          iar4(n)=0.
31                        endif
32                  1     continue
33                  c   longitude sort
34                  c   --------------
35                        nn = 0
36                        do 10 nx=1,360
37                        do 10 n=1,nrep
38                  c     ix = fln(n)+1.
39                        ix = iar4(n)+1.
40              9         if(ix.eq.nx) then
41                          nn = nn+1
42                          indd(nn) = n
43                        endif
44                  10    continue
45                  c   latitude sort
46                  c   -------------
47                        nob = 0
48                        do 20 ny=1,181
49                        do 20 n=1,nn
50                  c     iy = flt(indd(n))+91.
51                        iy = iar3(indd(n))+91.
52              7         if(iy.eq.ny) then
53                          nob = nob+1
54                          inob(nob) = indd(n)
55                        endif
56                  20    continue
```

① 함수 fln, wln, flt를 인라이닝 하였다. 함수 wln은 아래와 같이 변수 I,v을 받아 iar(n)=v 로 할당하는 역할을 수행한다.

```
FUNCTION wln(I,v)
integer maxrep_
parameter(maxrep_=80000)
parameter (im = maxrep_)
common /stor4/iar(im)
real iar,v
iar(i) = v
wln = 0
return
end
```

4.7 seldat.f

< Original 코드 >

```
33                          do 50 nptr=ind,ind+nxy-1
34                          i    = indd(nptr)
35          1               j    = fhd(i)
36                          kx = fkx(j)/100
37          3               do 45 k=kx,kx
38                          m = m + 1
39                          mchk(m,1) = i
40                          mchk(m,2) = k
41          1               fe   (m,1) = ffe(i,1)
42          2               fe   (m,2) = ffe(i,2)
43                          fe   (m,3) = fe (m,2)
44                    c   pick data from t and w groups
45                    c   ----------------------------
46                          do 20 ig=1,2
47                          npic = 0
48          2               do 15 ip=1,maxpic(ig)
49          5               if(npic.eq.maxn(ig)) goto 20
50          6               ipic = ftw(i,ip,ig)
51         30               if(ipic.eq.0) goto 20
52                          if(foe(ipic).gt.0) then
53          5                   npic = npic+1
54          7                   do 10 iq=ig,ig*2-1
55                              mdim(m,iq)            = npic
56                              mobs(m,npic,iq,1) = ipic
57                              mobs(m,npic,iq,2) = iq
58                    10        continue
59                          endif
60          1         15      continue
61                    20      continue
62                    c   pick data from same profile
63                    c   ---------------------------
64                          npic = 0
65                          do 30 ig=1,2
66                          do 25 ip=1,5
67                          ipic = fpf(i,ip,ig)
68          1               if(ipic.eq.0) goto 30
69                          if(foe(ipic).gt.0) then
70                              do 24 kk=k,k*2-1
71                              npic                  = npic+1
72                              mdim(m,4)             = npic
73                              mobs(m,npic,4,1) = ipic
```

```
74                                  mobs(m,npic,4,2) = kk
75                      24          continue
76                                  goto 30
77                                endif
78                      25        continue
79              1       30        continue
80              1       45        continue
81                      50        continue
```

< 최적화 코드 >

```
56                                  do 50 nptr=ind,ind+nxy-1
57                                  i    = indd(nptr)
59                                  j    = iar13(i)
61                                  kx = iar6(j)/100
62              4                   do 45 k=kx,kx
63                                  m = m + 1
64                                  mchk(m,1) = i
65                                  mchk(m,2) = k
68                                  fe   (m,1) = iar19(i,1)
69                                  fe   (m,2) = iar19(i,2)
70                                  fe   (m,3) = fe (m,2)
71                      c   pick data from t and w groups
72                      c   ----------------------------
73                                  do 20 ig=1,2
74                                  npic = 0
75              1                   do 15 ip=1,maxpic(ig)
76                                  if(npic.eq.maxn(ig)) goto 20
77                      c           ipic = ftw(i,ip,ig)
78                                  ipic = iar21(i,ip,ig)
79                                  if(ipic.eq.0) goto 20
80                      c           if(foe(ipic).gt.0) then
81              4                   if(iar18(ipic).gt.0) then
82              5                     npic = npic+1
83                                    do 10 iq=ig,ig*2-1
84                                    mdim(m,iq)            = npic
85                                    mobs(m,npic,iq,1) = ipic
86              1                     mobs(m,npic,iq,2) = iq
87                      10            continue
88                                  endif
89                      15        continue
90                      20        continue
91                      c   pick data from same profile
```

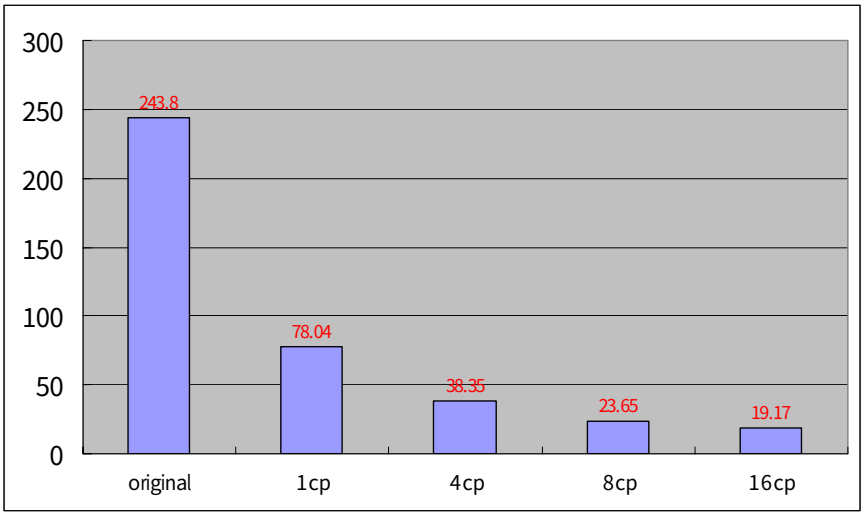
```
 92                  c   ---------------------------
 93                          npic = 0
 94                          do 30 ig=1,2
 95                          do 25 ip=1,5
 97                          ipic = iar22(i,ip,ig)
 98              2           if(ipic.eq.0) goto 30
100                          if(iar18(ipic).gt.0) then
101                             do 24 kk=k,k*2-1
102                             npic                = npic+1
103                             mdim(m,4)           = npic
104                             mobs(m,npic,4,1) = ipic
105                             mobs(m,npic,4,2) = kk
106                 24          continue
107                             goto 30
108                         endif
109                 25      continue
110                 30      continue
111                 45      continue
112                 50      continue
```

① 함수 fhd, fkx, fte, ftw, foe, fpf 등을 인라이닝 하였다.

5. oiqc summary

< 표 I.1 Original 코드와 최적화 코드의 Time Step당 계산 시간 비교 >

		Parallel Speed up
Original (Serial)	243.8 s	
Tuned 1cp	78.04 s	3.12 (original 대비)
4 cp	38.35 s	2.03 (Tuned 1cp 대비)
8 cp	23.65 s	3.29 (Tuned 1cp 대비)
16 cp	19.17 s	4.07 (Tuned 1cp 대비)

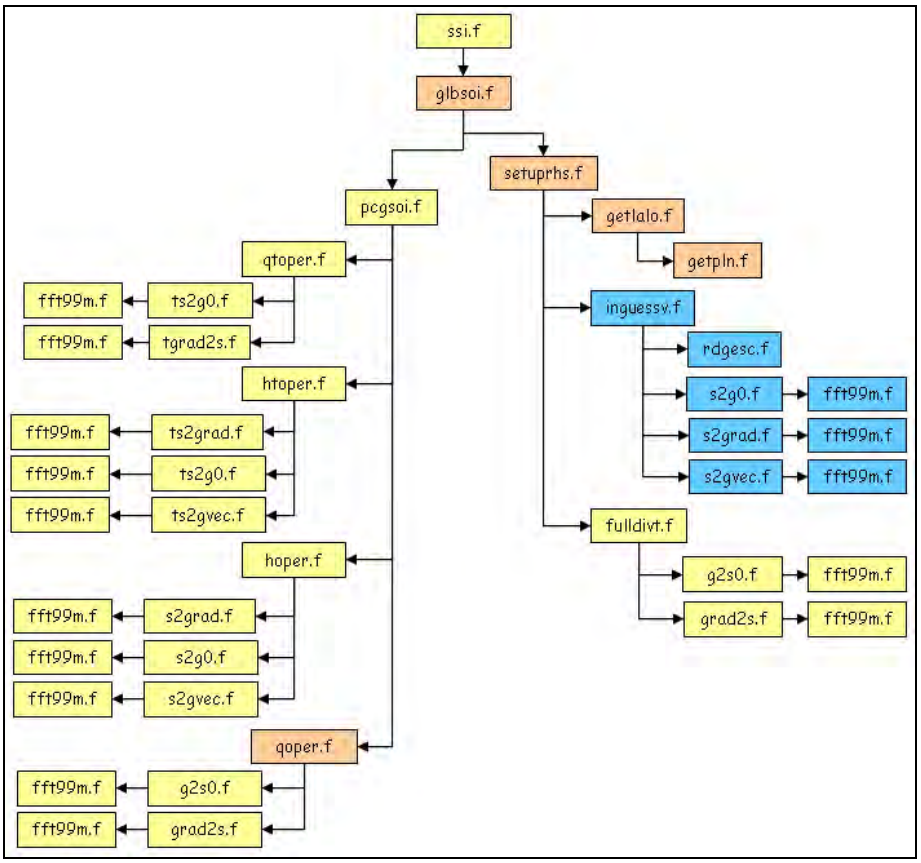


< 그림 I.1 오리지날 코드와 최적화 및 병렬화 코드의 성능 비교 그래프 >

oiqc.x 코드에서는 크기가 작은 함수를 많이 호출해 사용하고 있다. 크기가 작은 함수를 자주 호출해 사용하는 경우 호출된 함수의 자체 계산시간 보다 그 함수를 호출하는데 소비되는 시간이 많은 부분을 차지하게 되므로 비효율적이다. 이런 문제를 해결하기 위해 최적화 코드에서는 함수를 호출해 사용하지 않고 해당 함수의 계산 부분을 코드에 직접 삽입해 사용하는 인라이닝을 시도하였다. 그리고 log, cos, sin, exp 등의 고유함수 계산을 대신해 MASS 벡터 라이브러리를 사용하여 코드 성능을 높이고 있다.

또한 시간이 많이 걸리는 일부 루프에 대해 OpenMP 병렬지시어를 삽입해 병렬화를 진행하였다.

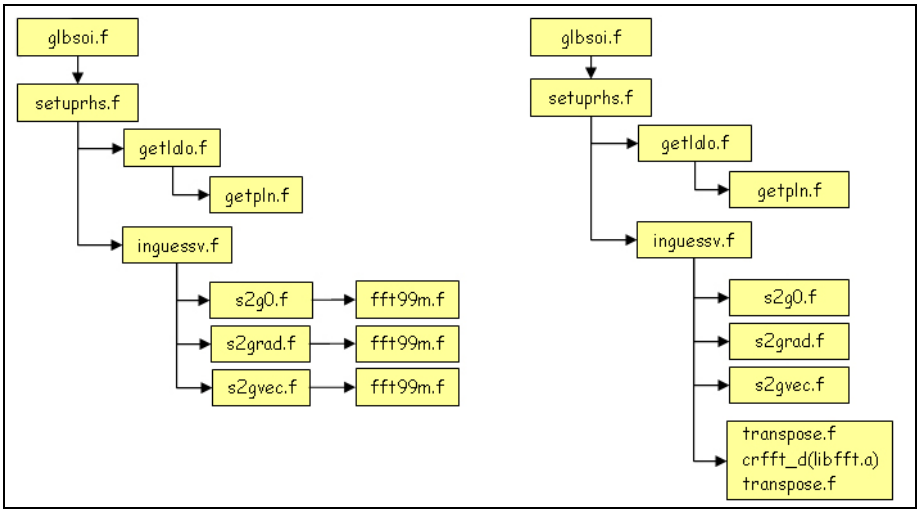
6. ssi.x 코드 최적화/병렬화 비교 및 분석



< 그림 I.2 ssi.x 프로그램 전체 Flowchart >

위 그림은 GDAS 프로그램의 주 실행부분인 ssi.x의 주요 소스코드 부분을 Flowchart로 나타낸 것이다. 전체 코드는 glbsoi.f가 setuprhs.f 와 pcgsoi.f 코드를 호출하고 이들이 호출하는 qtoper.f, htoper.f, hoper.f, qoper.f, inguessv.f, fulldivt.f 코드에서 핵심 계산들이 수행되도록 구성되어 있다. qtoper.f, htoper.f, hoper.f, qoper.f, inguessv.f, fulldivt.f 등이 호출해 사용하는 s2g0.f, ts2g0.f, s2grad.f, ts2grad.f, s2gvec.f, ts2gvec.f 등은 모두 유사한 계산을 실행하는 유사 루틴들이다. 최적화 코드에서는 이와 같은 유사 루틴들에 대해 모두 동일한 방식으로 최적화/병렬화를 수행

하고 있다. 따라서 이곳 에서는 setuprhs.f에서 호출해 사용하는 inguessv.f를 선택해 어떤 방식으로 최적화/병렬화가 수행되었는지 알아볼 것이다.



< 그림 I.3 setuprhs.f 이하 부분의 original 코드와 최적화 코드 flowchart 비교 >

위의 그림2에서 왼쪽은 original 코드의 setuprhs.f 이하 부분에 대한 flowchart를 나타낸 것이고 오른쪽은 최적화 코드에서의 setuprhs.f 이하 부분에 대한 flowchart를 나타낸 것이다. 우선 전체적으로 달라진 부분은 original 코드에서는 s2g0, s2grad, s2gvec과 같은 루틴에서 따로따로 fft99m을 호출하여 FFT 계산을 수행하고 있으나, 최적화 코드에서는 s2g0, s2grad, s2gvec를 호출한 후 마지막으로 한 번 FFT를 수행하고 있다는 점이다. Flowchart 상에서는 우선적으로 이와 같은 차이점이 눈에 들어오고 있으나 실제 코드에서는 좀 더 복잡하게 최적화/병렬화가 수행되어 있다. 앞에서 언급한 바와 같이 이곳에서의 병렬화는 단순하게 최적화된 순차코드에 루프 병렬화를 위한 parallel do 지시어 삽입으로 수행되어 있지 않다. 복잡한 자료구조를 명시적으로 분리시켜 각 스레드에 할당해 주는 방식으로 병렬화가 진행되었기에 순차코드에 대한 최적화 과정과 병렬화 과정을 따로 분리하기가 어려웠다. 그래서 최적화와 병렬화에 대한 내용을 같이

분석해 정리하였다.

6.1 glbsoi.f

<Original 코드>

```
195                             ii=-1
196                             do l=0,jcap
197                               do m=0,jcap-l
198                                 ii=ii+2
199                                 mlad(m,l)=ii
200                               end do
201                             end do
202                             ii=-1
203                             do m=0,jcap
204                               do l=0,jcap-m
205                                 ii=ii+2
206                                 lmad(m,l)=ii
207                               end do
208                             end do
209                             ii=-1
210                             do m=0,jcap
211                               do l=0,jcap-m
212                                 ii=ii+2
213                                 ml2lm(ii)=mlad(m,l)
214                                 ml2lm(ii+1)=ml2lm(ii)+1
215                               end do
216                             end do
217                             ii=-1
218                             do l=0,jcap
219                               do m=0,jcap-l
220                                 ii=ii+2
221                                 lm2ml(ii)=lmad(m,l)
222                                 lm2ml(ii+1)=lm2ml(ii)+1
223                               end do
224                             end do
```

<최적화/병렬화 코드>

```
329                             ii=0
330                             do m=0,jcap
331                               length(m)=ii+1
```

```
332                                ii=ii+2*(jcap+1-m)
333                               enddo
335          !$OMP PARALLEL DO PRIVATE(jm,jl,zero)
336                               do m=0,jcap
337                                jm=length(m)
338                                do l=0,jcap-m
339                                 jl=length(l)
340                                 mlad(l,m)=jm+2*l
341                                 lmad(m,l)=jm+2*l
342                                 ml2lm(jm+2*l   )=jl+2*m
343                                 ml2lm(jm+2*l+1)=jl+2*m+1
344                                 lm2ml(jm+2*l   )=jl+2*m
345                                 lm2ml(jm+2*l+1)=jl+2*m+1
346                                end do
```

① 배열 mlad, lmad, ml2lm, lm2ml에 대한 초기화를 수행하는 루프를 하나로 합쳤다. 이를 위해 length(m)을 새롭게 정의해 사용하고 있다. 아울러 하나로 결합된 초기화 과정을 parallel do 지시어를 삽입해 병렬 처리하였다.

<Original 코드>

```
225                               ii=-1
226                               do m=0,jcap
227                                  ii=ii+2
228                                  factslm(ii)=1.
229                                  factslm(ii+1)=0.
230                                  if(m.lt.jcap) then
231                                  do l=1,jcap-m
232                                     ii=ii+2
233                                     factslm(ii)=1.
234                                     factslm(ii+1)=1.
235                                  end do
236                                  end if
237                               end do
238                               do i=1,(jcap+1)*(jcap+2)
239                                  factvlm(i)=factslm(i)
240                               enddo
241                               factvlm(1)=0.
```

```
242                         ii=-1
243                         do l=0,jcap
244                           one=1.
245                           zero=min(1,l)
246                           do m=0,jcap-l
247                             ii=ii+2
248                             factsml(ii)=one
249                             factsml(ii+1)=zero
250                           end do
251                         end do
252                         do i=1,(jcap+1)*(jcap+2)
253                           factvml(i)=factsml(i)
254                         enddo
255                         factvml(1)=0.
```

<최적화/병렬화 코드>

```
335                     !$OMP PARALLEL DO PRIVATE(jm,jl,zero)
336                           do m=0,jcap
337                            jm=length(m)
348                            factslm(jm   )=1.
349                            factslm(jm+1)=0.
350                            factvlm(jm   )=factslm(jm)
351                            factvlm(jm+1)=factslm(jm+1)
352                            if(m.lt.jcap) then
353                             do l=1,jcap-m
354                               factslm(jm+2*l)=1.
355                               factslm(jm+2*l+1)=1.
356                               factvlm(jm+2*l   )=factslm(jm+2*l)
357                               factvlm(jm+2*l+1)=factslm(jm+2*l+1)
358                             end do
359                            end if
361                            if(m.eq.0) then
362                                zero=0.
363                            else
364                                zero=1.
365                            endif
366                            do l=0,jcap-m
367                             factsml(jm+2*l   )=1.
368                             factsml(jm+2*l+1)=zero
369                             factvml(jm+2*l   )=factsml(jm+2*l)
370                             factvml(jm+2*l+1)=factsml(jm+2*l+1)
371                            end do
```

```
372                     c
373                           do l=0,jcap-m
374                            in(jm+2*l)=m+l
375                            in(jm+2*l+1)=m+l
376                           end do
377                          end do
378                          factvlm(1)=0.
379                          factvml(1)=0.
```

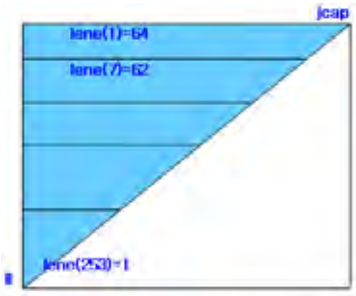
② 서로 다른 루프에서 수행되는 배열 factslm, factvlm, factsml, factvml의 초기화 과정을 하나의 루프로 합치고 있다. 또 하나로 합쳐진 루프에 대해 병렬 지시어를 삽입해 병렬 처리하고 있다.

< 최적화/병렬화 코드 >

```
259                     c--------
 261                           call cut_thds(ise,1,nlon,nthds)
 266                           do ll=1,2*(jcap+1),2
 267                               lene(ll)=(jcap-(ll-1)/2)/2+1
 268                               lene(ll+1)=(jcap-(ll-1)/2)/2+1
 269                           end do
 270                           do ll=1,2*jcap,2
 271                               leno(ll)=(jcap+1-(ll-1)/2)/2
 272                               leno(ll+1)=(jcap+1-(ll-1)/2)/2
 273                           end do
 274                               ll=2*jcap+1
 275                               leno(ll)=0
 276                               leno(ll+1)=0
 277                           do ll=1,2*(jcap+1)
 278                               lenej(ll)=lene(ll)*nlath
 279                               lenoj(ll)=leno(ll)*nlath
 280                           end do
 281                      c
 282                           call cut_thdstr(lse,jcap,lene,leno,nthds)
 283                               nne=0
 284                               nno=0
 285                               j=0
 286                               llse(j)=nne
 287                               llsbe(j)=nne*nlath
 288                               llso(j)=nno
 289                               llsbo(j)=nno*nlath
```

```
290                     do j=0,nthds-2
291                        do ll=lse(1,j),lse(2,j)
292                           nne=nne+lene(ll)
293                           nno=nno+leno(ll)
294                        enddo
295                        llse(j+1)=nne
296                        llsbe(j+1)=nne*nlath
297                        llso(j+1)=nno
298                        llsbo(j+1)=nno*nlath
299                     end do
```

③ 최적화 코드에서 추가된 부분으로, 실제 계산되는 데이터의 구조에 맞추어 인덱스를 만들어 주는 부분이다. 코드에서 사용되는 데이터는 아래 그림과 같이 역 삼각형 형태의 데이터를 사용하게 된다. 이를 위해 original 코드에서는 복잡한 인덱스 구조를 가지고 배열에 접근하게 된다(s2g0.f 참조). 최적화 코드에서는 각 ll에 대해 접근되는 데이터 수를 새로운 배열 lene(ll), leno(ll)에 저장하고 있다. 아울러 lenej, lenoj, llse, llsbe, llso, llsbo 역시 아래와 같은 데이터에 접근하기 위해 사용되는 인덱스 정보를 저장하기 위해 최적화 코드에서 새롭게 정의된 배열이다.



< 그림 I.4 배열 구조에 따른 인덱스 >

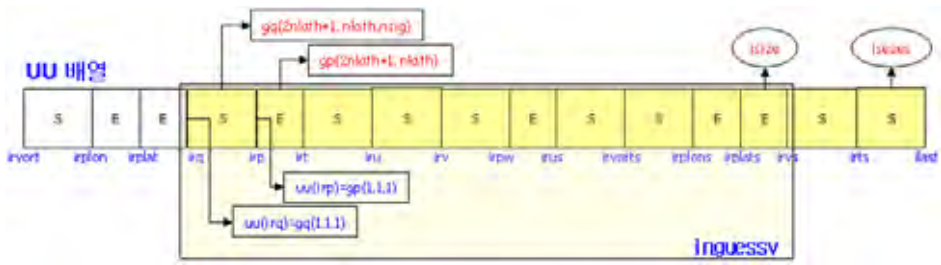
lene의 값:

lene(1)=64, lene(2)=64, lene(3)=63, lene(4)=63, lene(5)=63 …

lene(253)=1

< 최적화/병렬화 코드 >

```
304                     c-----pointer in uu array for hoper
305                           isze=(2*nlath+1)*(nlon+2)
306                           iszes=(2*nlath+1)*(nlon+2)*nsig
307                     c
308                           irvort=1
309                           irplon=irvort+iszes
310                           irplat=irplon+isze
311                     c
312                           irq=irplat+isze
313                           irp=irq+iszes
314                           irt=irp+isze
315                           iru=irt+iszes
316                           irv=iru+iszes
317                           irpw=irv+iszes
319                           irus=irpw+isze
320                           irvorts=irus+iszes
321                           irplons=irvorts+iszes
322                           irplats=irplons+isze
324                           irvs=irplats+isze
325                           irts=irvs+iszes
326                           ilast=irts+iszes
```



< 그림 I.5 UU 배열의 시작포인트 >

④ 최적화 코드에서 추가된 부분으로, uu 배열의 인덱스를 지정해주고 있다. original 코드에서 gq, qp, gu, gv 등의 이름으로 처리되는 여러 배열을, 최적화 코드에서는 하나의 일차원 배열 uu로 합

쳐 놓았다. 이곳에서 정의되는 irq는 uu배열에서 gq데이터가 시작되는 위치를 나타내고, irp는 uu배열에서 gp데이터가 시작하는 위치를 타낸다(위 그림 참조). irp, irt, iru 등도 역시 해당 배열의 uu에서의 위치를 나타낸다. 이렇게 여러 가지 배열을 하나의 일차원 배열로 합쳐 처리하는 이유는 FFT 계산의 병렬화를 위해서이며, 이는 뒤(inguessv.f 설명부분)에서 알아볼 것이다.

<cut_thds.f: cut_thds>

```
 1                       subroutine cut_thds(ks,kstart,kend,nthds)
 2                       integer ks(2,*)
 4                       nn=kend-kstart+1
 5                       nblk=nn/nthds
 6                       ndelta=mod(nn,nthds)
 7                       if(ndelta.gt.0) then
 8                           k=1
 9                           ks(1,1)=1
10                           ks(2,1)=ks(1,1)+nblk
11                       do k=2,ndelta
12                           ks(1,k)=ks(1,k-1)+nblk+1
13                           ks(2,k)=ks(1,k)+nblk
14                       enddo
15                           k=ndelta+1
16                           ks(1,k)=ks(1,k-1)+nblk+1
17                           ks(2,k)=ks(1,k)+nblk-1
18                       do k=ndelta+2,nthds
19                           ks(1,k)=ks(1,k-1)+nblk
20                           ks(2,k)=ks(1,k)+nblk-1
21                       enddo
22                       else
23                           ks(1,1)=1
24                           ks(2,1)=ks(1,1)+nblk-1
25                       do k=2,nthds
26                           ks(1,k)=ks(1,k-1)+nblk
27                           ks(2,k)=ks(1,k)+nblk-1
28                       enddo
29                       endif
30                       do k=1,nthds
31                           ks(1,k)=ks(1,k)+kstart-1
32                           ks(2,k)=ks(2,k)+kstart-1
33                       enddo
```

```
35                          return
36                          end
```

⑤ 병렬화를 위한 루틴이며, kstart에서 kend까지의 값을 주어진 개수(nthds)만큼의 스레드에 블록분할해 준다. nthds개의 스레드 중 p(0 ≤ p ≤ nthds-1) 스레드가 할당 받게되는 데이터 영역은 ks(1,p)부터 ks(2,p)까지가 된다.

nlon = 100(kstart=1, kend=100), nthds = 2 일 때

ks(1,1) = 1 , ks(2,1) = 50

ks(1,2) = 51 , ks(2,2) = 100

nlon = 100(kstart=1, kend=100), nthds = 3 일 때

ks(1, 1) = 1 , ks(2, 1) = 34 (34)

ks(1, 2) = 35 , ks(2, 2) = 67 (33)

ks(1, 3)= 68 , ks(2, 3) = 100 (33)

nlon = 100(kstart=1, kend=100), nthds = 6 일 때

ks(1, 1) = 1 , ks(2, 1) = 17 (17)

ks(1. 2) = 18 , ks(2, 2) = 34 (17)

ks(1, 3)= 35 , ks(2, 3) = 51 (17)

ks(1, 4) = 52 , ks(2, 4) = 68 (17)

ks(1. 5) = 69 , ks(2, 5) = 84 (16)

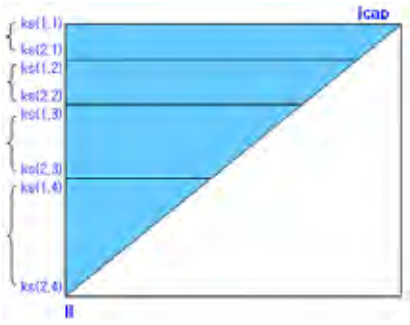
ks(1, 6)= 85 , ks(2, 6) = 100 (16)

<cut_thds.f: cut_thdstr>

```
37                          subroutine cut_thdstr(ks,jcap,lene,leno,nthds)
38                           integer ks(2,*)
39                           integer lene(*),leno(*)
41                           nc=(jcap+1)*(jcap+2)
42                           nct=max(1,nc/nthds)
43                           nn0=0
44                           ll0=1
45                           do np=1,nthds
```

```
46                              nn=0
47                              nc=nc-nn0
48                              nct=max(1,nc/(nthds-np+1))
50                            do ll=ll0,2*(jcap+1),2
51                              nn0=nn
52                              nn=nn+lene(ll)+leno(ll)+lene(ll+1)+leno(ll+1)
53                              if(nn.gt.nct) then
54                                ll0=ll
55                                goto 101
56                              endif
57                            enddo
58                              ll0=2*(jcap+1)+1
59                     101    continue
61                              ks(2,np)=ll0-1
62                            enddo
64                            ks(1,1)=1
65                            do np=2,nthds
66                              ks(1,np)=ks(2,np-1)+1
67                            enddo
78                            return
79                            end
```

⑥ 앞서 언급된 바와 같이 실제 계산에 사용되는 데이터는 아래와 같이 역삼각형 구조를 가진다. 병렬화 코드에서는 이와 같은 계산을 행방향으로 블록분할해 병렬 계산하고 있다. 이때 각 스레드에 할당되는 행의 수를 아래쪽으로 갈수록 많게 해 각 스레드가 담당해야 할 계산량을 균등하게 해야 할 필요가 있다. 서브루틴 cut_thdstr은 그림에서와 같이 스레드에 할당되는 계산량이 균등해 지도록 1 ~ ll까지의 데이터를 nthds개의 스레드에 블록분할해 준다. 이때 역 삼각형 형태의 데이터 개수는 앞서 정의한 배열 lene와 leno에 저장돼 있으므로 이를 이용하고 있다.



< 그림 I.6 로드벨런싱을 맞춘 병렬화 데이터 구조 >

실제 코드에서 스레드를 4개 사용할 경우 다음과 같이 데이터가 블록 분할된다.

< 표 I.2 스레드 블록분할 시 데이터 분포 >

np	ks(1,np)	ks(2,np)	size
1	1	34	34
2	35	74	40
3	75	126	52
4	127	256	130

6.2 getpln.f

< Original 코드 >

```
65                              ii=-1
66                              do m=0,jcap
67                                do l=0,jcap-m
68                                  ii=ii+2
69                                  iadr(l,m)=ii
70                                end do
71                              end do
72                              do m=0,jcap
73                                del2out(iadr(0,m))=del2(m,0)
74                                del2out(iadr(0,m)+1)=0.
75                                if(m.lt.jcap) then
76                                  do l=1,jcap-m
```

```
 77                                  del2out(iadr(l,m))=del2(m,l)
 78                                  del2out(iadr(l,m)+1)=del2(m,l)
 79                                 end do
 80                                end if
 81                               end do
```

< 최적화/병렬화 코드 >

```
 88                               do m=0,jcap
 89                                jm=length(m)
 90                                do l=0,jcap-m
 91                                 iadr(l,m)=jm+2*l
 92                                end do
 93                                del2out(iadr(0,m))=del2(m,0)
 94                                del2out(iadr(0,m)+1)=0.
 95                                if(m.lt.jcap) then
 96                                 do l=1,jcap-m
 97                                  del2out(iadr(l,m))=del2(m,l)
 98                                  del2out(iadr(l,m)+1)=del2(m,l)
 99                                 end do
100                                end if
101                               end do
```

① 루프 인덱스 m에 대해서 두 개의 루프를 하나로 합쳤다.

< Original 코드 >

```
 83                               do itmp=1,((jcap+1)*(jcap+2))*nlath
 84              3                   pln(itmp,1)=0.
 85                                 enddo
 87                                 do itmp=1,((jcap+1)*(jcap+2))*nlath
 88              2                   qln(itmp,1)=0.
 89                                 enddo
 91                                 do itmp=1,((jcap+1)*(jcap+2))*nlath
 92              2                   rln(itmp,1)=0.
 93                                 enddo
 95                                 do jtmp=0,jcap
 96                                   do itmp=1,nlath
 97                                     po(itmp,jtmp)=0.
 98                                   enddo
```

```
 99                                    enddo
101                                    do jtmp=0,jcap
102                                       do itmp=1,nlath
103                                          pe(itmp,jtmp)=pe0(itmp,jtmp)
104                                       enddo
105                                    enddo
107                                    do jtmp=0,jcap
108                                       do itmp=1,nlath
109                                          qo(itmp,jtmp)=0.
110                                       enddo
111                                    enddo
113                                    do jtmp=0,jcap
114                                       do itmp=1,nlath
115                                          qe(itmp,jtmp)=qe0(itmp,jtmp)
116                                       enddo
117                                    enddo
119                                    do jtmp=0,jcap
120                                       do itmp=1,nlath
121                                          re(itmp,jtmp)=0.
122                                       enddo
123                                    enddo
125                                    do jtmp=0,jcap
126                                       do itmp=1,nlath
127                                          ro(itmp,jtmp)=ro0(itmp,jtmp)
128                                       enddo
129                                    enddo
```

< 최적화/병렬화 코드 >

```
120                       !$OMP PARALLEL DO
121                             do j=1,nlath
122                                do itmp=1,((jcap+1)*(jcap+2))
123              1                    pln(itmp,j)=0.
124              5                    qln(itmp,j)=0.
125              1                    rln(itmp,j)=0.
126                                enddo
127                             enddo
128                        !$OMP PARALLEL DO
129                             do jtmp=0,jcap
130                                do itmp=1,nlath
131                                   po(itmp,jtmp)=0.
132                                   pe(itmp,jtmp)=pe0(itmp,jtmp)
133                                   qo(itmp,jtmp)=0.
```

```
134                                 qe(itmp,jtmp)=qe0(itmp,jtmp)
135                                 re(itmp,jtmp)=0.
136                                 ro(itmp,jtmp)=ro0(itmp,jtmp)
137                               enddo
138                             enddo
```

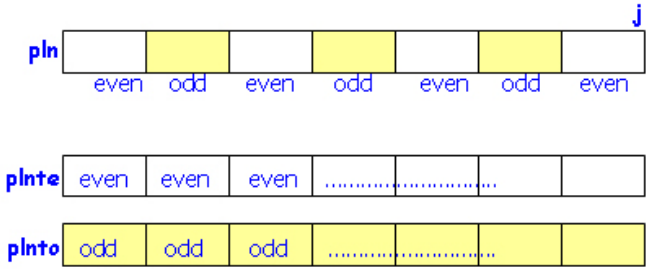
② 배열 pln, qln, rln에 대한 초기화 과정을 하나의 루프로 합쳤고, 배열 pe, po, qe, qo, re, ro를 초기화 하는 과정 역시 하나의 루프로 합쳤다. 그리고, 합쳐진 루프에 대해 OpenMP지시어를 삽입해 병렬화 하였다.

< 최적화 코드 – 추가됨>

```
182                             ii0=0
183                              do m=0,jcap-1,2
184                                jj=m/2+1
185                                mme0=0
186                                do ll=1,2*(jcap+1-m)
187                                   jj1=m/2+1+(j-1)*lene(ll)+mme0
188             2                     plnte(j,jj)=pln(ii0+ll,j)
189             1                     qlnte(j,jj)=qln(ii0+ll,j)
190             6                     rlnte(j,jj)=rln(ii0+ll,j)
191             1                     plnee(jj1)=pln(ii0+ll,j)
192             7                     qlnee(jj1)=qln(ii0+ll,j)
193                                   rlnee(jj1)=rln(ii0+ll,j)
194                                   jj=jj+lene(ll)
195             2                     mme0=mme0+lenej(ll)
196                                end do
197                                ii0=ii0+2*(jcap+1-m)
198                                jj=m/2+1
199                                mmo0=0
200                                do ll=1,2*(jcap-m)
201             4                     jj1=m/2+1+(j-1)*leno(ll)+mmo0
202                                   plnto(j,jj)=pln(ii0+ll,j)
203             8                     qlnto(j,jj)=qln(ii0+ll,j)
204             5                     rlnto(j,jj)=rln(ii0+ll,j)
205             2                     plnoo(jj1)=pln(ii0+ll,j)
206             9                     qlnoo(jj1)=qln(ii0+ll,j)
207             1                     rlnoo(jj1)=rln(ii0+ll,j)
208                                   jj=jj+leno(ll)
```

```
209          3                  mmo0=mmo0+lenoj(ll)
210                           end do
211                            ii0=ii0+2*(jcap-m)
212                          enddo
213                             m=jcap
214                           jj=m/2+1
215                           mme0=0
216                           do ll=1,2*(jcap+1-m)
217                             jj1=m/2+1+(j-1)*lene(ll)+mme0
218                             plnte(j,jj)=pln(ii0+ll,j)
219                             qlnte(j,jj)=qln(ii0+ll,j)
220                             rlnte(j,jj)=rln(ii0+ll,j)
221                             plnee(jj1)=pln(ii0+ll,j)
222                             qlnee(jj1)=qln(ii0+ll,j)
223                             rlnee(jj1)=rln(ii0+ll,j)
224                             jj=jj+lene(ll)
225                             mme0=mme0+lenej(ll)
226                           end do
227                      end do
```

③ 이 부분은 최적화 과정에서 추가된 것이다. Original 코드에서는 pln, qln, rln 등의 배열에 아래 그림과 같이 even과 odd 항이 번갈아 가며 나열되어 있다. 이로 인해 실제 계산에서는 각 배열에서 even 항과 odd 항을 구분해 사용하기 위해 복잡한 계산식을 사용하고 있다(s2g0.f 참조). 최적화 코드에서는 각 배열 pln, qln, rln의 even 항과 odd 항을 이곳에서 미리 분리해 even 항은 plnte, qlnte, rlnte, plnee, qlnee, rlnee에 odd 항은 plnto, qlnto, rlnto, plnoo, qlnoo, rlnoo에 저장해 두게 된다.



< 그림 I.7 pln 배열의 even항과 odd항 정리 >

6.3 inguessv.f

< Original 코드 >

```
110                         do kk=1,nsig*3+3
111                          if(kk.eq.nsig*3+1) then
113                           do itmp=1,(jcap+1)*(jcap+2)
114                           delps(itmp)=-del2(itmp)*pc(itmp)
115                           enddo
116                           call s2grad(delps,plonb,platb,jcap,nlon,nlath,
117                     *                 qln,rln,trigs,ifax)
118                          end if
119                          if(kk.eq.nsig*3+2)
120                     *      call s2g0(pc,gp,jcap,nlon,nlath,pln,trigs,ifax)
121                          if(kk.eq.nsig*3+3)
122                     *     call s2g0(rc,gmtns,jcap,nlon,nlath,pln,trigs,ifax)
123                          k=mod(kk-1,nsig)+1
124                          if(kk.ge.1.and.kk.le.nsig) then
125                         call s2g0(zc(1,k),vortb(1,1,k),jcap,nlon,nlath,pln,
126                     *                 trigs,ifax)
127                          call s2g0(dc(1,k),divb(1,1,k),jcap,nlon,nlath,pln,
128                     *                 trigs,ifax)
129                           call s2gvec(zc(1,k),dc(1,k),gu(1,1,k),gv(1,1,k),
130                     *           jcap,nlon,nlath,qln,rln,trigs,ifax)
131                          end if
132                          if(kk.ge.nsig+1.and.kk.le.2*nsig) then
133                            call s2g0(tc(1,k),gt(1,1,k),jcap,nlon,nlath,pln,
134                     *                 trigs,ifax)
135                          endif
136                          if(kk.ge.2*nsig+1.and.kk.le.3*nsig)
137                     *     call s2g0(qc(1,k),gq(1,1,k),jcap,nlon,nlath,pln,
138                     *                 trigs,ifax)
139                         end do
```

① original 코드의 구조

루프 인덱스 kk에 의해 nsig*3 +3회 반복되며 IF 구문으로 각 kk에서 실행부분을 판단하고 있다.

- kk가 1 ~ nsig인 경우 call s2g0(*zc(1,k)*, vortb(1,1,k), …), call

s2g0(*dc(1,k),* **divb(1,1,k),** …), call s2gvec(*zc(1,k), dc(1,k),* **gu(1,1,k), gv(1,1,k),** …)

- kk가 nsig+1 ~ 2*nsig인 경우 call s2g0*(tc(1,k),* **gt(1,1,k)**, …)
- kk가 2*nsig+1 ~ 3*nsig인 경우 call s2g0(*qc(1,k),* **gq(1,1,k)**, …)
- kk가 3*nsig+1인 경우 call s2grad(*delps*, **plonb, platb**, …)
- kk가 3*nsig+2인 경우 call s2g0(*pc*, **gp**, …)
- kk가 3*nsig+1인 경우 call s2g0(*rc*, **gmtns**, …)

호출되는 서브루틴 s2g0, s2gvec, s2grad은 기울임 글꼴로 표시된 변수를 입력으로 받아들여 굵게 표시된 변수를 해당 루틴의 결과로 리턴한다. 루프의 매 반복마다 여러 번의 IF 구문에 의해 코드 흐름이 결정되는 구조를 가지고 있어 효율적이지 못하고 따라서 많은 개선의 여지를 가지고 있다.

ssi.x에서 hotspot이 되는 몇몇 부분들은 모두 s2g0, s2grad, s2gvec과 동일한 형태의 서브루틴을 반복 호출해 계산하고 있다. 이런 이유로 이곳을 최적화/병렬화한 방법이 다른 유사한 부분에 모두 동일하게 반복 적용되고 있으므로 이부분에 대한 이해가 ssi.x 최적화/병렬화를 이해하는데 핵심적인 역할을 한다.

< 최적화/병렬화 코드 >

```
127                !$OMP PARALLEL DO PRIVATE(llps,llpe,me,mo,
                  * mme,mmo,mme0,mmo0)
128                  do np=0,nthds-1
129                    llps=lse(1,np)
130                    llpe=lse(2,np)
131                    mme=llse(np)+1
132                    mmo=llso(np)+1
133                    mme0=llsbe(np)+1
134                    mmo0=llsbo(np)+1
135                  do k=1,nsig
139                  me=llse(np)
140                  mo=llso(np)
```

```
      do ll=llps,llpe
         lle=ll
         do m=1,lene(ll)
            tse1(me+m)=qc(lle,k)
            lle=lle+2+4*(jcap+2-2*m)
         enddo
         llo=ll+2*(jcap+1)
         do m=1,leno(ll)
1           tso1(mo+m)=qc(llo,k)
            llo=llo-2+4*(jcap+2-2*m)
         enddo
         me=me+lene(ll)
         mo=mo+leno(ll)
      enddo
      call s2g0(tse1(mme),tso1(mmo),gq(1,1,k),jcap,
     * nlon,nlath,plnee(mme0),plnoo(mmo0),
     * lene,leno,lenej,lenoj,llps,llpe)
      enddo
      me=llse(np)
      mo=llso(np)
      do ll=llps,llpe
         lle=ll
         do m=1,lene(ll)
            tse1(me+m)=pc(lle)
            tse2(me+m)=-del2(lle)*pc(lle)
            lle=lle+2+4*(jcap+2-2*m)
         enddo
         llo=ll+2*(jcap+1)
         do m=1,leno(ll)
            tso1(mo+m)=pc(llo)
            tso2(mo+m)=-del2(llo)*pc(llo)
            llo=llo-2+4*(jcap+2-2*m)
         enddo
         me=me+lene(ll)
         mo=mo+leno(ll)
      enddo
      call s2g0(tse1(mme),tso1(mmo),gp,jcap,nlon,nlath,
     * plnee(mme0),plnoo(mmo0),
     * lene,leno,lenej,lenoj,llps,llpe)
      call s2grad(tse2(mme),tso2(mmo),plonb,platb,jcap,
     * nlon,nlath,qlnee(mme0),qlnoo(mmo0), rlnee
     * (mme0),rlnoo(mmo0),lene,leno,lenej,lenoj,llps,llpe)
      me=llse(np)
      mo=llso(np)
      do ll=llps,llpe
```

```
195                                 lle=ll
196                                 do m=1,lene(ll)
197                                     tse1(me+m)=tc(lle,k)
198                                     lle=lle+2+4*(jcap+2-2*m)
199                                 enddo
200                                 llo=ll+2*(jcap+1)
201                                 do m=1,leno(ll)
202                                     tso1(mo+m)=tc(llo,k)
203                                     llo=llo-2+4*(jcap+2-2*m)
204                                 enddo
205                                 me=me+lene(ll)
206                                 mo=mo+leno(ll)
207                               enddo
215                               call s2g0(tse1(mme),tso1(mmo),gt(1,1,k),jcap,nlon,
216                              * nlath,plnee(mme0),plnoo(mmo0),
217                              * lene,leno,lenej,lenoj,llps,llpe)
218                               enddo
221                               do k=1,nsig
225                               me=llse(np)
226                               mo=llso(np)
227                               do ll=llps,llpe
228                                   lle=ll
229                                   do m=1,lene(ll)
230                                       tse1(me+m)=zc(lle,k)
231                                       tse2(me+m)=dc(lle,k)
232                                       lle=lle+2+4*(jcap+2-2*m)
233                                   enddo
234                                   llo=ll+2*(jcap+1)
235                                   do m=1,leno(ll)
236                                       tso1(mo+m)=zc(llo,k)
237                                       tso2(mo+m)=dc(llo,k)
238                                       llo=llo-2+4*(jcap+2-2*m)
239                                   enddo
240                                   me=me+lene(ll)
241                                   mo=mo+leno(ll)
242                               enddo
243                                 call s2gvec(tse1(mme),tso1(mmo),tse2(mme),
244                              * tso2(mmo),gu(1,1,k),gv(1,1,k),
245                              * jcap,nlon,nlath,
246                              * qlnee(mme0),qlnoo(mmo0),rlnee(mme0),
247                              * rlnoo(mmo0),lene,leno,lenej,lenoj,llps,llpe)
248                               call s2g0(tse1(mme),tso1(mmo),vortb(1,1,k),jcap,
249                              * nlon,nlath,plnee(mme0),plnoo(mmo0),
250                              * lene,leno,lenej,lenoj,llps,llpe)
251                               enddo
```

```
  256                                 me=llse(np)
  257                                 mo=llso(np)
  258                                 do ll=llps,llpe
  259                                    lle=ll
  260                                    do m=1,lene(ll)
  261                                       tse1(me+m)=rc(lle)
  262                                       lle=lle+2+4*(jcap+2-2*m)
  263                                    enddo
  264                                    llo=ll+2*(jcap+1)
  265                                    do m=1,leno(ll)
  266                                       tso1(mo+m)=rc(llo)
  267                                       llo=llo-2+4*(jcap+2-2*m)
  268                                    enddo
  269                                    me=me+lene(ll)
  270                                    mo=mo+leno(ll)
  271                                 enddo
  272                                  mme=llse(np)+1
  273                                  mmo=llso(np)+1
  274                                  mme0=llsbe(np)+1
  275                                  mmo0=llsbo(np)+1
  276                              call s2g0(tse1(mme),tso1(mmo),gmtns,jcap,nlon,nlath,
  277                                 * plnee(mme0),plnoo(mmo0),
  278                                 * lene,leno,lenej,lenoj,llps,llpe)
  284                                  me=llse(np)
  285                                  mo=llso(np)
  286                                  do ll=llps,llpe
  287                                     lle=ll
  288                                     do m=1,lene(ll)
  289                                        tse1(me+m)=dc(lle,k)
  290                                        lle=lle+2+4*(jcap+2-2*m)
  291                                     enddo
  292                                     llo=ll+2*(jcap+1)
  293                                     do m=1,leno(ll)
  294                                        tso1(mo+m)=dc(llo,k)
  295                                        llo=llo-2+4*(jcap+2-2*m)
  296                                     enddo
  297                                     me=me+lene(ll)
  298                                     mo=mo+leno(ll)
  299                                  enddo
  300                                                                          call
s2g0(tse1(mme),tso1(mmo),divb(1,1,k),jcap,nlon,
  301                                 * nlath,plnee(mme0),plnoo(mmo0),
  302                                 * lene,leno,lenej,lenoj,llps,llpe)
  303                                  enddo
  304                                  enddo
```

② 최적화/병렬화 코드의 구조

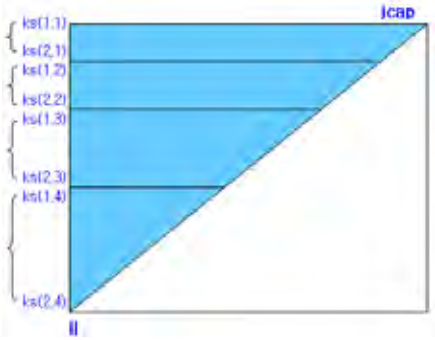
루프 인덱스 kk에 의해 nsig*3 +3회 반복되며 IF 구문으로 각 kk에서 실행부분을 판단하고 있다.

- k를 1 ~ nsig 반복하면서 call s2g0(*tse1, tso1,* **gq(1,1,k)**, …): gq 계산
- call s2g0(*tse1, tso1*, **gp**, …) : gp 계산
- call s2grad(*tse2, tso2*, **plonb, platb**, …): plonb, platb 계산
- k를 1 ~ nsig 반복하면서 call s2g0*(tse1, tso1,* **gt(1,1,k)**, …): gt 계산
- k를 1 ~ nsig 반복하면서 call s2gvec(*tse1, tso1, tse2, tso2,* **gu(1,1,k), gv(1,1,k),** …), call s2g0(*tse1, tso1*, **vortb(1,1,k)**, …): gu, gv, vortb 계산
- call s2g0(*tse1, tso1*, **gmtns**, …): gmtns 계산
- k를 1 ~ nsig 반복하면서 call s2g0(*tse1, tso1,* **divb(1,1,k),** …): divb 계산

original 코드에서 복잡한 IF 구문에 의해 흐름이 결정되는 것을 모두 풀어 코드 성능을 높이고 있다. 각 서브루틴 s2g0, s2gvec, s2grad의 입력 데이터 zc, dc, tc, qc 등은 모두 even 항(tse1, tse2)과 odd 항(tso1, tso2)으로 미리 구분해 넣고 있다. 아울러 또 다른 입력 데이터인 pln도 even 항(plnee)과 odd 항(plnoo)으로 나누어 넣고 있다.

③ s2g0 등의 서브루틴에서 수행되는 계산은 앞서 언급한 바와 같이 아래와 같은 데이터를 처리하게 된다. 이 계산을 그림에서와 같이 명시적인 블록분할을 해서 병렬화를 진행한 것이다. 코드에서는 각 스레드별로 계산할 영역을 llps, llpe로 지정하였다. np 스레드는 그림에서의 ks(1,np)(=llps)부터 ks(2,np)(=llpe)까지의 계산을

담당하게 된다.



< 그림 I.8 병렬화와 데이터 구조 >

④ 전체 계산을 do np=0,nthds-1로 묶고 OpenMP 병렬 지시어를 넣어 각 스레드 별로 한 번의 np에 대한 계산을 담당하도록 병렬화 하였다. 여기서 nthds는 스레드 개수를 나타낸다.

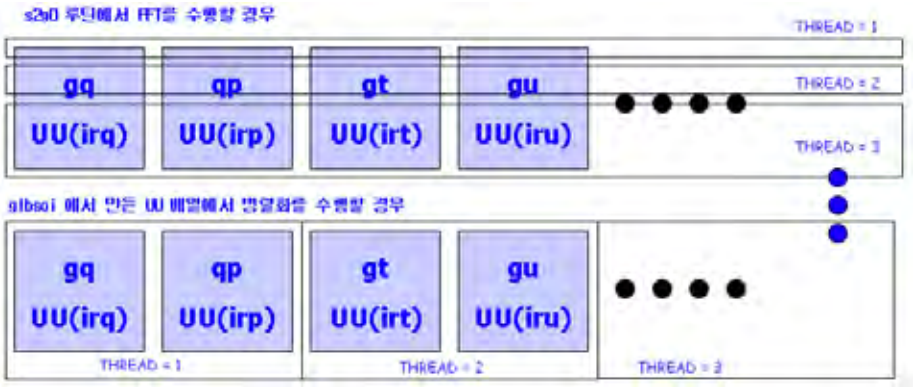
< 최적화 코드 >

```
309                    !$OMP PARALLEL DO PRIVATE(j,jr,tt)
310                          do k=1,6*nsig+4
311                          call transpose(tt,lonf_+2,gq(1,1,k),2*latg2_+1,
312                         & 2*nlath+1,2*(jcap+1))
313                              do j=1,nlath
314                               jr=2*nlath+1-j
315                               do ll=2*(jcap+1)+1,nlon+2
316                                   tt(ll,j )=0.
317                                   tt(ll,jr)=0.
318                               end do
319                              end do
320                        call crfft_d(tt,nlon+2,tt,nlon+2,nlon,2*nlath,isign,scale)
321                          call transpose(gq(1,1,k),2*latg2_+1,tt,lonf_+2,
322                         & nlon+2,2*nlath)
323                          do ll=1,nlon+2
324                              j=2*nlath+1
325                              gq(j,ll,k)=0.
326                          enddo
```

327	enddo

⑤ original 코드에서의 서브루틴 s2g0, s2grad, s2gvec은, 그 내부에서 얻어지는 gq, gt, gu, gp, gv, … 등의 데이터에 대해 서브루틴 fft99m을 호출해 FFT를 수행하게 된다(s2g0 참조). 즉, 각 서브루틴의 결과로 리턴되는 gq, gp, gu, gv 등의 데이터는 모두 FFT를 수행한 결과이다. 그러나, 최적화/병렬화 코드에서의 서브루틴 s2g0, s2grad, s2gvec에서는 각 데이터에 대해 따로 FFT를 수행하지 않는다(s2g0.f 참조). 마지막에 최적화된 FFT 라이브러리 루틴인 crfft_d를 호출해 한꺼번에 FFT를 수행하고 있다.

⑥ 최적화/병렬화 코드에서 서브루틴 별로 FFT를 따로 수행하지 않은 이유는 아래 그림과 같이 데이터를 행방향으로 블록 분할해 병렬화했기 때문이다. 행방향 블록 분할된 데이터에 대해 열방향의 1차원 FFT를 수행할 수 없으므로 최적화 코드에서는 마지막 단계에서 한꺼번에 FFT를 수행하고 있다. 또한 이 단계에서 gq, gp, gt, gu, … 등의 데이터를 묶어 하나의 1차원 배열 uu로 처리함으로서 FFT 계산의 병렬화 효율성을 높이고 있다.



< 그림 I.9 UU 배열의 병렬화 >

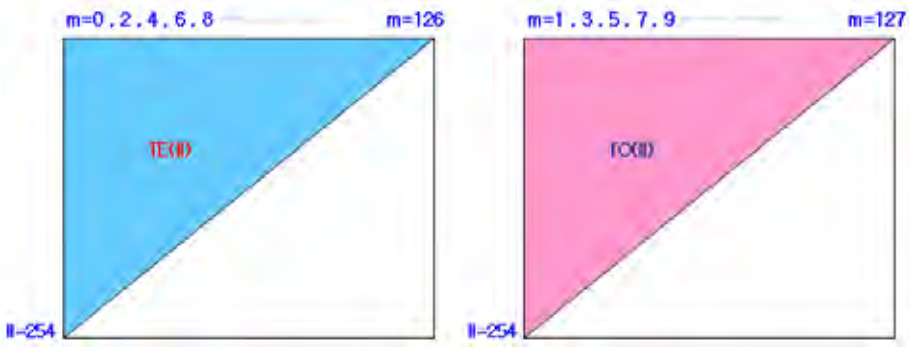
6.4 s2g0.f

inguessv.f에서 호출하는 서브루틴은 s2g0, s2grad, s2gvec 이다. 이 세 개의 루틴은 내부 구조가 거의 유사하며, 따라서 최적화/병렬화 역시 동일한 방법으로 수행되고 있다. 이에 이곳에서는 s2g0.f에 대해서만 original 코드와 최적화/병렬화 코드를 비교 분석해 본다.

< Original 코드 >

```
54                          do itmp=1,(2*nlath+1)*(nlon+2)
55          706               t(itmp,1)=0.
56                          enddo
57            1             do j=1,nlath
58                            jr=2*nlath+1-j
59                            ii0=0
61                              do itmp=1,2*jcap+2
62           36                   te(itmp)=0.
63                              enddo
65                            do itmp=1,2*jcap+2
66           39                   to(itmp)=0.
67                            enddo
68            4               do m=0,jcap,2
69           14                 do ll=1,2*(jcap+1-m)
70         4542                   te(ll)=te(ll)+pln(ii0+ll,j)*ts(ii0+ll)
71                              end do
72           17                 if(m.lt.jcap) then
73                                ii0=ii0+2*(jcap+1-m)
74           13                   do ll=1,2*(jcap-m)
75         4543                     to(ll)=to(ll)+pln(ii0+ll,j)*ts(ii0+ll)
76                              end do
77            1                 ii0=ii0+2*(jcap-m)
78                              end if
79            3               end do
83                            do ll=1,2*(jcap+1)
84          103                 t(j,ll)=te(ll)+to(ll)
85           76                 t(jr,ll)=te(ll)-to(ll)
86                            end do
87                          end do
91                          lot=nlath*2
92                          nlax=lot+1
93                          call fft99m (t,work,trigs,ifax,nlax,1,nlon,lot,1)
```

① dummy 인수 pln과 ts의 배열에서 아래 그림과 같은 형태의 데이터를 even 항과 odd 항으로 분리해 읽어 들인다. 이를 위해 복잡한 루프 구조와 더불어 배열 pln과 ts의 인덱스가 ii0+ll로 계산되고 있다.



< 그림 I.10 TE 배열과 TO 배열의 구조 >

② j 루프가 밖에서 ll 루프가 안에서 반복 되지만, 배열 t(j,ll)에 대한 접근이 불연속적이다.

③ te+to를 dummy 배열 t에 할당한 후 배열 t에 대한 FFT를 수행하고 있다. 이 t가 이 서브루틴의 리턴 결과가 된다.

< 최적화/병렬화 코드>

```
 31                          je=nlath-mod(nlath,12)
  32                          mme=0
  33                          mmo=0
  34                          mme0=0
  35                          mmo0=0
  37                          do ll=llps,llpe
  38              1               do j=1,je,12
  39                                  jr=2*nlath+1-j
  40                                   me0=mme0+(j-1)*lene(ll)
  41                                   me1=me0+lene(ll)
  42                                   me2=me1+lene(ll)
  43                                   me3=me2+lene(ll)
```

```
44                                  me4=me3+lene(ll)
45                                  me5=me4+lene(ll)
46                                  me6=me5+lene(ll)
47                                  me7=me6+lene(ll)
48                                  me8=me7+lene(ll)
49                                  me9=me8+lene(ll)
50                                  me10=me9+lene(ll)
51                                  me11=me10+lene(ll)
52                                  tte0=0.
53                                  tte1=0.
54                                  tte2=0.
55                                  tte3=0.
56                                  tte4=0.
57                                  tte5=0.
58                                  tte6=0.
59                                  tte7=0.
60                                  tte8=0.
61                                  tte9=0.
62                                  tte10=0.
63                                  tte11=0.
64                                  do m=1,lene(ll)
65                50                    tte0=tte0+plnee(me0+m)*tse(mme+m)
66                                      tte1=tte1+plnee(me1+m)*tse(mme+m)
67                 2                    tte2=tte2+plnee(me2+m)*tse(mme+m)
68                13                    tte3=tte3+plnee(me3+m)*tse(mme+m)
69                 1                    tte4=tte4+plnee(me4+m)*tse(mme+m)
70                20                    tte5=tte5+plnee(me5+m)*tse(mme+m)
71                 7                    tte6=tte6+plnee(me6+m)*tse(mme+m)
72                20                    tte7=tte7+plnee(me7+m)*tse(mme+m)
73                                      tte8=tte8+plnee(me8+m)*tse(mme+m)
74                23                    tte9=tte9+plnee(me9+m)*tse(mme+m)
75                                  tte10=tte10+plnee(me10+m)*tse(mme+m)
76                                  tte11=tte11+plnee(me11+m)*tse(mme+m)
77                                  enddo
78                                  mo0=mmo0+(j-1)*leno(ll)
79                                  mo1=mo0+leno(ll)
80                                  mo2=mo1+leno(ll)
81                                  mo3=mo2+leno(ll)
82                                  mo4=mo3+leno(ll)
83                                  mo5=mo4+leno(ll)
84                                  mo6=mo5+leno(ll)
85                                  mo7=mo6+leno(ll)
86                                  mo8=mo7+leno(ll)
87                                  mo9=mo8+leno(ll)
88                                  mo10=mo9+leno(ll)
```

```
 89                                  mo11=mo10+leno(ll)
 90                                  tto0=0.
 91                                  tto1=0.
 92                                  tto2=0.
 93                                  tto3=0.
 94                                  tto4=0.
 95                                  tto5=0.
 96                                  tto6=0.
 97                                  tto7=0.
 98                                  tto8=0.
 99                                  tto9=0.
100                                  tto10=0.
101                                  tto11=0.
102                                  do m=1,leno(ll)
103                 24                   tto0=tto0+plnoo(mo0+m)*tso(mmo+m)
104                  2                   tto1=tto1+plnoo(mo1+m)*tso(mmo+m)
105                  2                   tto2=tto2+plnoo(mo2+m)*tso(mmo+m)
106                 18                   tto3=tto3+plnoo(mo3+m)*tso(mmo+m)
107                 11                   tto4=tto4+plnoo(mo4+m)*tso(mmo+m)
108                 21                   tto5=tto5+plnoo(mo5+m)*tso(mmo+m)
109                  9                   tto6=tto6+plnoo(mo6+m)*tso(mmo+m)
110                 16                   tto7=tto7+plnoo(mo7+m)*tso(mmo+m)
111                                      tto8=tto8+plnoo(mo8+m)*tso(mmo+m)
112                  7                   tto9=tto9+plnoo(mo9+m)*tso(mmo+m)
113              4                      tto10=tto10+plnoo(mo10+m)*tso(mmo+m)
114              3                      tto11=tto11+plnoo(mo11+m)*tso(mmo+m)
115                                  enddo
116                                  t(j     ,ll)=tte0+tto0
117                 13               t(jr    ,ll)=tte0-tto0
118                                  t(j+1 ,ll)=tte1+tto1
119                                  t(jr-1,ll)=tte1-tto1
120                                  t(j+2 ,ll)=tte2+tto2
121                  2               t(jr-2,ll)=tte2-tto2
122                                  t(j+3 ,ll)=tte3+tto3
123                                  t(jr-3,ll)=tte3-tto3
124                                  t(j+4 ,ll)=tte4+tto4
125                                  t(jr-4,ll)=tte4-tto4
126                                  t(j+5 ,ll)=tte5+tto5
127                                  t(jr-5,ll)=tte5-tto5
128                                  t(j+6 ,ll)=tte6+tto6
129                                  t(jr-6,ll)=tte6-tto6
130                                  t(j+7 ,ll)=tte7+tto7
131                  1               t(jr-7,ll)=tte7-tto7
132                                  t(j+8 ,ll)=tte8+tto8
133                  1               t(jr-8,ll)=tte8-tto8
```

```
134                                     t(j+9 ,ll)=tte9+tto9
135                                     t(jr-9,ll)=tte9-tto9
136                                     t(j+10 ,ll)=tte10+tto10
137                                     t(jr-10,ll)=tte10-tto10
138                                     t(j+11 ,ll)=tte11+tto11
139                                     t(jr-11,ll)=tte11-tto11
140                                 end do
141                                 do j=je+1,nlath
142                                    jr=2*nlath+1-j
143                                     me0=mme0+(j-1)*lene(ll)
144                                     tte0=0.
145                                     do m=1,lene(ll)
146                                        tte0=tte0+plnee(me0+m)*tse(mme+m)
147                                     enddo
148                                     mo0=mmo0+(j-1)*leno(ll)
149                                     tto0=0.
150                                     do m=1,leno(ll)
151                                        tto0=tto0+plnoo(mo0+m)*tso(mmo+m)
152                                     enddo
153                                     t(j ,ll)=tte0+tto0
154                                     t(jr,ll)=tte0-tto0
155                                 end do
156                                     mme=mme+lene(ll)
157                                     mmo=mmo+leno(ll)
158                                     mme0=mme0+lenej(ll)
159               1                     mmo0=mmo0+lenoj(ll)
160                             end do
```

④ 미리 even 항과 odd 항으로 구분된 plnee, tse, plnoo, tso 입력을 받게 수정되었다. original 코드에서 pln과 ts 배열의 even 항과 odd 항을 구분해 읽어 들이기 위해 복잡한 루프 반복이 간략하게 되었다. 이 과정에서 역 삼각형 형태로 읽어들이게 될 데이터를 결정하기 위해 앞서 glbsoi.f에서 정의한 lene, leno를 이용하고 있다.

⑤ j루프와 ll루프 순서를 바꿔 배열 t(j,ll)에 대한 접근을 연속적이 되도록 하였다.

⑥ 루프 ll에 대해 병렬화 되었다. 각 스레드는 할당받은 영역 llps부터 llpe까지 계산을 담당하게 된다(inguessv.f 참조).

⑦ 루프 내에서 수행되는 계산은 a+b*c의 형태로 FMA 연산이다. 이러한 연산에서는 계산 수행을 위한 메모리 load/store 회수 대비 FMA 연산수가 높을수록 더 나은 성능을 보여 주게 된다. 최적화 코드에서는 이 load/store 회수 대비 FMA 연산수를 높이기 위해 바깥쪽 루프를 depth 12로 unrolling해 코드 성능을 높이고 있다. 일반적으로 Unrolling depth를 정할 때 unrolling 하면서 사용되는 변수들이 POWER4 프로세서의 레지스터 수(32)를 최대한 사용하도록 정하게 되는데, 이 코드에서는 추가적으로 변수 nlath의 약수가 되도록 하여 마지막 부분에 있는 추가적인 연산부분을 수행하지 않도록 하였다.

6.5 qoper.f, qtoper.f, htoper.f, qtoper.f 최적화 및 병렬화

ssi.x의 실행시간에서 많은 부분을 차지하는 루틴들로 qoper, qtoper, htoper, qtoper 등이 있다. 이 루틴들은 다시 g2s0, grad2s, tg2s0, tgrad2s, … 등의 루틴을 호출해 대부분의 계산을 수행하고 있는데 이 루틴들의 구조가 앞서 살펴본 inguessv.f에서 호출해 사용하는 s2g0와 매우 유사하다. 루틴 g2s0는 s2g0의 역함수이고 ts2g0는 s2g0의 계산에서 FFT의 수행을 마지막이 아닌 처음 시작단계에서 수행한다는 정도의 차이만 있다.

qoper, qtoper, htoper, qtoper 등의 최적화의 핵심은 이곳에서 호출해 사용하는 g2s0, tg2s0, ts2g0, … 등의 최적화에 있으며 그 기법은 이미 살펴본 s2g0의 최적화 과정과 유사하다. 이런 이유로, 이곳에서는 g2s0에 적용된 최적화 과정을 s2g0와 비교해 간단히 알아 볼 것이고, 또 최적화된 g2s0를 호출하는 qoper 루틴이 어떻게 바뀌었는지에 대해 간단히 알아본다. 우선 아래는 qoper, qtoper, htoper, qtoper 루틴에서 호출해 사용하는 g2s0 등의 루틴에 대해 정리한 것이다.

- qoper가 호출하는 서브루틴 : g2s0, grad2s

- qtoper가 호출하는 서브루틴 : tg2s0, tgrad2s
- htoper가 호출하는 서브루틴 : ts2grad, ts2g0, ts2gvec
- hoper가 호출하는 서브루틴 : s2grad, s2g0, s2gvec

< Original 코드 – qoper.f >

```
100                          do j=1,2*nlath(=96)
101                            do itmp=1,nlon(=384)
102          45                      coriolis(j,itmp)=2.*omega*sin(rlats(j))
103                            end do
107                           do k=1,nsig(=26)
108             1              do i=1,nlon
109         19870                uw(j,i,k)=-gascon*t(j,i,k)*rplons(j,i)
110                        *           -gascon*rts(j,i,k)*plon(j,i)
111                        *           +rvs(j,i,k)*vort(j,i,k)
112                        *           +v(j,i,k)*(rvorts(j,i,k)+coriolis(j,i))
113          5358                  vw(j,i,k)=-rus(j,i,k)*vort(j,i,k)
114                        *           -gascon*t(j,i,k)*rplats(j,i)
115                        *           -gascon*rts(j,i,k)*plat(j,i)
116                        *            -u(j,i,k)*(rvorts(j,i,k)+coriolis(j,i))
117          2457                  vort(j,i,k)=u(j,i,k)*rus(j,i,k)
118                        *           +v(j,i,k)*rvs(j,i,k)
119                            end do
120                           end do
121                           do k=1,nsig
122             3              do l=1,nsig
123           449                do i=1,nlon
124         34347                  vort(j,i,k)=vort(j,i,k)+eaccel*a3(k,l)*t(j,i,l)
125                            end do
126             3              end do
127                           end do
128             1            end do
```

① 우선 qoper의 original 코드에서 루틴 g2s0, grad2s을 호출하기 이전에 이 루틴들에 입력으로 들어가게될 배열 uw, vw, vort에 값을 할당하는 부분이다. 루프의 반복에 의한 각 배열에 대한 접근이 많은 부분 불연속적으로 이루어지게 되어있어 g2s0, grad2s의 호출과 상관없이 qoper 자체의 실행 속도를 저하시키는 원인이

되고 있다.

< 최적화/병렬화 코드 – qoper.f >

```
111                                 do j=1,2*nlath
112                                     coriolis(j)=2.*omega*sin(rlats(j))
113                                 enddo
117                 !$OMP PARALLEL DO PRIVATE(is,ie)
118                             do np=0,nthds-1
119                                 is=ise(1,np)
120                                 ie=ise(2,np)
121                             do l=1,nsig
122                                   do i=is,ie
123                                    do j=1,2*nlath
124                 652                 ttt(l,j,i)=t(j,i,l)
125                                    end do
126                  10                end do
127                             end do
128                             do k=1,nsig
130                                 do i=is,ie
131                                   do j=1,2*nlath
132                                    uw(j,i,k)=-gascon*t(j,i,k)*rplons(j,i)
133                               *                  -gascon*rts(j,i,k)*plon(j,i)
134                               *                  +rvs(j,i,k)*vort(j,i,k)
135                               *                  +v(j,i,k)*(rvorts(j,i,k)+coriolis(j))
136                                    vw(j,i,k)=-rus(j,i,k)*vort(j,i,k)
137                               *                  -gascon*t(j,i,k)*rplats(j,i)
138                               *                  -gascon*rts(j,i,k)*plat(j,i)
139                               *                  -u(j,i,k)*(rvorts(j,i,k)+coriolis(j))
140                                    vortw(j,i,k)= u(j,i,k)*rus(j,i,k)
141                               *                     +v(j,i,k)*rvs(j,i,k)
142                 1404              end do
143                  17               do j=1,2*nlath
144                  35                t0=vortw(j,i,k)
145                                    do l=1,nsig
148                 2682                t0=t0+eaccel*a3t(l,k)*ttt(l,j,i)
149                                    end do
150                                    vortw(j,i,k)=t0
151                  117              end do
152                   2              end do
153                             end do
154                             end do
```

② 최적화 코드에서는 불필요하게 2차원으로 정의된 변수 coriolis(j,itmp)을 1차원 배열 coriolis(j)로 변경하였다.

③ Original 코드에서 루프가 j, k, i 순으로 반복되어 uw(j,i,k), vw(j,i,k), t(j,i,k), rus(j,i,k), … 등과 같은 배열에 대한 접근이 불연속적으로 수행된다. 이에 j 루프를 가장 안쪽에서 반복되도록 해 메모리 접근이 연속적으로 이루어지도록 하였다. 이 과정에서 j,k,l,i 로 반복 계산되는 아래쪽 vort 계산은 k, i, j, l로 반복된다. 이때 이곳에서 다시 접근되는 t(j,i,l)에서 불연속 접근이 발생하므로 t(j,i,l)를 ttt(l,j,i)로 새롭게 할당해 사용하고 있다.

④ 1에서 nlon까지 반복되는 루프 계산을 병렬화 하였다. 병렬 코드에서 np 스레드는 ise(1,np)에서 ise(2,np)까지의 계산을 담당하게 된다.

< Original 코드 – qoper.f >

```
135                         do k=1,nsig
136                          call g2s0(ts(1,k),vort(1,1,k),jcap,nlon,nlath,
137                     *           wgts,pln,trigs,ifax)
138                        call grad2s(ds(1,k),uw(1,1,k),vw(1,1,k),jcap,
139                     * nlon,nlath,qln,rln,trigs,ifax,wgts,del2)
140                          do itmp=1,nc
141              13                    ts(itmp,k)=del2(itmp)*ts(itmp,k)
142                          end do
143                          do itmp=1,nc
144               8                    ds(itmp,k)=ds(itmp,k)+ts(itmp,k)
145                          end do
146                         end do
```

⑤ 앞서 만들어진 배열 vort, uw, vw에 대해 g2s0, grad2s의 계산을 수행하는 부분이다. g2s0는 배열 ts를 grad2s는 배열 ds를 결과로 리턴하고 있다. 이렇게 얻어진 ts, ds를 이용해 ts*del2 + ds = ds의 최종결과를 얻고 있다.

< 최적화/병렬화 코드 – qoper.f >

```
158                         isign=1
159                         scale=1./float(nlon)
160                   !$OMP PARALLEL DO PRIVATE(tt)
161                         do k=1,3*nsig
162                         call transpose(tt,lonf_+2,uw(1,1,k),2*latg2_+1,
163                        * 2*nlath+1,nlon+2)
164                       call rcfft_d(tt,nlon+2,tt,nlon+2,nlon,2*nlath,isign,scale)
165                         call transpose(uw(1,1,k),2*latg2_+1,tt,lonf_+2,
166                        * nlon+2,2*nlath+1)
167                         enddo
168                   c
169                   !$OMP PARALLEL DO PRIVATE(me,mo,llps,llpe)
170                         do np=0,nthds-1
171                            llps=lse(1,np)
172                            llpe=lse(2,np)
173                   c
174                            me=llse(np)+1
175                            mo=llso(np)+1
176                         do ll=llps,llpe
177                           call g2s1(dse(me,1),dso(mo,1),vortw(1,ll,1),
178                        * plnte(1,me),plnto(1,mo),wgts,
179                        * jcap,nlath,nlon,
180                        * lene(ll),leno(ll),nsig)
181                            me=me+lene(ll)
182                            mo=mo+leno(ll)
183                         enddo
184                            me=llse(np)+1
185                            mo=llso(np)+1
186                         do ll=llps,llpe,2
187                            me1=me+lene(ll)
188                            mo1=mo+leno(ll)
189                           call grad2s1(dse(me,k),dso(mo,k),dse(me1,k),
190                        & dso(mo1,k),uw(1,ll,k),vw(1,ll,k),
191                        & jcap,nlath,nlon,
192                        & qlnte(1,me),qlnto(1,mo),rlnte(1,me),rlnto(1,mo),
193                       & qlnte(1,me1),qlnto(1,mo1),rlnte(1,me1),rlnto(1,mo1),
194                        & wgts,del2e(me),del2o(mo),del2e(me1),del2o(mo1),
195                        & lene(ll),leno(ll),nsig)
196                            me=me+2*lene(ll)
197                            mo=mo+2*leno(ll)
198                         enddo
```

```
199                                   end do
```

⑥ Original 코드에서 루틴 g2s0, grad2s는 먼저 FFT를 수행하고 그 결과를 even 항과 odd 항으로 나눈 후 이를 이용해 ts를 계산하게 된다(아래 g2s0.f 참조). 최적화 코드에서는 g2s0, grad2s의 최적화 버전인 g2s1, grad2s1의 내부에서 FFT를 수행하지 않고, 이 루틴들을 호출하기에 앞서 미리 FFT를 수행하고 있다. 여기서 사용된 dummy 배열 uw, vw, vortw는 3차원 배열 ww의 ww(:,:,1:nsig), ww(:,:,nsig+1:2*nsig), ww(:,:,2*nsig+1:3*nsig)가 각각 전달된 것이다. 따라서 코드에서는 uw, vw, vortw 각각에 대한 FFT를 수행하지 않고 k를 1부터 3*nsig 만큼 변화시키며 uw(1,1,k)에 대한 FFT를 병렬로 수행하고 있다.

⑦ 다음 단계는 루틴 g2s1을 이용해 vortw로부터 original 코드에서의 ts에 대응하는 dse, dso를 얻는다. Original 코드에서 even 항과 odd 항이 반복적으로 나열된 배열 ts를 얻는 것과 달리 최적화 코드에서는 even 항은 dse로 odd 항은 dso로 각각 분리해 할당하고 있다. 이렇게 얻어진 dse, dso를 uw, vw, del2와 같이 루틴 grad2s1에 넣어 최종결과가 되는 dse, dso를 계산한다. Original 코드에서 ts, ds 계산 이후 다시 두 번의 루프를 이용해 계산되는 ts*del2 + ds의 계산을 grad2s1 안으로 넣어 불필요하게 반복되는 계산을 줄이고 있다(grad2s1.f 참조).

⑧ 이 과정에서 1부터 nsig까지 반복되는 k루프가 루틴 g2s1과 grad2s1의 안에서 처리되도록 하였고, 병렬 루프 II을 루틴 밖으로 꺼내어 처리하고 있는데 이런 것들은 앞서 살펴보았던 s2g0, s2grad의 최적화/병렬화와 다른 점이다.

6.6 g2s0.f, g2s1.f, grad2s1.f

< Original 코드 – g2s0.f >

```
 58                                lot=nlath*2
 59                                nlax=lot+1
 60                                call fft99m (t,work,trigs,ifax,nlax,1,nlon,lot,-1)
 62                                do itmp=1,(jcap+1)*(jcap+2)
 63              23                  ts(itmp)=0.
 64                                enddo
 65                                do j=1,nlath
 66                                 jr=2*nlath+1-j
 67                          c---------- separate even and odd parts
 68                                 do ll=1,2*jcap+2
 69              70                  te(ll)=(t(j,ll)+t(jr,ll))*wgts(j)
 70              16                  to(ll)=(t(j,ll)-t(jr,ll))*wgts(j)
 71                                 end do
 72                                 ii0=0
 73                                 do m=0,jcap,2
 74               4                  do ll=1,2*(jcap+1-m)
 75             598                   ts(ii0+ll)=ts(ii0+ll)+pln(ii0+ll,j)*te(ll)
 76                                 end do
 77               2                 if(m.lt.jcap) then
 78                                  ii0=ii0+2*(jcap+1-m)
 79               1                  do ll=1,2*(jcap-m)
 80             590                   ts(ii0+ll)=ts(ii0+ll)+pln(ii0+ll,j)*to(ll)
 81                                 end do
 82                                 ii0=ii0+2*(jcap-m)
 83                                end if
 84               1                end do
 85                               end do
```

① g2s0는 s2g0의 역함수 이다. s2g0가 수행하는 계산은 입력으로 받아오는 배열 ts와 pln으로부터 각각 te와 to를 얻고, 이 두 배열을 합친 배열 t에 대해 FFT를 수행해 최종 결과를 내는 것이었다. g2s0는 이 과정이 역으로 수행된다. 입력으로 들어오는 dummy 배열 t에 대해 역FFT를 수행하고 이것으로부터 te와 to를 얻는다. 이렇게해서 얻어진 te, to를 이용해 배열 ts를 얻는 과정을 위에서 볼 수 있다.

② 배열 ts, pln은 even 항과 odd 항이 함께 들어 있는 배열로 even, odd 항을 제자리에 넣기 위해 복잡한 인덱스 계산을 하고 있다.

< 최적화/병렬화 코드 – g2s1.f >

```
 34                          ke=nsig-mod(nsig,4)
 35                           me=lene-mod(lene,4)
 36                           mo=leno-mod(leno,4)
 38                           do k=1,nsig
 39                               do j=1,nlath
 40                                   jr=2*nlath+1-j
 41          68                       te(j,k)=(t(j,1,k)+t(jr,1,k))*wgts(j)
 42           8                       to(j,k)=(t(j,1,k)-t(jr,1,k))*wgts(j)
 43                               enddo
 44           1                   enddo
 46                           do k=1,ke,4
 47                               do m=1,me,4
 48                                   te00=0.
 49                                   te10=0.
 50                                   te20=0.
 51                                   te30=0.
 52                                   te01=0.
 53                                   te11=0.
 54                                   te21=0.
 55                                   te31=0.
 56                                   te02=0.
 57                                   te12=0.
 58                                   te22=0.
 59                                   te32=0.
 60                                   te03=0.
 61                                   te13=0.
 62                                   te23=0.
 63                                   te33=0.
 64                                 do j=1,nlath
 65           7                         te00=te00+plnte(j,m  )*te(j,k)
 66          23                         te10=te10+plnte(j,m+1)*te(j,k)
 67          22                         te20=te20+plnte(j,m+2)*te(j,k)
 68          23                         te30=te30+plnte(j,m+3)*te(j,k)
 69                                     te01=te01+plnte(j,m  )*te(j,k+1)
 70                                     te11=te11+plnte(j,m+1)*te(j,k+1)
 71                                     te21=te21+plnte(j,m+2)*te(j,k+1)
 72                                     te31=te31+plnte(j,m+3)*te(j,k+1)
 73           9                         te02=te02+plnte(j,m  )*te(j,k+2)
 74          20                         te12=te12+plnte(j,m+1)*te(j,k+2)
 75                                     te22=te22+plnte(j,m+2)*te(j,k+2)
 76          18                         te32=te32+plnte(j,m+3)*te(j,k+2)
 77           2                         te03=te03+plnte(j,m  )*te(j,k+3)
```

```
 78                                te13=te13+plnte(j,m+1)*te(j,k+3)
 79                                te23=te23+plnte(j,m+2)*te(j,k+3)
 80          11                    te33=te33+plnte(j,m+3)*te(j,k+3)
 81                              end do
 82                               ttse(m   ,k)=te00
 83                               ttse(m+1,k)=te10
 84                               ttse(m+2,k)=te20
 85                               ttse(m+3,k)=te30
 86                               ttse(m   ,k+1)=te01
 87                               ttse(m+1,k+1)=te11
 88                               ttse(m+2,k+1)=te21
 89                               ttse(m+3,k+1)=te31
 90                               ttse(m   ,k+2)=te02
 91                               ttse(m+1,k+2)=te12
 92                               ttse(m+2,k+2)=te22
 93                               ttse(m+3,k+2)=te32
 94                               ttse(m   ,k+3)=te03
 95           1                   ttse(m+1,k+3)=te13
 96                               ttse(m+2,k+3)=te23
 97                               ttse(m+3,k+3)=te33
 98                            end do
 99                            do m=me+1,lene
100                               te00=0.
101                               te01=0.
102                               te02=0.
103                               te03=0.
104                              do j=1,nlath
105          1                      te00=te00+plnte(j,m)*te(j,k)
106          3                      te01=te01+plnte(j,m)*te(j,k+1)
107          5                      te02=te02+plnte(j,m)*te(j,k+2)
108          1                      te03=te03+plnte(j,m)*te(j,k+3)
109                              end do
110                               ttse(m,k)=te00
111                               ttse(m,k+1)=te01
112                               ttse(m,k+2)=te02
113                               ttse(m,k+3)=te03
114                            end do
115                         enddo
116                         do k=ke+1,nsig
117                            do m=1,me,4
118                               te00=0.
119                               te10=0.
120                               te20=0.
121                               te30=0.
122                              do j=1,nlath
```

```
123                                 te00=te00+plnte(j,m   )*te(j,k)
124                                 te10=te10+plnte(j,m+1)*te(j,k)
125                                 te20=te20+plnte(j,m+2)*te(j,k)
126                                 te30=te30+plnte(j,m+3)*te(j,k)
127                               end do
128                                 ttse(m   ,k)=te00
129                                 ttse(m+1,k)=te10
130                                 ttse(m+2,k)=te20
131                                 ttse(m+3,k)=te30
132                             end do
133                             do m=me+1,lene
134                                 te00=0.
135                               do j=1,nlath
136                                 te00=te00+plnte(j,m)*te(j,k)
137                               end do
138                                 ttse(m,k)=te00
139                             end do
140                         enddo
142                         do k=1,ke,4
143                             do m=1,mo,4
144                                 to00=0.
145                                 to10=0.
146                                 to20=0.
147           2                     to30=0.
148                                 to01=0.
149                                 to11=0.
150                                 to21=0.
151                                 to31=0.
152                                 to02=0.
153                                 to12=0.
154           1                     to22=0.
155                                 to32=0.
156                                 to03=0.
157                                 to13=0.
158           1                     to23=0.
159                                 to33=0.
160                               do j=1,nlath
161           7                     to00=to00+plnto(j,m   )*to(j,k)
162          12                     to10=to10+plnto(j,m+1)*to(j,k)
163          11                     to20=to20+plnto(j,m+2)*to(j,k)
164          26                     to30=to30+plnto(j,m+3)*to(j,k)
165                                 to01=to01+plnto(j,m   )*to(j,k+1)
166                                 to11=to11+plnto(j,m+1)*to(j,k+1)
167                                 to21=to21+plnto(j,m+2)*to(j,k+1)
168                                 to31=to31+plnto(j,m+3)*to(j,k+1)
```

```
169          11                    to02=to02+plnto(j,m   )*to(j,k+2)
170          24                    to12=to12+plnto(j,m+1)*to(j,k+2)
171                                to22=to22+plnto(j,m+2)*to(j,k+2)
172           7                    to32=to32+plnto(j,m+3)*to(j,k+2)
173                                to03=to03+plnto(j,m   )*to(j,k+3)
174                                to13=to13+plnto(j,m+1)*to(j,k+3)
175                                to23=to23+plnto(j,m+2)*to(j,k+3)
176           9                    to33=to33+plnto(j,m+3)*to(j,k+3)
177                              end do
178                               ttso(m   ,k)=to00
179                               ttso(m+1,k)=to10
180                               ttso(m+2,k)=to20
181                               ttso(m+3,k)=to30
182                               ttso(m   ,k+1)=to01
183                               ttso(m+1,k+1)=to11
184                               ttso(m+2,k+1)=to21
185                               ttso(m+3,k+1)=to31
186                               ttso(m   ,k+2)=to02
187                               ttso(m+1,k+2)=to12
188                               ttso(m+2,k+2)=to22
189           1                   ttso(m+3,k+2)=to32
190                               ttso(m   ,k+3)=to03
191                               ttso(m+1,k+3)=to13
192                               ttso(m+2,k+3)=to23
193                               ttso(m+3,k+3)=to33
194                            end do
195                            do m=mo+1,leno
196                                to00=0.
197                                to01=0.
198                                to02=0.
199                                to03=0.
200                              do j=1,nlath
201           4                    to00=to00+plnto(j,m)*to(j,k)
202           3                    to01=to01+plnto(j,m)*to(j,k+1)
203           7                    to02=to02+plnto(j,m)*to(j,k+2)
204           2                    to03=to03+plnto(j,m)*to(j,k+3)
205                              end do
206                               ttso(m,k)=to00
207                               ttso(m,k+1)=to01
208                               ttso(m,k+2)=to02
209                               ttso(m,k+3)=to03
210                            end do
211                         enddo
212                         do k=ke+1,nsig
213                            do m=1,mo,4
```

```
214                                 to00=0.
215                                 to10=0.
216                                 to20=0.
217                                 to30=0.
218                                do j=1,nlath
219                                   to00=to00+plnto(j,m   )*to(j,k)
220                                   to10=to10+plnto(j,m+1)*to(j,k)
221                                   to20=to20+plnto(j,m+2)*to(j,k)
222                                   to30=to30+plnto(j,m+3)*to(j,k)
223                                end do
224                                 ttso(m   ,k)=to00
225                                 ttso(m+1,k)=to10
226                                 ttso(m+2,k)=to20
227                                 ttso(m+3,k)=to30
228                              end do
229                              do m=mo+1,leno
230                                 to00=0.
231                                do j=1,nlath
232                                   to00=to00+plnto(j,m)*to(j,k)
233                                end do
234                                 ttso(m,k)=to00
235                              end do
236                           enddo
```

③ 최적화/병렬화된 g2s1루틴에서는 입력으로 들어오는 pln을 even 항(plnte), odd항 (plnto)로 미리 분리해 넣고 있다. 또한 결과가 되는 ts역시 even 항과 odd 항을 분리해 ttse와 ttso로 각각 할당하고 있다. 이를 통해 original 코드에서와 같은 복잡한 인덱스 계산을 피할 수 있다.

④ Original 코드에서 j, m, ll 순으로 반복되는 루프를 ll, m, j로 변화시키는 과정은 앞서 s2g0의 최적화/병렬화 과정에서 시도된 것과 동일한 과정이다(s2g0.f 참조). Original 코드의 g2s0는 qoper에서 인덱스 k가 1부터 nsig만큼 반복되며 호출되는 루틴이다. 최적화/병렬화 코드의 g2s1은 이 k루프를 루틴 안으로 넣고 대신 병렬 루프 ll을 밖으로 빼내어 계산하고 있다. 그래서 g2s1에서는 루프 nest는 k, m, j 순으로 구성되어 있다.

⑤ k, m, j 루프 내에서 수행되는 계산은 a+b*c의 형태로 FMA 연산

이다. 일반적으로 계산 수행을 위한 메모리 load/store 회수 대비 FMA 연산수가 높을수록 더 나은 성능을 보여 주게 된다. 최적화 코드에서는 이 load/store 회수 대비 FMA 연산수를 높이기 위해 바깥쪽 루프 m과 k를 각각 depth 4로 unrolling해 결과적으로는 depth 16으로 unrolling한 효과를 얻고 있다.

< 최적화/병렬화 코드 – grad2s1.f >

전체적으로 unrolling depth를 8로 하여 loop를 최적화하였으며, 또한 original 코드에서 g2s0, grad2s의 호출 이후에 반복 계산되는 ts*del2 + ds의 계산을 grad2s1의 내부에서 처리하도록 추가하여 반복 계산을 피할 수 있도록 하였다.

```
 75            do k=1,nsig
 76            do j=1,nlath
 77            jr=2*nlath+1-j
 78     8      ue(1,j,k)=(uu(j,1,k)+uu(jr,1,k))*wgts(j)
 79    13      uo(1,j,k)=(uu(j,1,k)-uu(jr,1,k))*wgts(j)
 80    14      ve(1,j,k)=(vv(j,1,k)+vv(jr,1,k))*wgts(j)
 81            vo(1,j,k)=(vv(j,1,k)-vv(jr,1,k))*wgts(j)
 82    12      ue(2,j,k)=(uu(j,2,k)+uu(jr,2,k))*wgts(j)
 83    15      uo(2,j,k)=(uu(j,2,k)-uu(jr,2,k))*wgts(j)
 84    16      ve(2,j,k)=(vv(j,2,k)+vv(jr,2,k))*wgts(j)
 85    13      vo(2,j,k)=(vv(j,2,k)-vv(jr,2,k))*wgts(j)
 86             end do
 87             enddo
 89             ke=nsig-mod(nsig,2)
 90             me=lene-mod(lene,4)
 91             mo=leno-mod(leno,4)
 93             do k=1,ke,2
 94             do m=1,me,4
 95             dse00=0.
 96             dse10=0.
 97             dse20=0.
 98             dse30=0.
 99             dse01=0.
100             dse11=0.
101             dse21=0.
102             dse31=0.
```

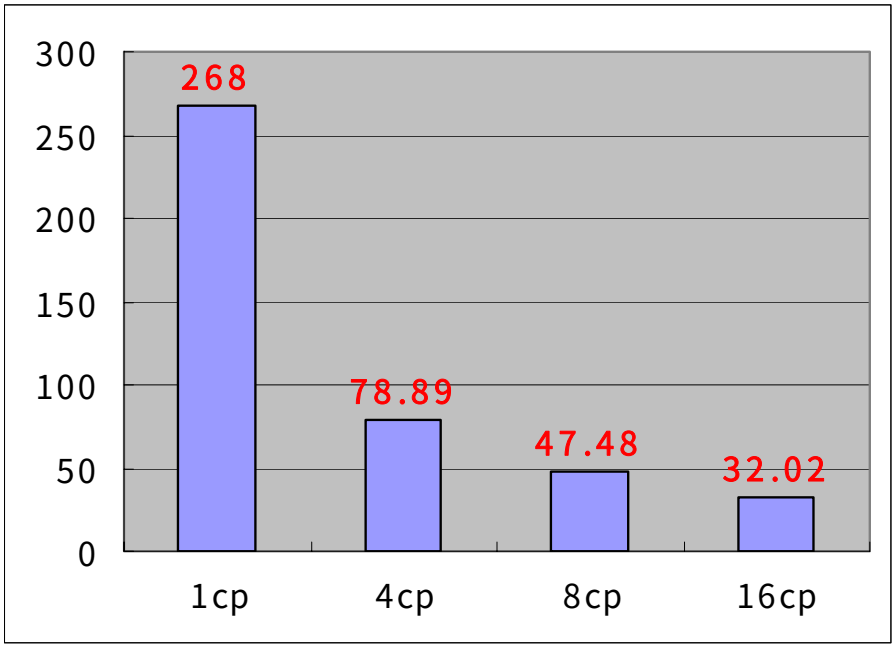
```
  103                do j=1,nlath
  104                dse00=dse00-qlnte2(j,m   )*ue(2,j,k)-rlnte1(j,m)*vo(1,j,k)
  105                dse10=dse10-qlnte2(j,m+1)*ue(2,j,k)-rlnte1(j,m+1)*vo(1,j,k)
  106                dse20=dse20-qlnte2(j,m+2)*ue(2,j,k)-rlnte1(j,m+2)*vo(1,j,k)
  107                dse30=dse30-qlnte2(j,m+3)*ue(2,j,k)-rlnte1(j,m+3)*vo(1,j,k)
  108    29          dse01=dse01-qlnte2(j,m   )*ue(2,j,k+1)-rlnte1(j,m)*vo(1,j,k+1)
  109           31                                dse11=dse11-qlnte2(j,m+1)*ue(2,j,k+1)-
rlnte1(j,m+1)*vo(1,j,k+1)
  110                                             dse21=dse21-qlnte2(j,m+2)*ue(2,j,k+1)-
rlnte1(j,m+2)*vo(1,j,k+1)
  111           14                                dse31=dse31-qlnte2(j,m+3)*ue(2,j,k+1)-
rlnte1(j,m+3)*vo(1,j,k+1)
  112              end do
  113     9        dse1(m   ,k)=(dse1(m   ,k)+dse00)*del2e1(m)
  114    21        dse1(m+1,k)=(dse1(m+1,k)+dse10)*del2e1(m+1)
  115    18        dse1(m+2,k)=(dse1(m+2,k)+dse20)*del2e1(m+2)
  116    38        dse1(m+3,k)=(dse1(m+3,k)+dse30)*del2e1(m+3)
  117     3        dse1(m   ,k+1)=(dse1(m   ,k+1)+dse01)*del2e1(m)
  118              dse1(m+1,k+1)=(dse1(m+1,k+1)+dse11)*del2e1(m+1)
  119     2        dse1(m+2,k+1)=(dse1(m+2,k+1)+dse21)*del2e1(m+2)
  120     2        dse1(m+3,k+1)=(dse1(m+3,k+1)+dse31)*del2e1(m+3)
  121              end do
  132              enddo
  159              do k=1,ke,2
  160              do m=1,me,4
  161              dse00=0.
  162              dse10=0.
  163              dse20=0.
  164              dse30=0.
  165              dse01=0.
  166              dse11=0.
  167              dse21=0.
  168              dse31=0.
  169              do j=1,nlath
  170    1         dse00=dse00+qlnte1(j,m   )*ue(1,j,k)-rlnte2(j,m   )*vo(2,j,k)
  171              dse10=dse10+qlnte1(j,m+1)*ue(1,j,k)-rlnte2(j,m+1)*vo(2,j,k)
  172              dse20=dse20+qlnte1(j,m+2)*ue(1,j,k)-rlnte2(j,m+2)*vo(2,j,k)
  173              dse30=dse30+qlnte1(j,m+3)*ue(1,j,k)-rlnte2(j,m+3)*vo(2,j,k)
  174    22        dse01=dse01+qlnte1(j,m   )*ue(1,j,k+1)-rlnte2(j,m   )*vo(2,j,k+1)
  175    31       dse11=dse11+qlnte1(j,m+1)*ue(1,j,k+1)-rlnte2(j,m+1)*vo(2,j,k+1)
  176     1       dse21=dse21+qlnte1(j,m+2)*ue(1,j,k+1)-rlnte2(j,m+2)*vo(2,j,k+1)
  177    11       dse31=dse31+qlnte1(j,m+3)*ue(1,j,k+1)-rlnte2(j,m+3)*vo(2,j,k+1)
  178              end do
  179     15       dse2(m   ,k)=(dse2(m   ,k)+dse00)*del2e2(m)
  180     22       dse2(m+1,k)=(dse2(m+1,k)+dse10)*del2e2(m+1)
```

```
181     16      dse2(m+2,k)=(dse2(m+2,k)+dse20)*del2e2(m+2)
182     22      dse2(m+3,k)=(dse2(m+3,k)+dse30)*del2e2(m+3)
183     4       dse2(m   ,k+1)=(dse2(m   ,k+1)+dse01)*del2e2(m)
184     4       dse2(m+1,k+1)=(dse2(m+1,k+1)+dse11)*del2e2(m+1)
185     2       dse2(m+2,k+1)=(dse2(m+2,k+1)+dse21)*del2e2(m+2)
186             dse2(m+3,k+1)=(dse2(m+3,k+1)+dse31)*del2e2(m+3)
187     1       end do
198     3       enddo
```

7. ssi.x summary

< 표 I.3 Original 코드와 최적화 코드의 Time Step당 계산시간 비교 및 Speep up >

		Parallel Speed up
Original (Serial)	2102 s	
Tuned 1cp	268	7.84
4 cp	78.89	3.39
8 cp	47.48	5.64
16 cp	32.02	8.36



< 그림 I.11 Original 코드와 최적화 및 병렬화 코드의 성능 비교 그래프 >

앞에서 살펴본 바와 같이 ssi.x 코드는 복잡한 자료구조와 계산을 수행하고 있다. 최적화/병렬화 코드에서는 전체적으로 메모리 접근의 연속성을 높이고, 불필요한 판단문의 사용을 줄여 코드의 흐름을 단순화 시켰다. 크

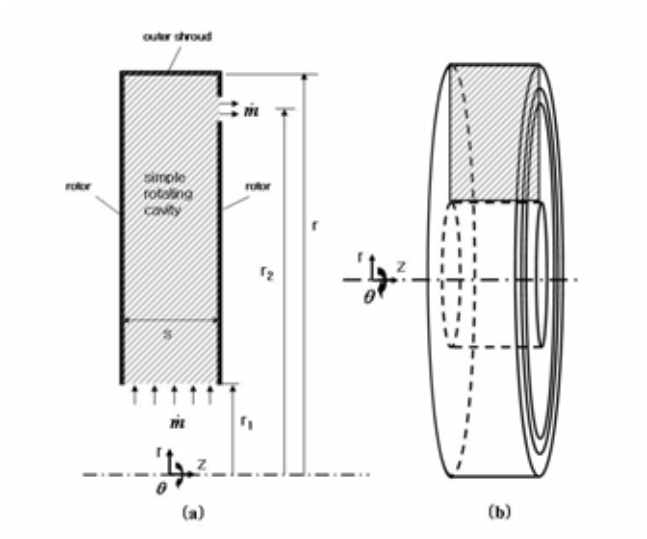
기가 큰 루프 앞에 parallel do를 삽입하는 루프 병렬화를 몇 몇 부분에서 시도하고 있지만, 전체적으로 수행되는 계산을 각 스레드에 명시적으로 할당하는 병렬화를 시도 하였다. 계산과정에서 주어진 배열의 데이터가 모두 사용되지 않고 역삼각형 모양으로 분포된 데이터만 사용되므로 각 스레드에 할당되는 병렬 작업량을 균등하게 해주기 위해 로드 밸런싱을 고려한 블록 분할이 시도 되었다.

병렬화 과정에서 코드가 전체적으로 수정 되었기에 순차 최적화 코드를 따로 테스트할 수가 없었고 따라서 위의 tuned 1cp 실행 결과는 병렬 프로그램을 1 cpu를 사용해 테스트한 결과이다. 일반적으로 순차 코드보다 1 cpu를 사용한 병렬 프로그램의 실행시간이 더 높게 나오는 것을 감안 하더라도 original 코드와 비교해 최적화된 코드의 성능이 상당부분 개선 되었음을 확인할 수 있다.

CHAPTER II. RCAV 코드

1. Code information

Numerical Study on the Flow between a pair of Co-Rotating Disks



< 그림 II.1 Schematic diagram >

반경방향의 입구와 z 방향의 출구를 가지는 실린더가 회전하는 물리 현상으로 이와 같은 회전디스크 시스템의 내부 유동은 가스 터빈 엔진이나 하드디스크 드라이브 등과 관련된 산업에서 오랫동안 연구되어 왔던 분야이다. 내부 유동이 물리적으로 매우 복잡하며, 계산 시간이 오래 걸리기 때문에 그 동안의 연구는 실험이나, 2차원 문제를 푸는 것에 그쳤으며 3차원 문제라 하더라도 비교적 작은 Re(레이놀즈수)에서 이루어 졌었는데, 이 연구에서는 3차원이며 레이놀즈 넘버가 10,000인 경우에 해당되는 계산을 MPI로 병렬화된 프로그램을 통해 수행하게 된다.

2. Makefile

< Original 코드의 Makefile >

```
.SUFFIXES: .f90 .f .o

FFLAGS   = -q64 -O3 -qarch=pwr4 -qtune=pwr4 -qcache=auto -qsuffix=f=f90 -pg -g
LDFLAGS = -q64 -pg -L/applic/mpitrace/lib -lmpitrace
F90       = mpxlf90
LD         = mpxlf90_r
OUT       = main.x

SRC = main.f90 bcp.f90 bcu.f90 bcv.f90 bcw.f90 calc_p.f90 calc_u.f90 ₩
        calc_v.f90 calc_w.f90 coeff.f90 grid.f90 init.f90 metric.f90 ₩
        plot.f90 tdma.f90 define_array.f90 exchange.f90

OBJ = main.o bcp.o bcu.o bcv.o bcw.o calc_p.o calc_u.o ₩
        calc_v.o calc_w.o coeff.o grid.o init.o metric.o ₩
        plot.o tdma.o define_array.o exchange.o

CPF = comdat_param.f90
CSF = comdat_shared.f90

CPO = comdat_param.o
CSO = comdat_shared.o

$(OUT) : $(OBJ) $(CPO) $(CSO)
        $(LD) $(LDFLAGS) -o $(OUT) $(OBJ) $(CPO) $(CSO)

$(OBJ) : $(CPO) $(CSO)

$(CPO) : $(CPF)
        $(F90) $(FFLAGS) -c $(CPF)

$(CSO) : $(CPO) $(CSF)
        $(F90) $(FFLAGS) -c $(CSF)

.f90.o:
        $(F90) $(FFLAGS) -c $<

clean:
        rm -f *.o $(OUT)
        rm -f *.mod
```

< 최적화 코드의 Makefile >

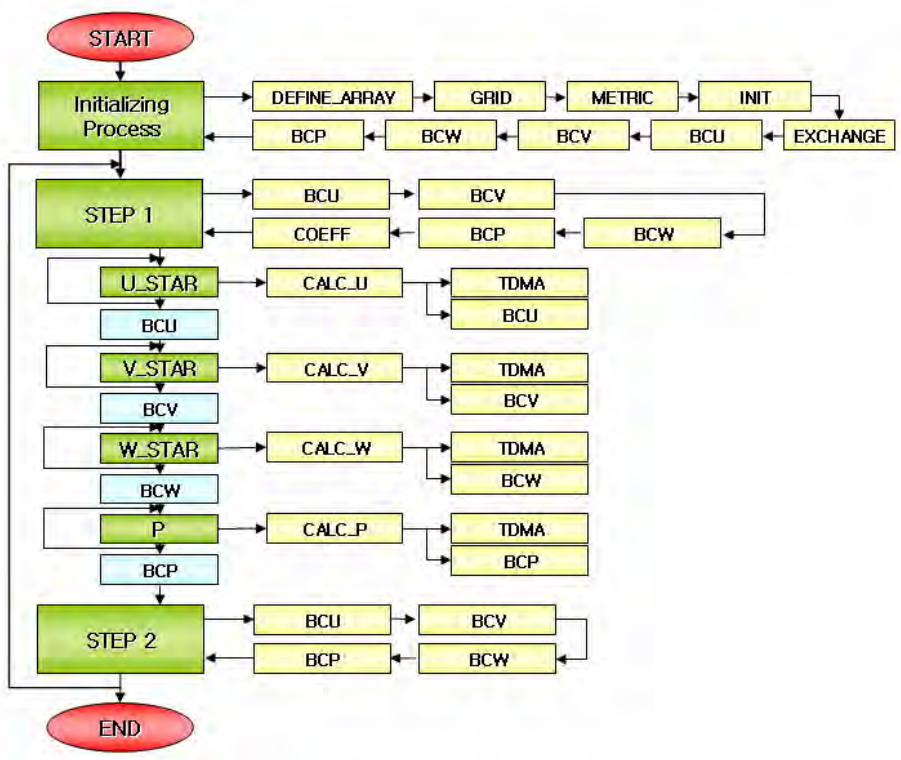
```
.SUFFIXES: .f90 .f .o

FFLAGS     = -q64 -O3 -qarch=pwr4 -qtune=pwr4 -qcache=auto -qsuffix=f=f90 -pg -g
LDFFLAGS = -q64 -pg -L/applic/mpitrace/lib -lmpitrace
F90          = mpxlf90
LD           = mpxlf90_r
OUT          = main.x
SRC = main.f90 bcp.f90 bcu.f90 bcv.f90 bcw.f90 calc_p.f90 calc_u.f90 ₩
         calc_v.f90 calc_w.f90 coeff.f90 grid.f90 init.f90 metric.f90 ₩
         plot.f90 tdma.f90 define_array.f90 exchange.f90
OBJ = main.o bcp.o bcu.o bcv.o bcw.o calc_p.o calc_u.o ₩
         calc_v.o calc_w.o coeff.o grid.o init.o metric.o ₩
         plot.o tdma.o define_array.o exchange.o
CPF = comdat_param.f90
CSF = comdat_shared.f90
CPO = comdat_param.o
CSO = comdat_shared.o
$(OUT) : $(SRC) $(CPF) $(CSF) $(OBJ) $(CPO) $(CSO)
         $(LD) $(LDFFLAGS) -o $(OUT) $(OBJ) $(CPO) $(CSO)
$(OBJ) : $(CPO) $(CSO)

$(CPO) : $(CPF)
         $(F90) $(FFLAGS) -c $(CPF)
$(CSO) : $(CPO) $(CSF)
         $(F90) $(FFLAGS) -c $(CSF)
.f90.o:
         $(F90) -c $(FFLAGS) -c $<
clean:
         rm -f *.o $(OUT)
         rm -f *.mod
```

MPI로 병렬화되어 있는 Original 코드와 최적화된 코드의 Makefile에서 동일한 컴파일 및 링크 옵션을 사용하고 있으며, 프로파일링을 위해 -pg -g 및 -pg 옵션이 컴파일 및 링크할 때 각각 추가되어 있다.

또한 통신량을 확인하기 위해 mpitrace 라이브러리를 링크할 때 추가하였다.

3. Flow Chart

아래의 Flow Chart 그림과 같이 STEP1과 STEP2를 지나면서 time Iteration을 하고, STEP1 및 STEP2 사이에서 U_STAR, V_STAR, W_STAR, P 부분을 계산하면서도 각각의 계산 과정에서 Iteration이 진행된다. BCU, BCV, BCW, BCP 루틴은 Initializing Process, STEP1, STEP2에서 동일하고 call되어 사용된다.



< 그림 II.2 전체 순서도 >

4. Profiling 결과

<Original 코드의 profiling 결과>

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
33.8	11577.59	11577.59	5670	2041.90	2291.26	.calc_p [3]
17.2	17461.91	5884.32	1010	5826.06	5826.06	.coeff [4]
7.7	20090.71	2628.80	11790	222.97	222.98	.tdma [8]
7.6	22701.89	2611.18	10	261118.00	2985698.05	.main [1]
7.4	25223.37	2521.48				._lapi_shm_dispatcher [9]
6.6	27502.47	2279.10	2080	1095.72	1343.76	.calc_w [5]
6.5	29727.42	2224.95	2020	1101.46	1353.19	.calc_v [6]
6.3	31893.49	2166.07	2020	1072.31	1325.55	.calc_u [7]
2.2	32660.94	767.45				._lapi_dispatcher [10]
1.4	33156.69	495.75				.LAPI__Msgpoll [11]
0.9	33450.30	293.61				._is_yield_queue_empty [12]
0.5	33626.30	176.00	20050	8.78	8.79	.bcp [13]
0.2	33707.17	80.87	9100	8.89	10.09	.bcu [14]
0.2	33784.76	77.59	9100	8.53	9.58	.bcv [15]
0.2	33862.12	77.36	9280	8.34	8.35	.bcw [16]
0.2	33923.19	61.07				._tag_waiting [17]
0.1	33953.11	29.92	10	2992.00	2992.13	.metric [18]
0.1	33982.14	29.03				._dgsm_gather [19]
0.1	34009.08	26.94				._cntr_waiting [21]
0.1	34034.95	25.87				.__mcount [22]
0.1	34053.59	18.64				.global_unlock_ppc_mp [25]
0.0	34065.96	12.37	42507744	0.00	0.00	.cvtloop [27]
0.0	34077.66	11.70	165526200	0.00	0.00	._sin [28]
0.0	34087.38	9.72				.FormatControl [30]
0.0	34096.58	9.20	165526200	0.00	0.00	._cos [31]
0.0	34105.43	8.85				.global_lock_ppc_mp [32]
0.0	34113.95	8.52				.__mcount [33]
0.0	34121.78	7.83				.__mcount [34]
0.0	34128.75	6.97	42740078	0.00	0.00	.__cvt_r [24]
0.0	34134.86	6.11	146884858	0.00	0.00	.pthread_mutex_lock [35]
0.0	34140.88	6.02				.icopy [36]
0.0	34146.74	5.86				.FmtRToQED [20]
0.0	34152.40	5.66				._dgsm_scatter [37]
0.0	34158.01	5.61				.fetch_and_add [38]
0.0	34163.43	5.42	41184000	0.00	0.00	._pow [29]
0.0	34168.82	5.39	10	539.00	539.56	.init [39]
0.0	34173.31	4.49				.MPI__Sendrecv [40]
0.0	34177.40	4.09	10	409.00	1569.25	.grid [26]
0.0	34180.87	3.47				._moveeq [42]
0.0	34184.31	3.44	41184000	0.00	0.00	.expinner2 [43]

0.0	34187.54	3.23				.LogEvent [45]
0.0	34190.60	3.06				.lw_mutex_trylock [46]
0.0	34193.44	2.84				._try_to_free [47]
0.0	34196.13	2.69	41184000	0.00	0.00	.loginner2 [49]

<최적화 코드의 profiling 결과>

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
25.7	1002.04	1002.04	5520	181.53	261.23	.calc_p [3]
16.5	1645.23	643.19	11530	55.78	56.64	.tdma [4]
10.8	2066.64	421.41	1010	417.24	417.24	.coeff [7]
9.4	2434.10	367.46	10	36746.00	366805.88	.main [1]
7.0	2708.39	274.29	2030	135.12	214.94	.calc_w [5]
7.0	2980.64	272.25	1990	136.81	215.75	.calc_v [6]
6.7	3240.59	259.95	1990	130.63	210.70	.calc_u [8]
3.9	3390.80	150.21	14080	10.67	11.53	.bcp [9]
2.6	3491.32	100.52				._lapi_shm_dispatcher [10]
2.0	3567.51	76.19	7020	10.85	11.71	.bcu [12]
2.0	3643.69	76.18	7100	10.73	11.59	.bcw [11]
1.9	3715.90	72.21	7020	10.29	11.15	.bcv [13]
1.0	3755.92	40.02	46920	0.85	0.85	.exchange [14]
0.8	3786.66	30.74				._lapi_dispatcher [15]
0.5	3806.02	19.36				.LAPI__Msgpoll [16]
0.4	3821.53	15.51				.icopy [17]
0.3	3833.93	12.40				._is_yield_queue_empty [18]
0.2	3842.92	8.99				._tag_waiting [19]
0.2	3849.40	6.48				.__mcount [20]
0.1	3855.19	5.79	10	579.00	586.74	.metric [21]
0.1	3859.50	4.31	10	431.00	431.00	.grid [22]
0.1	3862.15	2.65				.global_unlock_ppc_mp [23]
0.1	3864.55	2.40				._moveeq [24]
0.1	3866.85	2.30				._cntr_waiting [25]
0.1	3868.91	2.06	10	206.00	206.00	.init [26]
0.0	3870.43	1.52				.WriteUnit [27]
0.0	3871.86	1.43				.global_lock_ppc_mp [28]
0.0	3872.90	1.04				.__mcount [30]
0.0	3873.83	0.93				.IOWrite [32]
0.0	3874.68	0.85				._xlfWriteUfmt [33]
0.0	3875.45	0.77	21123963	0.00	0.00	.pthread_mutex_lock [34]
0.0	3876.22	0.77				.fetch_and_add [35]
0.0	3876.93	0.71				._xlfBeginIO [29]
0.0	3877.55	0.62				._xlfEndIO [36]
0.0	3877.99	0.44				.MPI__Sendrecv [41]
0.0	3878.37	0.38				._barrier_onnode [42]
0.0	3878.75	0.38				._receive_shm_contig_message [43]

0.0	3879.12	0.37			._send_shm_processing [45]
0.0	3879.47	0.35			.PrepareUnit [46]
0.0	3879.82	0.35			.__mcount [47]
0.0	3880.16	0.34			.mpci_recv [48]
0.0	3880.49	0.33	21123993	0.00	0.00 .pthread_mutex_unlock [49]
0.0	3880.82	0.33			.mpci_send [44]
0.0	3881.14	0.32	19030634	0.00	0.00 ._lib_mutex_lock [31]
0.0	3881.44	0.30			._ptrgl [50]
0.0	3881.74	0.30			._xlfWriteUfmt.GL [51]
0.0	3882.03	0.29			.LogEvent [52]

MPI task를 10개로 설정한 후 수행한 결과이다. Origincal 코드와 최적화 코드에서 각각 대응되는 서브루틴들을 비교해보면 수행시간이 많이 줄어든 것을 확인해 볼 수 있다.

다음 그림은 xprofiler 결과를 보여준다.

<그림 II.3 Original 코드의 xprofiler 결과>

<그림 II.4 최적화 코드의 xprofiler 결과>

5. hpmcount 결과

컴파일 된 실행파일을 hpmcount와 같이 수행하면 다음과 같은 코드 수행에 관한 정보를 볼 수 있다. Original 코드 및 최적화 코드 모두 10개의 CPU를 이용하여 수행하였으며 hpmcount 결과는 0번 task에 해당되는 내용이다.

<Original 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 3642.139102 seconds

########  Resource Usage Statistics  ########

Total amount of time in user mode            : 3630.430000 seconds
Total amount of time in system mode          : 2.830000 seconds
Maximum resident set size                    : 480952 Kbytes
Average shared memory use in text segment    : 1059191 Kbytes*sec
Average unshared memory use in data segment  : 1727733830 Kbytes*sec
Number of page faults without I/O activity   : 121564
Number of page faults with I/O activity      : 232
Number of times process was swapped out      : 0
Number of times file system performed INPUT  : 0
Number of times file system performed OUTPUT : 0
Number of IPC messages sent                  : 0
Number of IPC messages received              : 0
Number of signals delivered                  : 0
Number of voluntary context switches         : 279117
Number of involuntary context switches       : 3834

#######  End of Resource Statistics  ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                    :       1750286335
 PM_FPU_FMA (FPU executed multiply-add instruction)             :      19653315893
 PM_FPU0_FIN (FPU0 produced a result)                           :      65347503817
 PM_FPU1_FIN (FPU1 produced a result)                           :      66315674590
 PM_CYC (Processor cycles)                                      :    6168264351451
 PM_FPU_STF (FPU executed store instruction)                    :      75676214802
 PM_INST_CMPL (Instructions completed)                          :    1244908223846
 PM_LSU_LDF (LSU executed Floating Point load instruction)      :     114778697412

 Utilization rate                                 :              99.389 %
```

```
Load and store operations                    :        190454.912 M
MIPS                                         :            341.807
Instructions per cycle                       :              0.202
HW Float points instructions per Cycle       :              0.021
Floating point instructions + FMAs           :        75640.279 M
Float point instructions + FMA rate          :           20.768 Mflip/s
FMA percentage                               :            51.965 %
Computation intensity                        :              0.397
```

<최적화 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 397.068083 seconds

########  Resource Usage Statistics  ########

Total amount of time in user mode            : 387.310000 seconds
Total amount of time in system mode          : 3.040000 seconds
Maximum resident set size                    : 397772 Kbytes
Average shared memory use in text segment    : 115117 Kbytes*sec
Average unshared memory use in data segment  : 152869829 Kbytes*sec
Number of page faults without I/O activity   : 100239
Number of page faults with I/O activity      : 199
Number of times process was swapped out      : 0
Number of times file system performed INPUT  : 0
Number of times file system performed OUTPUT : 0
Number of IPC messages sent                  : 0
Number of IPC messages received              : 0
Number of signals delivered                  : 0
Number of voluntary context switches         : 5788
Number of involuntary context switches       : 6786

#######  End of Resource Statistics  ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                    :       1230948556
 PM_FPU_FMA (FPU executed multiply-add instruction)             :      17293272758
 PM_FPU0_FIN (FPU0 produced a result)                           :      26187141094
 PM_FPU1_FIN (FPU1 produced a result)                           :      30319611673
 PM_CYC (Processor cycles)                                      :     657389581005
 PM_FPU_STF (FPU executed store instruction)                    :      12471930473
 PM_INST_CMPL (Instructions completed)                          :     208858400436
 PM_LSU_LDF (LSU executed Floating Point load instruction)      :      51376409716

 Utilization rate                             :            97.160 %
 Load and store operations                    :        63848.340 M
 MIPS                                         :           526.001
```

Instructions per cycle	:	0.318
HW Float points instructions per Cycle	:	0.086
Floating point instructions + FMAs	:	61328.095 M
Float point instructions + FMA rate	:	154.452 Mflip/s
FMA percentage	:	56.396 %
Computation intensity	:	0.961

Original 코드와 Serial 최적화 코드의 hpmcount 결과를 비교해 보면 먼저 original 코드의 경우 부동소수점 연산횟수가 75640M정도 되고 부동소수점 연산 속도가 20.8MFLOPS 정도 되며, 최적화 코드의 경우 부동소수점 연산횟수가 61328M 정도 되고 부동소수점 연산 속도가 154.5MFLOPS 정도 된다. 최적화 결과 연산회수가 약 20% 정도 줄어들었고 연산속도는 거의 7~8배 가까이 빨라졌음을 알 수 있다.

6. 최적화 코드 비교 및 분석

다음 subroutine들에 대해서 최적화를 진행하였다.

< 표 II.1 서브루틴 별 최적화 tick수 비교 및 speed-up >

	Subroutine 명	Original 코드의 tick 수	최적화 코드의 tick수	Speed-up
1	main	261118	36746	7.11
2	define_array	0	0	1.0
3	grid	409	431	0.95
4	metric	2992	579	5.17
5	exchange	46	4002	0.01
6	coeff	588432	42141	13.96
7	calc_u	216607	25995	8.33
8	calc_v	222495	27225	8.17
9	calc_w	227910	27429	8.31
10	calc_p	1157759	100204	11.55
11	tdma	262880	64319	4.09
total	-	2940648	329071	8.94

여기서 tick 수는 해당 subroutine의 대략적인 수행시간을 나타내는 것으로 1 tick당 0.01초를 나타내며 speed-up은 해당 코드에 대해 최적화 작업으로 빨라진 비율을 나타낸다. 전체적으로 Main Calculation Process에서 각 subroutine들을 최적화하여 약 8.94배의 성능향상이 되었음을 알 수 있다. 각 subroutine에 대해서 하나씩 분석해 보면 다음과 같다.

6.1 main 루틴

A. Cartesian Toplogy 를 포함하는 Communicator 생성

<Original 코드>

```
 42                    CALL MPI_INIT(ierr)
 43                    CALL MPI_COMM_RANK(MPI_COMM_WORLD,myPE,ierr)
 44                    CALL MPI_COMM_SIZE(MPI_COMM_WORLD,NtotalPE,ierr)
 45
 46                    PERIODIC(1) = .FALSE.     !--- up & down
 47                    PERIODIC(2) = .TRUE.      !--- left & right
 48                    REORDER = .TRUE.
 49
 50                    CALL MPI_CART_CREATE(MPI_COMM_WORLD,
                      >NDIMS,DIMS, PERIODIC,REORDER,COMM2D,ierr)
 51                    CALL MPI_COMM_RANK(COMM2D,myPE,ierr)
 52                    CALL MPI_CART_COORDS(COMM2D,myPE,
                      >NDIMS,COORDS,ierr)
 68                    CALL MPI_CART_SHIFT(COMM2D,1,1,PEw,PEe,ierr)
 69                    CALL MPI_CART_SHIFT(COMM2D,0,1,PEs,PEn,ierr)
 71                    CALL DEFINE_ARRAY
 73                    count=EY1-SY1+3
 74                    block=1
 75                    stride=EX1-SX1+3
 76
 77                    call MPI_Type_vector(count,block,stride,
                      >MPI_DOUBLE_PRECISION,column0,ierr)
 78                    call MPI_Type_commit(column0,ierr)
 79
 80                    call MPI_Type_contiguous(EX1-SX1+3,
                      >MPI_DOUBLE_PRECISION,row0,ierr)
 81                    call MPI_Type_commit(row0,ierr)
```

① COMM2D라는 새로운 Communicator를 생성하고 COMM2D에서 새로운 Rank(myPE)를 지정하였다.

B. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
148                       IF (NITER.NE.1) THEN
149                       DO I=SX,EX
150                       DO J=SY,EY
```

```
151          12              DO K=SZ,EZ
152        3496              UTEMP(I,J,K)=U(I,J,K)
153         515              ENDDO
154           4              ENDDO
155                          ENDDO
156                          ENDIF

162                          DO I=SX,EX
163           2              DO J=SY,EY
164                          DO K=SZ,EZ
165           1              IF (U(I,J,K).eq.0.0D0) THEN
166                          RESORU = RESORU
167                          ELSE
168       14799              RESORU = RESORU+ABS((UTEMP(I,J,K)-U(I,J,K))
                          >                /U(I,J,K) )
169                          ENDIF
170                          ENDDO
171           6              ENDDO
172                          ENDDO

191                          DO I=SX,EX
192                          DO J=SY,EY
193                          DO K=SZ,EZ
194        3555              U_S(I,J,K)=U(I,J,K)
195         557              ENDDO
196          27              ENDDO
197                          ENDDO
```

```
206                          IF (NITER.NE.1) THEN
207                          DO I=SX,EX
208                          DO J=SY,EY
209          12              DO K=SZ,EZ
210        3400              VTEMP(I,J,K)=V(I,J,K)
211         564              ENDDO
212           3              ENDDO
213                          ENDDO
214                          ENDIF

220                          DO I=SX,EX
221           5              DO J=SY,EY
222                          DO K=SZ,EZ
223           1              IF (V(I,J,K).eq.0.0D0) THEN
224                          RESORU = RESORU
225                          ELSE
226       14501              RESORV=RESORV+ABS((VTEMP(I,J,K)-V(I,J,K))
                          >                /V(I,J,K))
227                          ENDIF
```

```
228                     ENDDO
229           1         ENDDO
230                     ENDDO

249                     DO I=SX,EX
250                     DO J=SY,EY
251           1         DO K=SZ,EZ
252        3468         V_S(I,J,K)=V(I,J,K)
253         500         ENDDO
254          35         ENDDO
255                     ENDDO
```

```
264                     IF (NITER.NE.1) THEN
265                     DO I=SX,EX
266                     DO J=SY,EY
267           7         DO K=SZ,EZ
268        3635         WTEMP(I,J,K)=W(I,J,K)
269         539         ENDDO
270           2         ENDDO
271                     ENDDO
272                     ENDIF

278                     DO I=SX,EX
279           1         DO J=SY,EY
280                     DO K=SZ,EZ
281          31         IF (W(I,J,K).eq.0.0D0) THEN
282                     RESORU = RESORU
283                     ELSE
284       14358         RESORV=RESORV+ABS((WTEMP(I,J,K)-W(I,J,K))
                      >             /W(I,J,K))
285                     ENDIF
286                     ENDDO
287          10         ENDDO
288                     ENDDO

307                     DO I=SX,EX
308                     DO J=SY,EY
309                     DO K=SZ,EZ
310        3397         W_S(I,J,K)=W(I,J,K)
311         593         ENDDO
312          34         ENDDO
313                     ENDDO
```

```
322                     IF (NITER.NE.1) THEN
```

```
323                       DO I=SX,EX
324                       DO J=SY,EY
325           69          DO K=SZ,EZ
326        15911          PTEMP(I,J,K)=P(I,J,K)
327         2304          ENDDO
328            9          ENDDO
329                       ENDDO
330                       ENDIF

336            1          DO I=SX,EX
337           34          DO J=SY,EY
338                       DO K=SZ,EZ
339        11868          IF (P(I,J,K).eq.0.0D0) THEN
340                       RESORU = RESORU
341                       ELSE
342        24383          RESORP=RESORP+ABS((PTEMP(I,J,K)-P(I,J,K))
                        >                 /P(I,J,K))
343                       ENDIF
344                       ENDDO
345           27          ENDDO
346                       ENDDO
```

```
374                       DO I=SX1,EX1
375            9          DO J=SY1,EY1
376           41          DO K=SZ1,EZ1
377
378        31906                 UC(I,J,K)=-DEL_T/2.0D0   &
379                       *(XIX(I,J,K)*(P(I+1,J,K)-P(I-1,J,K))       &
380                         +ETX(I,J,K)*(P(I,J+1,K)-P(I,J-1,K))      &
381                         +ZTX(I,J,K)*(P(I,J,K+1)-P(I,J,K-1)))
382        23491            VC(I,J,K)=-DEL_T/2.0D0                   &
383                         *(XIY(I,J,K)*(P(I+1,J,K)-P(I-1,J,K))     &
384                         +ETY(I,J,K)*(P(I,J+1,K)-P(I,J-1,K))      &
385                         +ZTY(I,J,K)*(P(I,J,K+1)-P(I,J,K-1)))
386        23496           WC(I,J,K)=-DEL_T/2.0D0                    &
387                         *(XIZ(I,J,K)*(P(I+1,J,K)-P(I-1,J,K))     &
388                         +ETZ(I,J,K)*(P(I,J+1,K)-P(I,J-1,K))      &
389                         +ZTZ(I,J,K)*(P(I,J,K+1)-P(I,J,K-1)))
390
391         4216                  U(I,J,K)=U_S(I,J,K)+UC(I,J,K)
392        11282                  V(I,J,K)=V_S(I,J,K)+VC(I,J,K)
393        14394                  W(I,J,K)=W_S(I,J,K)+WC(I,J,K)
394                       ENDDO
395           52          ENDDO
396                       ENDDO

407                       DO I=sx,ex
```

```
408                       DO J=sy,ey
409            4          DO K=sz,ez
410         6902          V_RAD(I,J,K) = -1.0D0*U(I,J,K)*COS(THT(I))
                        >                + V(I,J,K)*SIN(THT(I))
411         4204          V_THT(I,J,K) =          U(I,J,K)*SIN(THT(I))
                        >                + V(I,J,K)*COS(THT(I))
412                       ENDDO
413           35          ENDDO
414                       ENDDO

416                       DO I=sx1,ex1
417           14          DO J=sy1,ey1
418            1          DO K=sz1,ez1
419
420           23          DEL_RAD = (    RAD(J+1) -    RAD(J-1) )/2.0D0
421                       DEL_THT = (    THT(I+1) -    THT(I-1) )/2.0D0
422         7414          DEL_Z    = ( Z(I,J,K+1) - Z(I,J,K-1) )/2.0D0
423
424         8365          CFL_NUM = DABS( V_RAD(I,J,K)*DEL_T
                        >             / DEL_RAD ) +   &
425                                    DABS( V_THT(I,J,K)*DEL_T
                        >             / DEL_THT ) +   &
426                                    DABS(W(I,J,K)*DEL_T / DEL_Z )
427
428                       IF (CFL_NUM .GT. CFL_MAX) THEN
429                             CFL_MAX = CFL_NUM
430                       ENDIF
431
432                       enddo
433            6          enddo
434                       enddo
```

<최적화 코드>

```
152                       IF (NITER.NE.1) THEN
153                       DO K=SZ,EZ
154                       DO J=SY,EY
155            7          DO I=SX,EX
156          470          UTEMP(I,J,K)=U(I,J,K)
157           16          ENDDO
158                       ENDDO
159                       ENDDO
160                       ENDIF

166            1          DO K=SZ,EZ
167           11          DO J=SY,EY
168                       DO I=SX,EX
```

```
169          9          IF (U(I,J,K).eq.0.0D0) THEN
170                     RESORU = RESORU
171                     ELSE
172       2002          RESORU = RESORU+ABS( ( UTEMP(I,J,K)-U(I,J,K) )
                       >             / U(I,J,K) )
173                     ENDIF
174                     ENDDO
175                     ENDDO
176                     ENDDO

195                     DO K=SZ,EZ
196          1          DO J=SY,EY
197          8          DO I=SX,EX
198        545          U_S(I,J,K)=U(I,J,K)
199         23          ENDDO
200          3          ENDDO
201                     ENDDO
```

```
210                     IF (NITER.NE.1) THEN
211                     DO K=SZ,EZ
212                     DO J=SY,EY
213          8          DO I=SX,EX
214        492          VTEMP(I,J,K)=V(I,J,K)
215         20          ENDDO
216          4          ENDDO
217                     ENDDO
218                     ENDIF

224                     DO K=SZ,EZ
225         16          DO J=SY,EY
226                     DO I=SX,EX
227                     IF (V(I,J,K).eq.0.0D0) THEN
228                     RESORU = RESORU
229                     ELSE
230       2032          RESORV=RESORV+ABS((VTEMP(I,J,K)
                     >             -V(I,J,K))/V(I,J,K))
231                     ENDIF
232                     ENDDO
233                     ENDDO
234                     ENDDO

253                     DO K=SZ,EZ
254                     DO J=SY,EY
255          3          DO I=SX,EX
256        579          V_S(I,J,K)=V(I,J,K)
257         34          ENDDO
258                     ENDDO
```

```
259                     ENDDO
```

```
268                     IF (NITER.NE.1) THEN
269                     DO K=SZ,EZ
270                     DO J=SY,EY
271           3         DO I=SX,EX
272         512         WTEMP(I,J,K)=W(I,J,K)
273          16         ENDDO
274                     ENDDO
275                     ENDDO
276                     ENDIF

282           1         DO K=SZ,EZ
283          16         DO J=SY,EY
284                     DO I=SX,EX
285          24         IF (W(I,J,K).eq.0.0D0) THEN
286                     RESORU = RESORU
287                     ELSE
288        2050         RESORV=RESORV+ABS((WTEMP(I,J,K)-W(I,J,K))
                       >       /W(I,J,K))
289                     ENDIF
290                     ENDDO
291                     ENDDO
292                     ENDDO

311                     DO K=SZ,EZ
312           2         DO J=SY,EY
313           5         DO I=SX,EX
314         579         W_S(I,J,K)=W(I,J,K)
315          34         ENDDO
316           4         ENDDO
317                     ENDDO
```

```
326                     IF (NITER.NE.1) THEN
327                     DO K=SZ,EZ
328           2         DO J=SY,EY
329          21         DO I=SX,EX
330        2291         PTEMP(I,J,K)=P(I,J,K)
331         103         ENDDO
332          22         ENDDO
333                     ENDDO
334                     ENDIF

340           1         DO K=SZ,EZ
```

```
341          64          DO J=SY,EY
342                      DO I=SX,EX
343        4316           IF (P(I,J,K).eq.0.0D0) THEN
344                      RESORU = RESORU
345                      ELSE
346        1485           RESORP=RESORP+ABS((PTEMP(I,J,K)-P(I,J,K))
                         >            /P(I,J,K))
347                      ENDIF
348                      ENDDO
349                      ENDDO
350                      ENDDO
```

```
378                      DO K=SZ1,EZ1
379          21          DO J=SY1,EY1
380          19          DO I=SX1,EX1
381        2623                UC(I,J,K)=-DEL_T/2.0D0   &
382                             *(XIX(I,J,K)*(P(I+1,J,K)-P(I-1,J,K))       &
383                               +ETX(I,J,K)*(P(I,J+1,K)-P(I,J-1,K))       &
384                               +ZTX(I,J,K)*(P(I,J,K+1)-P(I,J,K-1)))
385        1938                VC(I,J,K)=-DEL_T/2.0D0                    &
386                             *(XIY(I,J,K)*(P(I+1,J,K)-P(I-1,J,K))       &
387                               +ETY(I,J,K)*(P(I,J+1,K)-P(I,J-1,K))       &
388                               +ZTY(I,J,K)*(P(I,J,K+1)-P(I,J,K-1)))
389        4283                WC(I,J,K)=-DEL_T/2.0D0                    &
390                             *(XIZ(I,J,K)*(P(I+1,J,K)-P(I-1,J,K))       &
391                               +ETZ(I,J,K)*(P(I,J+1,K)-P(I,J-1,K))       &
392                               +ZTZ(I,J,K)*(P(I,J,K+1)-P(I,J,K-1)))
393
394         221                U(I,J,K)=U_S(I,J,K)+UC(I,J,K)
395        1889                V(I,J,K)=V_S(I,J,K)+VC(I,J,K)
396         566                W(I,J,K)=W_S(I,J,K)+WC(I,J,K)
397                      ENDDO
398           8          ENDDO
399                      ENDDO

410                      DO K=sz,ez
411           4          DO J=sy,ey
412           3          DO I=sx,ex
413         548          V_RAD(I,J,K) = -1.0D0*U(I,J,K)*COSTHT(I)
                       >                + V(I,J,K)*SINTHT(I)
414         835          V_THT(I,J,K) =          U(I,J,K)*SINTHT(I)
                       >                + V(I,J,K)*COSTHT(I)
415                      ENDDO
416          24          ENDDO
417                      ENDDO
```

```
419                         DO K=sz1,ez1
420           1             DO J=sy1,ey1
421                         DO I=sx1,ex1
422
423           1             DEL_RAD = (     RAD(J+1) -     RAD(J-1) )/2.0D0
424         833             DEL_THT = (     THT(I+1) -     THT(I-1) )/2.0D0
425         489             DEL_Z     = ( Z(I,J,K+1) - Z(I,J,K-1) )/2.0D0
426
427        1388             CFL_NUM = DABS( V_RAD(I,J,K)*DEL_T
                                          / DEL_RAD ) +   &
428                                      DABS( V_THT(I,J,K)*DEL_T
                                            / DEL_THT ) +   &
429                                 DABS(       W(I,J,K)*DEL_T / DEL_Z )
430
431           1             IF (CFL_NUM .GT. CFL_MAX) THEN
432                               CFL_MAX = CFL_NUM
433                         ENDIF
434
435                         enddo
436           7             enddo
437                         enddo
```

① Original 코드에서 비연속적인 메모리 access를 피하기 위해서 inner loop와 outer loop의 수행순서를 바꾸어 연속적인 메모리 access를 하도록 수정하였다.

6.2 define_array 루틴

A. 영역 분할 및 array allocation

<Original 코드>

```
 18                         Temp1 = ( Nx+(2*dims(2)-2) ) / dims(2)
 19                         Temp2 = ( Ny+(2*dims(1)-2) ) / dims(1)
 20
 21                         SX1 = ( coords(2) * (Temp1-2) ) + 2
 22                         EX1 = SX1 + Temp1 - 3
 23
 24                         SY1 = ( coords(1) * (Temp2-2) ) + 2
 25                         EY1 = SY1 + Temp2 - 3
```

```
26
27                                      SZ1 = 2
28                                      EZ1 = Nz-1
29
30                                  IF ( coords(2).eq.(dims(2)-1) ) EX1 = Nx-1
31                                  IF ( coords(1).eq.(dims(1)-1) ) EY1 = Ny-1
32
33                                      SX = SX1-1
34                                      EX = EX1+1
35
36                                      SY = SY1-1
37                                      EY = EY1+1
38
39                                      SZ = SZ1-1
40                                      EZ = EZ1+1

43                                  allocate(U(SX:EX,SY:EY,SZ:EZ),stat=ALLOC_ERR)
44                                  allocate(V(SX:EX,SY:EY,SZ:EZ),stat=ALLOC_ERR)
45                                  allocate(W(SX:EX,SY:EY,SZ:EZ),stat=ALLOC_ERR)
46                                  allocate(P(SX:EX,SY:EY,SZ:EZ),stat=ALLOC_ERR)
```

<최적화 코드>

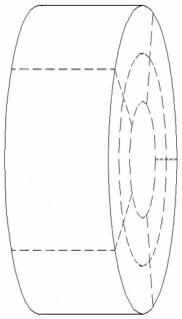
```
18                                  Temp1 = ( Nx+(2*dims(2)-2) ) / dims(2)
20                                  Temp2 = ( Nz+(2*dims(1)-2) ) / dims(1)
21
22                                  SX1 = ( coords(2) * (Temp1-2) ) + 2
23                                  EX1 = SX1 + Temp1 - 3
24
27                                  SZ1 = ( coords(1) * (Temp2-2) ) + 2
28                                  EZ1 = SZ1 + Temp2 - 3
29
32                                      SY1 = 2
33                                      EY1 = Ny-1
34
35                                  IF ( coords(2).eq.(dims(2)-1) ) EX1 = Nx-1
37                                  IF ( coords(1).eq.(dims(1)-1) ) EZ1 = Nz-1
38
39                                      SX = SX1-1
40                                      EX = EX1+1
42                                      SY = SY1-1
43                                      EY = EY1+1
44
45                                      SZ = SZ1-1
46                                      EZ = EZ1+1

49                                  allocate(U(SX:EX,SY:EY,SZ:EZ),stat=ALLOC_ERR)
```

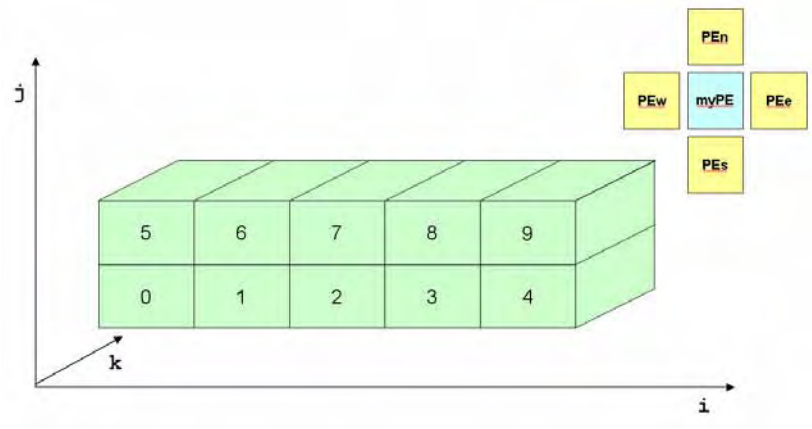
```
50                                  allocate(V(SX:EX,SY:EY,SZ:EZ),stat=ALLOC_ERR)
51                                  allocate(W(SX:EX,SY:EY,SZ:EZ),stat=ALLOC_ERR)
52                                  allocate(P(SX:EX,SY:EY,SZ:EZ),stat=ALLOC_ERR)
```

① Original 코드에서는 I, J 방향으로 영역을 분할 하는 반면, I, K 방향으로 영역을 분할한다.

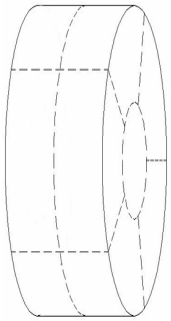
② 영역분할 후 각 영역별로 해당하는 array를 allocation하고 있다.

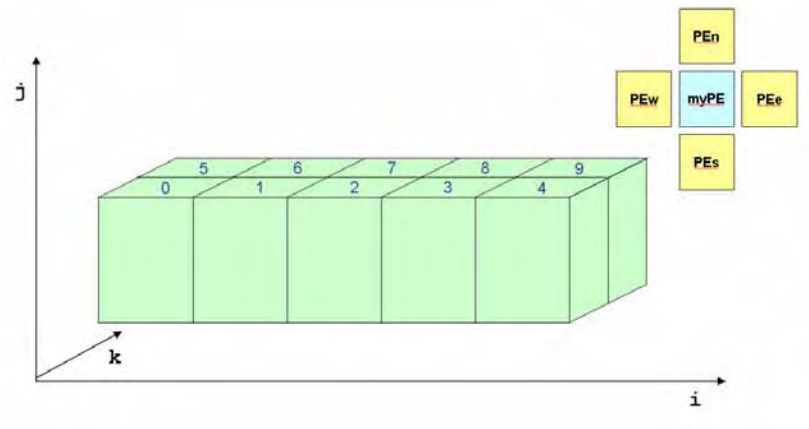
③ Original 코드의 영역분할 형태





④ 최적화 코드의 영역분할 형태





6.3 grid 루틴

A. 반복되는 계산과정 do loop 밖으로 이동 및 do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
53                  DO K=1,NZ
54                  DO I=1,NX

56                      if (i==nx) tht(i) = tht(i-1)
57            1         THT(i)=2.*ACOS(-1.0D0)*(i-1)/FLOAT(Nx-2)
63                  jS = 1
64                  jE = Ny_1
65                  DO j=jS,jE
66           20         RAD(j) = R1+(OUTS-R1)*(j-jS)/FLOAT(jE-jS)
```

```
 67          57              X(I,J,K) = -RAD(j)*COS(THT(i))
 68          73              Y(I,J,K) =   RAD(j)*SIN(THT(i))
 69          48              Z(I,J,K) =   HB+(HT-HB)*(K-1)/FLOAT(NZ-1)
 70                        ENDDO

 74           1        DO J = JS, JE
 75                    ETA_p = (DBLE(J)-JS)/(DBLE(JE)-JS)
 76          73        ETA(J) = ( (2.0D0*ALP-BET)+(2.0D0*ALP+BET) &
 77                    *(((BET+1.0D0)/(BET-1.0D0)))**((ETA_p-ALP)&
                       /(1.0D0-ALP)) ) &
 78                   /( (2.0D0*ALP+1.0D0)*( ((BET+1.0D0)/(BET-1.0D0)) &
 79                     **((ETA_p-ALP)/(1.0D0-ALP)) + 1.0D0 ) )
 80           1        END DO

 82                    DO J=JS,JE
 83                    T1_X(I) = X(I,JS,K)
 84                    T2_X(I) = X(I,JE,K)
 85                    T1_Y(I) = Y(I,JS,K)
 86          13        T2_Y(I) = Y(I,JE,K)
 87          24        X(I,J,K) = (1.0D0 - ETA(J))*T1_X(I) + ETA(J)*T2_X(I)
 88                    Y(I,J,K) = (1.0D0 - ETA(J))*T1_Y(I) + ETA(J)*T2_Y(I)
 89                    END DO

 95                    JS = NY_1
 96                    JE = NY_1+NY_2
 97                    DO J=JS, JE
 98           3              RAD(J)=OUTS+(OUTE-OUTS)*(J-JS)/FLOAT(JE-JS)
 99           4              X(I,J,K) =-RAD(J)*COS(THT(I))
100           6              Y(I,J,K) = RAD(J)*SIN(THT(I))
101          12              Z(I,J,K) = HB+(HT-HB)*(K-1)/FLOAT(NZ-1)
102                        ENDDO

108                    JS = NY_1+NY_2
109                    JE = NY_1+NY_2+NY_3
110                    DO J=JS,JE
111           2              RAD(J)=OUTE+(R2-OUTE)*(J-JS)/FLOAT(JE-JS)
112          11              X(I,J,K) = -RAD(J)*COS(THT(I))
113          14              Y(I,J,K) =   RAD(J)*SIN(THT(I))
114          11              Z(I,J,K) =   HB+(HT-HB)*(K-1)/FLOAT(NZ-1)
115                        END DO

119                    DO J = JS,JE
120                    ETA_p = (DBLE(J)-JS)/(DBLE(JE)-JS)
121       15           ETA(J) = ( (BET2+1.0D0)-(BET2-1.0D0)*((BET2+1.0D0)&
                       /(BET2-1.0D0))**(ETA_P) ) &
122                    / ( ((BET2+1.0D0)/(BET2-1.0D0))**(ETA_P)+1.0D0 )
123                    END DO

125                    DO J=JS,JE
```

```
 127                     T1_X(I) = X(I,JS,K)
 128                     T2_X(I) = X(I,JE,K)
 129                     T1_Y(I) = Y(I,JS,K)
 130                     T2_Y(I) = Y(I,JE,K)
 132                     X(I,J,K) = (1.0D0 - ETA(J))*T2_X(I) + ETA(J)*T1_X(I)
 133               2     Y(I,J,K) = (1.0D0 - ETA(J))*T2_Y(I) + ETA(J)*T1_Y(I)
 135                     END DO

 138                     END DO
 139                     END DO
```

<최적화 코드>

```
  52                              jS = 1
  53                              jE = Ny_1
  54                              DO j=jS,jE
  55                              RAD(j) = R1+(OUTS-R1)*(j-jS)/FLOAT(jE-jS)
  56                            ETA_p = (DBLE(J)-JS)/(DBLE(JE)-JS)
  57                            ETA(J) = ( (2.0D0*ALP-BET)+(2.0D0*ALP+BET) &
  58                            *(((BET+1.0D0)/(BET-1.0D0)))**((ETA_p-ALP)
                                      /(1.0D0-ALP)) ) &
  59                                    /( (2.0D0*ALP+1.0D0)
                                        *( ((BET+1.0D0)/(BET-1.0D0)) &
  60                                    **((ETA_p-ALP)/(1.0D0-ALP)) + 1.0D0 ) )
  61                              ENDDO
  62                              JS = NY_1
  63                              JE = NY_1+NY_2
  64                              DO J=JS, JE
  65                                 RAD(J)=OUTS+(OUTE-OUTS)*(J-JS)/FLOAT(JE-
JS)
  66                              ENDDO
  67                              JS = NY_1+NY_2
  68                              JE = NY_1+NY_2+NY_3
  69                              DO J=JS,JE
  70                              RAD(J)=OUTE+(R2-OUTE)*(J-JS)/FLOAT(JE-JS)
  71                            ETA_p = (DBLE(J)-JS)/(DBLE(JE)-JS)
  72                            ETA(J) = ( (BET2+1.0D0)-(BET2-1.0D0)
                                        *((BET2+1.0D0)
                                        /(BET2-1.0D0))**(ETA_P) ) &
  73                                    / ( ((BET2+1.0D0)/(BET2-1.0D0))
                                        **(ETA_P)+1.0D0 )
  74                              ENDDO

  78                            DO I=1,NX
  79                            if (i==nx) tht(i) = tht(i-1)
  80                            THT(i)=2.*ACOS(-1.0D0)*(i-1)/FLOAT(Nx-2)
  81                              COSTHT(I)=COS(THT(I))
```

```
 82                        SINTHT(I)=SIN(THT(I))
 83                       ENDDO

 85                       DO K=1,NZ

 90                       jS = 1
 91                       jE = Ny_1
 92                       DO j=jS,jE
 93                       DO I=1,NX
 94          87           X(I,J,K) = -RAD(j)*COSTHT(i)
 95         101           Y(I,J,K) =   RAD(j)*SINTHT(i)
 96         107           Z(I,J,K) =   HB+(HT-HB)*(K-1)/FLOAT(NZ-1)
 97                       ENDDO
 98                       ENDDO

102                       DO J=JS,JE
103                       DO I=1,NX
104                       T1_X(I) = X(I,JS,K)
105                       T2_X(I) = X(I,JE,K)
106                       T1_Y(I) = Y(I,JS,K)
107                       T2_Y(I) = Y(I,JE,K)
108          17           X(I,J,K) = (1.0D0 - ETA(J))*T1_X(I)
                                     + ETA(J)*T2_X(I)
109          17           Y(I,J,K) = (1.0D0 - ETA(J))*T1_Y(I)
                                     + ETA(J)*T2_Y(I)
110                       END DO
111                       END DO

117                       JS = NY_1
118                       JE = NY_1+NY_2
120                       DO J=JS, JE
121                       DO I=1,NX
122           7           X(I,J,K) =-RAD(J)*COSTHT(I)
123           4           Y(I,J,K) = RAD(J)*SINTHT(I)
124          11           Z(I,J,K) = HB+(HT-HB)*(K-1)/FLOAT(NZ-1)
125                       ENDDO
126                       ENDDO

132                       JS = NY_1+NY_2
133                       JE = NY_1+NY_2+NY_3
135                       DO J=JS,JE
136                       DO I=1,NX
137          16           X(I,J,K) = -RAD(J)*COSTHT(I)
138          19           Y(I,J,K) =   RAD(J)*SINTHT(I)
139          21           Z(I,J,K) =   HB+(HT-HB)*(K-1)/FLOAT(NZ-1)
140                       END DO
141                       END DO

145                       DO J=JS,JE
```

```
146                     DO I=1,NX
148                     T1_X(I) = X(I,JS,K)
149                     T2_X(I) = X(I,JE,K)
150                     T1_Y(I) = Y(I,JS,K)
151                     T2_Y(I) = Y(I,JE,K)
153           1         X(I,J,K) = (1.0D0 - ETA(J))*T2_X(I)
                                   + ETA(J)*T1_X(I)
154           3         Y(I,J,K) = (1.0D0 - ETA(J))*T2_Y(I)
                                   + ETA(J)*T1_Y(I)
156                     END DO
157                     END DO

160                     END DO
```

① Original 코드의 K, I, J loop 안에서 계산되는 RAD(J), ETA(J)를 K,I loop와는 상관 없기 때문에 최적화 코드에서는 전체 loop에서 분리하여 따로 J loop만 수행하면서 RAD(J), ETA(J)가 계산되도록 하여 전체 계산량을 줄였다.

② RAD(J), ETA(J) 계산과 동일하게 Original 코드에서는 K,I,J loop 안에서 THT(i) 및 COS(THT(I)), SIN(THT(I))를 계산하고 있지만, 최적화 코드에서는 이를 분리하여 I loop만 따로 수행하면서 THT(I)를 구하고, COS(THT(I)) 및 SIN(THT(I)) 값을 COSTHT(I) 및 SINTHT(I)에 저장하여 전체 계산량을 줄였다.

③ X(I,J,K), Y(I,J,K), Z(I,J,K)를 계산할 때 Original 코드에서 J,I,K loop 순으로 수행됨으로써 발생되는 비연속적인 메모리 access를 피하기 위해서 최적화 코드에서는 I,J,K loop 순으로 수행순서를 바꾸어 연속적인 메모리 access를 하도록 하였다.

④ 최적화 코드에서 추가로 Z(I,J,K)를 계산할 때 HB+(HT-HB)*(K-1)/FLOAT(NZ-1)의 값은 K에 의해서만 결정되기 때문에 I,J loop에서 빼내어 계산할 수 있다.

6.4 metric 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
508                         DO I=SX,EX
509                         DO J=SY,EY
510                         DO K=SZ,EZ
511          404             DJAC(I,J,K)      &
512                         =   XXI(I,J,K)*(YET(I,J,K)*ZZT(I,J,K)-YZT(I,J,K)
                                *ZET(I,J,K))      &
513                              -YXI(I,J,K)*(XET(I,J,K)*ZZT(I,J,K)-ZET(I,J,K)
                                *XZT(I,J,K))      &
514                              +ZXI(I,J,K)*(XET(I,J,K)*YZT(I,J,K)-XZT(I,J,K)
                                *YET(I,J,K))
515
516                             IF (ABS(DJAC(I,J,K)).LT.1.0E-15) THEN
517                                WRITE(6,*) I,J,K,DJAC(I,J,K)
518                                STOP
519                           ENDIF
520               !          IF ((DJAC(I,J,K)).LE.0.0) WRITE(6,*) I,J,K,DJAC(I,J,K)
521                         ENDDO
522                         ENDDO
523                         ENDDO

526                         DO I=SX,EX
527                         DO J=SY,EY
528                         DO K=SZ,EZ
529          270             XIX(I,J,K) = (YET(I,J,K)*ZZT(I,J,K)-YZT(I,J,K)
                                        *ZET(I,J,K))/DJAC(I,J,K)
530          267             XIY(I,J,K) =-(XET(I,J,K)*ZZT(I,J,K)-XZT(I,J,K)
                                        *ZET(I,J,K))/DJAC(I,J,K)
531           39             XIZ(I,J,K) = (XET(I,J,K)*YZT(I,J,K)-XZT(I,J,K)
                                        *YET(I,J,K))/DJAC(I,J,K)
532
533          222             ETX(I,J,K) =-(YXI(I,J,K)*ZZT(I,J,K)-YZT(I,J,K)
                                        *ZXI(I,J,K))/DJAC(I,J,K)
534           32             ETY(I,J,K) = (XXI(I,J,K)*ZZT(I,J,K)-XZT(I,J,K)
                                        *ZXI(I,J,K))/DJAC(I,J,K)
535          157             ETZ(I,J,K) =-(XXI(I,J,K)*YZT(I,J,K)-XZT(I,J,K)
                                        *YXI(I,J,K))/DJAC(I,J,K)
536
537           64             ZTX(I,J,K) = (YXI(I,J,K)*ZET(I,J,K)-YET(I,J,K)
                                        *ZXI(I,J,K))/DJAC(I,J,K)
538           47             ZTY(I,J,K) =-(XXI(I,J,K)*ZET(I,J,K)-XET(I,J,K)
                                        *ZXI(I,J,K))/DJAC(I,J,K)
539           98             ZTZ(I,J,K) = (XXI(I,J,K)*YET(I,J,K)-XET(I,J,K)
                                        *YXI(I,J,K))/DJAC(I,J,K)
540                        ENDDO
541            1           ENDDO
542                        ENDDO
```

```
545                         DO I=SX,EX
546                         DO J=SY,EY
547              1          DO K=SZ,EZ
548            209       Q11(I,J,K)= XIX(I,J,K)*XIX(I,J,K)+XIY(I,J,K)
                                    *XIY(I,J,K)+XIZ(I,J,K)*XIZ(I,J,K)
549            248       Q12(I,J,K)= XIX(I,J,K)*ETX(I,J,K)+XIY(I,J,K)
                                    *ETY(I,J,K)+XIZ(I,J,K)*ETZ(I,J,K)
550            224       Q13(I,J,K)= XIX(I,J,K)*ZTX(I,J,K)+XIY(I,J,K)
                                    *ZTY(I,J,K)+XIZ(I,J,K)*ZTZ(I,J,K)
551
552             66       Q21(I,J,K)= Q12(I,J,K)
553             34       Q22(I,J,K)= ETX(I,J,K)*ETX(I,J,K)+ETY(I,J,K)
                                   *ETY(I,J,K)+ETZ(I,J,K)*ETZ(I,J,K)
554            173       Q23(I,J,K)= ETX(I,J,K)*ZTX(I,J,K)+ETY(I,J,K)
                                   *ZTY(I,J,K)+ETZ(I,J,K)*ZTZ(I,J,K)
555
556             65       Q31(I,J,K)= Q13(I,J,K)
557             28       Q32(I,J,K)= Q23(I,J,K)
558             95       Q33(I,J,K)= ZTX(I,J,K)*ZTX(I,J,K)+ZTY(I,J,K)
                                   *ZTY(I,J,K)+ZTZ(I,J,K)*ZTZ(I,J,K)
559              6             ENDDO
560              2             ENDDO
561                           ENDDO
562
```

<최적화 코드>

```
512                         DO K=SZ,EZ
513                         DO J=SY,EY
514                         DO I=SX,EX
515             35              DJAC(I,J,K)      &
516                             =   XXI(I,J,K)*(YET(I,J,K)*ZZT(I,J,K)
                                   -YZT(I,J,K)*ZET(I,J,K))      &
517                                -YXI(I,J,K)*(XET(I,J,K)*ZZT(I,J,K)
                                   -ZET(I,J,K)*XZT(I,J,K))      &
518                                +ZXI(I,J,K)*(XET(I,J,K)*YZT(I,J,K)
                                   -XZT(I,J,K)*YET(I,J,K))
519
520                             IF (ABS(DJAC(I,J,K)).LT.1.0E-15) THEN
521                                 WRITE(6,*) I,J,K,DJAC(I,J,K)
522                                 STOP
523                           ENDIF
524               !         IF ((DJAC(I,J,K)).LE.0.0) WRITE(6,*) I,J,K,DJAC(I,J,K)
525                         ENDDO
526                         ENDDO
527                         ENDDO
```

```
530                     DO K=SZ,EZ
531           1         DO J=SY,EY
532                     DO I=SX,EX
533          12         XIX(I,J,K) = (YET(I,J,K)*ZZT(I,J,K)-YZT(I,J,K)
                                    *ZET(I,J,K))/DJAC(I,J,K)
534          13         XIY(I,J,K) =-(XET(I,J,K)*ZZT(I,J,K)-XZT(I,J,K)
                                    *ZET(I,J,K))/DJAC(I,J,K)
535           9         XIZ(I,J,K) = (XET(I,J,K)*YZT(I,J,K)-XZT(I,J,K)
                                    *YET(I,J,K))/DJAC(I,J,K)
536
537          18         ETX(I,J,K) =-(YXI(I,J,K)*ZZT(I,J,K)-YZT(I,J,K)
                                    *ZXI(I,J,K))/DJAC(I,J,K)
538          12         ETY(I,J,K) = (XXI(I,J,K)*ZZT(I,J,K)-XZT(I,J,K)
                                    *ZXI(I,J,K))/DJAC(I,J,K)
539          13         ETZ(I,J,K) =-(XXI(I,J,K)*YZT(I,J,K)-XZT(I,J,K)
                                    *YXI(I,J,K))/DJAC(I,J,K)
540
541          21         ZTX(I,J,K) = (YXI(I,J,K)*ZET(I,J,K)-YET(I,J,K)
                                    *ZXI(I,J,K))/DJAC(I,J,K)
542          21         ZTY(I,J,K) =-(XXI(I,J,K)*ZET(I,J,K)-XET(I,J,K)
                                    *ZXI(I,J,K))/DJAC(I,J,K)
543          15         ZTZ(I,J,K) = (XXI(I,J,K)*YET(I,J,K)-XET(I,J,K)
                                    *YXI(I,J,K))/DJAC(I,J,K)
544                     ENDDO
545           1         ENDDO
546                     ENDDO

549                     DO K=SZ,EZ
550           1         DO J=SY,EY
551                     DO I=SX,EX
552          14   Q11(I,J,K)= XIX(I,J,K)*XIX(I,J,K)+XIY(I,J,K)
                              *XIY(I,J,K)+XIZ(I,J,K)*XIZ(I,J,K)
553          20   Q12(I,J,K)= XIX(I,J,K)*ETX(I,J,K)+XIY(I,J,K)
                              *ETY(I,J,K)+XIZ(I,J,K)*ETZ(I,J,K)
554          18   Q13(I,J,K)= XIX(I,J,K)*ZTX(I,J,K)+XIY(I,J,K)
                              *ZTY(I,J,K)+XIZ(I,J,K)*ZTZ(I,J,K)
555
556          16   Q21(I,J,K)= Q12(I,J,K)
557          11   Q22(I,J,K)= ETX(I,J,K)*ETX(I,J,K)+ETY(I,J,K)
                              *ETY(I,J,K)+ETZ(I,J,K)*ETZ(I,J,K)
558          12   Q23(I,J,K)= ETX(I,J,K)*ZTX(I,J,K)+ETY(I,J,K)
                              *ZTY(I,J,K)+ETZ(I,J,K)*ZTZ(I,J,K)
559
560          14   Q31(I,J,K)= Q13(I,J,K)
561          13   Q32(I,J,K)= Q23(I,J,K)
562          11   Q33(I,J,K)= ZTX(I,J,K)*ZTX(I,J,K)+ZTY(I,J,K)
                              *ZTY(I,J,K)+ZTZ(I,J,K)*ZTZ(I,J,K)
563                     ENDDO
564                     ENDDO
```

```
565                                   ENDDO
```

① Original 코드에서 1차원으로 선언한 후 3개의 각 do-loop에서 반복해서 동일하게 계산되는 visc array를 serial 최적화 코드에서는 3차원 array로 선언한 후 한번만 계산하여 사용하였다.

6.5 exchange 루틴

A. MPI_Sendrecv 통신

<Original 코드>

```
19     1            DO K=SZ,EZ
23     4            CALL MPI_Sendrecv(val(EX1   ,SY1-1,K),1,COLUMN0,PEe,2, &
24                  val(SX1-1,SY1-1,K),1,COLUMN0,PEw,2,comm2d,status,ierr)

26    13            CALL MPI_Sendrecv(val(SX1   ,SY1-1,K),1,COLUMN0,PEw,3, &
27                  val(EX1+1,SY1-1,K),1,COLUMN0,PEe,3,comm2d,status,ierr)

31    13            CALL MPI_Sendrecv(val(SX1-1,   EY1,K),1,ROW0,PEn,0, &
32                  val(SX1-1,SY1-1,K),1,ROW0,PEs,0,comm2d,status,ierr)

34     6            CALL MPI_Sendrecv(Val(SX1-1,   SY1,K),1,ROW0,PEs,1, &
35                   val(SX1-1,EY1+1,K),1,ROW0,PEn,1,comm2d,status,ierr)
37               2       ENDDO
```

main.f90에서 column0, row0 MPI derived type 생성

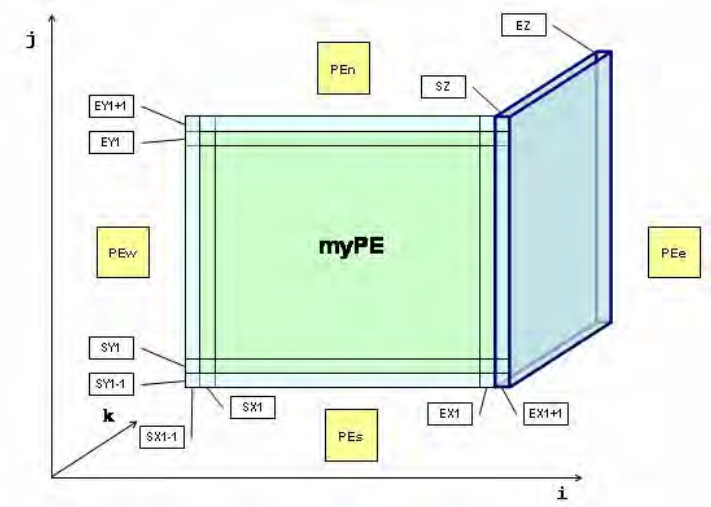
```
73                    count=EY1-SY1+3
74                    block=1
75                    stride=EX1-SX1+3

77                    call MPI_Type_vector(count,block,stride,
                                MPI_DOUBLE_PRECISION,column0,ierr)
78                    call MPI_Type_commit(column0,ierr)

80                    call MPI_Type_contiguous(EX1-SX1+3,
                                MPI_DOUBLE_PRECISION,row0,ierr)
81                    call MPI_Type_commit(row0,ierr)
```

<최적화 코드>

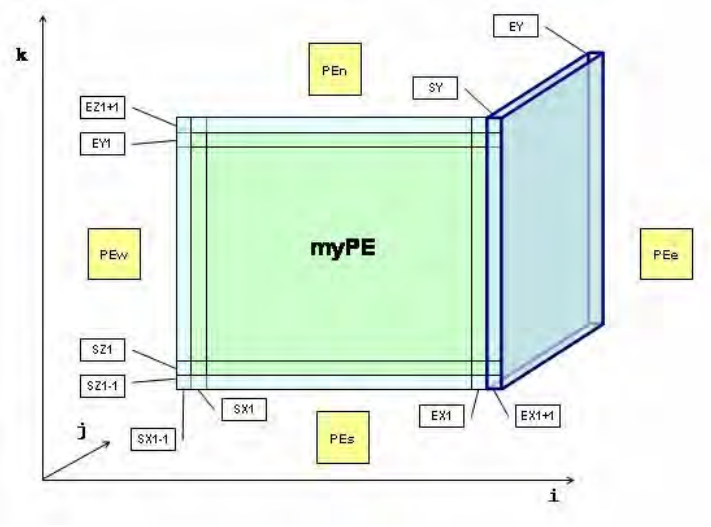
```
24                          icount=1
25          2               DO K=SZ,EZ
26                          DO J=SY,EY
27       1605                   BUFFSY(icount)=val(EX1,J,K)
28                              icount=icount+1
29         20               ENDDO
30                          ENDDO
31                          len=(EY-SY+1)*(EZ-SZ+1)
32          1             CALL MPI_Sendrecv(BUFFSY,len,
                            MPI_DOUBLE_PRECISION,PEe,2, &
33                             BUFFRY,len,MPI_DOUBLE_PRECISION,PEw,2,
                            comm2d,status,ierr)
34                          icount=1
35          2               DO K=SZ,EZ
36                          DO J=SY,EY
37        565                   val(SX,J,K)=BUFFRY(icount)
38         73                   icount=icount+1
39         79               ENDDO
40          8               ENDDO
41
42                          icount=1
43          2               DO K=SZ,EZ
44                          DO J=SY,EY
45        367                   BUFFSY(icount)=val(SX1,J,K)
46          6                   icount=icount+1
47          3               ENDDO
48                          ENDDO
49                          len=(EY-SY+1)*(EZ-SZ+1)
50                          CALL MPI_Sendrecv(BUFFSY,len,
                            MPI_DOUBLE_PRECISION,PEw,3, &
51                              BUFFRY,len,MPI_DOUBLE_PRECISION,PEe,
                                3,comm2d,status,ierr)
52                          icount=1
53                          DO K=SZ,EZ
54                          DO J=SY,EY
55        412                   val(EX,J,K)=BUFFRY(icount)
56         58                   icount=icount+1
57         63               ENDDO
58          7               ENDDO
59                    !---TOP&BOTTOM
60          1               if(PEn.ne.MPI_PROC_NULL) then
61                          icount=1
62          1               DO J=SY,EY
63                          DO I=SX,EX
```

```
 64          203                BUFFSX(icount)=val(I,J,EZ1)
 65                             icount=icount+1
 66           20               ENDDO
 67            7               ENDDO
 68                            endif
 69                            len=(EX-SX+1)*(EY-SY+1)
 70            2               CALL MPI_Sendrecv(BUFFSX,len,
                                    MPI_DOUBLE_PRECISION,PEn,0, &
 71                             BUFFRX,len,MPI_DOUBLE_PRECISION,
                                    PEs,0,comm2d,status,ierr)
 72                            if(PEs.ne.MPI_PROC_NULL) then
 73                            icount=1
 74                            DO J=SY,EY
 75                            DO I=SX,EX
 76          113                val(I,J,SZ)=BUFFRX(icount)
 77                             icount=icount+1
 78           15               ENDDO
 79            1               ENDDO
 80                            endif
 81
 82                            if(PEs.ne.MPI_PROC_NULL) then
 83                            icount=1
 84                            DO J=SY,EY
 85                            DO I=SX,EX
 86          184                BUFFSX(icount)=val(I,J,SZ1)
 87                             icount=icount+1
 88           19               ENDDO
 89           11               ENDDO
 90                            endif
 91                            len=(EX-SX+1)*(EY-SY+1)
 92                            CALL MPI_Sendrecv(BUFFSX,len,
                                  MPI_DOUBLE_PRECISION,PEs,1, &
 93                               BUFFRX,len,MPI_DOUBLE_PRECISION,
                                    PEn,1,comm2d,status,ierr)
 94
 95                            if(PEn.ne.MPI_PROC_NULL) then
 96                            icount=1
 97            1               DO J=SY,EY
 98                            DO I=SX,EX
 99          104                val(I,J,EZ)=BUFFRX(icount)
100            1                icount=icount+1
101           10               ENDDO
102            8               ENDDO
103                            endif
```

① 각 myPE(rank)에 대해 다음 그림과 같이 가장자리에 있는 부분에 대해 통신을 하게 된다.

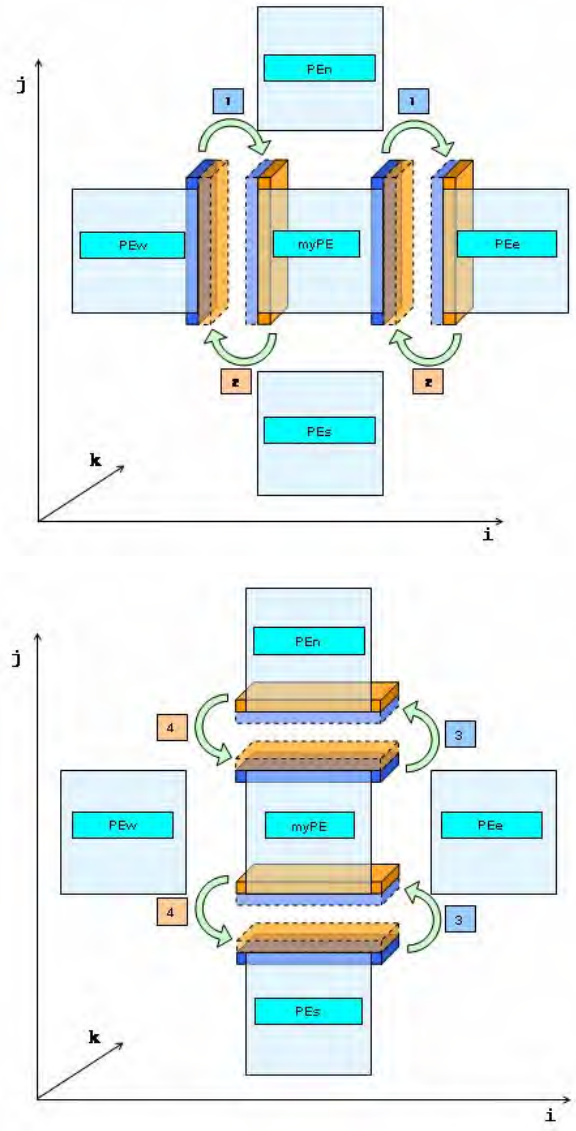


< 그림 II.5 Original 코드에서 통신이 필요한 부분 >



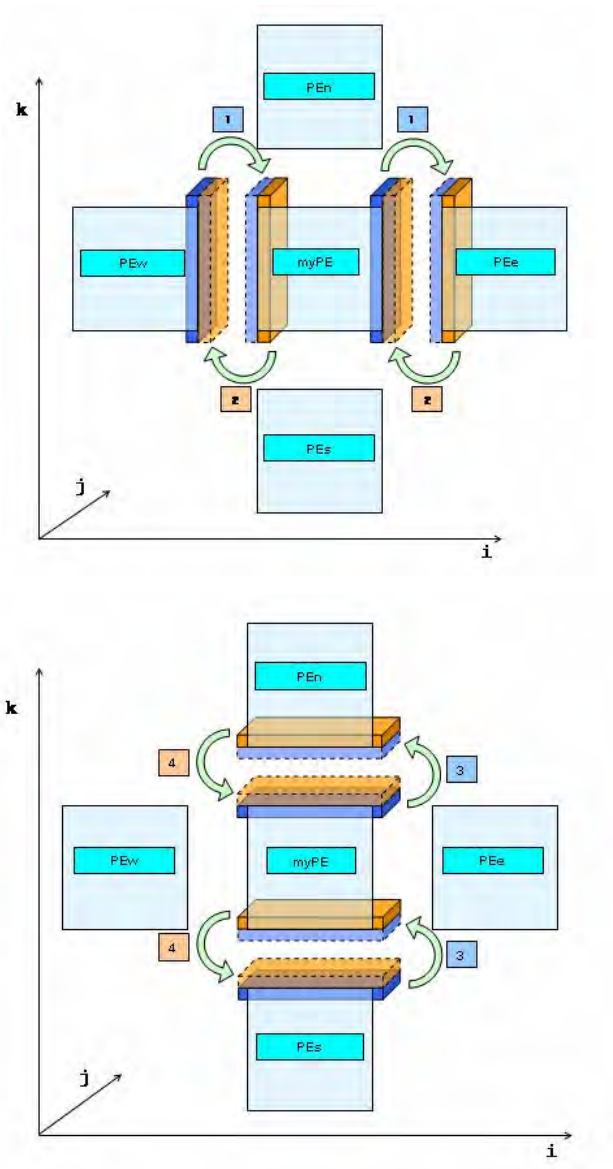
< 그림 II. 6 최적화 코드에서 통신이 필요한 부분 >

② Original 코드에서는 먼저 I 방향으로 통신을 한 후 J 방향으로 통신을 하게 된다.



< 그림 II.7 Original 코드의 통신 방법 >

③ 최적화 코드에서는 먼저 I 방향으로 통신을 한 후 K 방향으로 통신을 하게 된다.



< 그림 II.8 최적화 코드의 통신 방법 >

④ Original 코드는 main.f90에서 생성한 COLUMN0, ROW0의 MPI derived type을 이용하여 k방향으로 각각 MPI_Sendrecv를 call

하여 데이터 전송을 수행하지만, 최적화 코드에서는 전송한 데이터를 BUFFSY, BUFFSX에 저장한 후 각각 한번씩만 MPI_Sendrecv를 call하여 데이터 전송을 수행한다.

⑤ 즉 Original 코드에서는 적은 양의 데이터를 MPI_Sendrecv를 여러 번 call하여 통신을 하지만, 최적화 코드에서는 한번에 많은 양의 데이터를 해당 array에 저장한 다음 한번의 MPI_Sendrecv를 call하여 통신을 하게 된다.

⑥ 이렇게 수정된 각각의 코드에 대해 MPI_Sendrecv 통신시간을 비교해 보면 다음과 같다. 참고로 MPI_Sendrecv는 exchange 루틴에서만 사용하기 때문에 직접적으로 비교해 볼 수 있다.

A. Original 코드에 mpitrace(rank=4) 결과.

```
-----------------------------------------------------------------
MPI Routine                #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Sendrecv               2379600          560.0       430.532
-----------------------------------------------------------------
Message size distributions:

MPI_Sendrecv               #calls     avg. bytes      time(sec)
                           1189800          448.0       321.142
                           1189800          672.0       109.390
```

B. 최적화 코드의 mpitrace(rank=4) 결과

```
-----------------------------------------------------------------
MPI Routine                #calls     avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Sendrecv                 18768        59400.0        19.150
-----------------------------------------------------------------
Message size distributions:

MPI_Sendrecv               #calls     avg. bytes      time(sec)
                             9384         44880.0        14.799
                             9384         73920.0         4.351
```

⑦ MPI_Sendrecv 통신 시간만 놓고 볼 때 약 20배 이상 통신시간을 줄일 수 있음을 확인할 수 있다.

6.6 coeff 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
 21                      DO I=SX,EX
 22          3           DO J=SY,EY
 23         24           DO K=SZ,EZ
 24      60746               U_cont(I,J,K) = (XIX(I,J,K)*U_N(I,J,K)
                              +XIY(I,J,K)*V_N(I,J,K)+
 25      14919               V_cont(I,J,K) = (ETX(I,J,K)*U_N(I,J,K)
                              +ETY(I,J,K)*V_N(I,J,K)+
 26      21661               W_cont(I,J,K) = (ZTX(I,J,K)*U_N(I,J,K)
                              +ZTY(I,J,K)*V_N(I,J,K)+
 27                      ENDDO
 28         69           ENDDO
 29                      ENDDO
 30
 31                      DO I=SX1,EX1
 32          3           DO J=SY1,EY1
 33          5           DO K=SZ1,EZ1
 34        346     U_cont_E(I,J,K)=(U_cont(I,J,K)+U_cont(I+1,J,K))/2.0D0
 35       3073     U_cont_W(I,J,K)=(U_cont(I,J,K)+U_cont(I-1,J,K))/2.0D0
 36       5524     V_cont_N(I,J,K) = (V_cont(I,J,K)+V_cont(I,J+1,K))/2.0D0
 37       1415     V_cont_S(I,J,K) = (V_cont(I,J,K)+V_cont(I,J-1,K))/2.0D0
 38       5734     W_cont_T(I,J,K) = (W_cont(I,J,K)+W_cont(I,J,K+1))/2.0D0
 39       5817     W_cont_B(I,J,K) = (W_cont(I,J,K)+W_cont(I,J,K-1))/2.0D0
 40                      ENDDO
 41         26           ENDDO
 42          3           ENDDO
 43
 44                      DO I=SX1,EX1
 45         12           DO J=SY1,EY1
 46        186           DO K=SZ1,EZ1
 47
 48      14314          DE(I,J,K) = 0.5D0*(DJAC(I,J,K)*Q11(I,J,K)
                         +DJAC(I+1,J,K)*Q11(I+1,J,K))
 49      33657          DW(I,J,K) = 0.5D0*(DJAC(I,J,K)*Q11(I,J,K)
                         +DJAC(I-1,J,K)*Q11(I-1,J,K))
 50      13791          DN(I,J,K) = 0.5D0*(DJAC(I,J,K)*Q22(I,J,K)
                         +DJAC(I,J+1,K)*Q22(I,J+1,K))
 51       8069          DS(I,J,K) = 0.5D0*(DJAC(I,J,K)*Q22(I,J,K)
                         +DJAC(I,J-1,K)*Q22(I,J-1,K))
 52      26085          DT(I,J,K) = 0.5D0*(DJAC(I,J,K)*Q33(I,J,K)
                         +DJAC(I,J,K+1)*Q33(I,J,K+1))
 53      12844          DB(I,J,K) = 0.5D0*(DJAC(I,J,K)*Q33(I,J,K)
                         +DJAC(I,J,K-1)*Q33(I,J,K-1))
```

```
 54
 55          6              AE(I,J,K)=DE(I,J,K)
 56       6548              AW(I,J,K)=DW(I,J,K)
 57      15361              AN(I,J,K)=DN(I,J,K)
 58         49              AS(I,J,K)=DS(I,J,K)
 59       8203              AT(I,J,K)=DT(I,J,K)
 60       7467              AB(I,J,K)=DB(I,J,K)
 61
 62         48                    NLX_N2(I,J,K)=NLX_N1(I,J,K)
 63         51                    NLY_N2(I,J,K)=NLY_N1(I,J,K)
 64         92                    NLZ_N2(I,J,K)=NLZ_N1(I,J,K)
 65
 66         26                    NLX_N1(I,J,K)=NLX_N(I,J,K)
 67       7389                    NLY_N1(I,J,K)=NLY_N(I,J,K)
 68       7330                    NLZ_N1(I,J,K)=NLZ_N(I,J,K)
 69
 70      87416                  NLX_N(I,J,K)=0.5D0/DJAC(I,J,K)    &
 71                            *(U_cont_E(I,J,K)*(U_N(I,J,K)+U_N(I+1,J,K)) &
 72                            -U_cont_W(I,J,K)*(U_N(I,J,K)+U_N(I-1,J,K)) &
 73                            +V_cont_N(I,J,K)*(U_N(I,J,K)+U_N(I,J+1,K)) &
 74                            -V_cont_S(I,J,K)*(U_N(I,J,K)+U_N(I,J-1,K)) &
 75                            +W_cont_T(I,J,K)*(U_N(I,J,K)+U_N(I,J,K+1)) &
 76                            -W_cont_B(I,J,K)*(U_N(I,J,K)+U_N(I,J,K-1)))
 77
 78      60311                  NLY_N(I,J,K)=0.5D0/DJAC(I,J,K) &
 79                            *(U_cont_E(I,J,K)*(V_N(I,J,K)+V_N(I+1,J,K)) &
 80                            -U_cont_W(I,J,K)*(V_N(I,J,K)+V_N(I-1,J,K)) &
 81                            +V_cont_N(I,J,K)*(V_N(I,J,K)+V_N(I,J+1,K)) &
 82                            -V_cont_S(I,J,K)*(V_N(I,J,K)+V_N(I,J-1,K)) &
 83                            +W_cont_T(I,J,K)*(V_N(I,J,K)+V_N(I,J,K+1)) &
 84                            -W_cont_B(I,J,K)*(V_N(I,J,K)+V_N(I,J,K-1)))

136      20508                       DIFZ(I,J,K)=1.0D0/RE/DJAC(I,J,K)&
137                                 *(DE(I,J,K)*(W_N(I+1,J,K)-W_N(I,J,K)) &
138                                 -DW(I,J,K)*(W_N(I,J,K)   -W_N(I-1,J,K)) &
139                                 +DN(I,J,K)*(W_N(I,J+1,K)-W_N(I,J,K)) &
140                                 -DS(I,J,K)*(W_N(I,J,K)   -W_N(I,J-1,K))&
141                                 +DT(I,J,K)*(W_N(I,J,K+1)-W_N(I,J,K)) &
142                                 -DB(I,J,K)*(W_N(I,J,K)   -W_N(I,J,K-1)))&
143                                 +0.25D0/RE/DJAC(I,J,K)&
144          *(Q12(I+1,J,K)*DJAC(I+1,J,K)*(W_N(I+1,J+1,K)-W_N(I+1,J-1,K))
145          -Q12(I-1,J,K)*DJAC(I-1,J,K)*(W_N(I-1,J+1,K)-W_N(I-1,J-1,K))&
146          +Q13(I+1,J,K)*DJAC(I+1,J,K)*(W_N(I+1,J,K+1)-W_N(I+1,J,K-1))&
147          -Q13(I-1,J,K)*DJAC(I-1,J,K)*(W_N(I-1,J,K+1)-W_N(I-1,J,K-1))&
148          +Q21(I,J+1,K)*DJAC(I,J+1,K)*(W_N(I+1,J+1,K)-W_N(I-1,J+1,K))&
149          -Q21(I,J-1,K)*DJAC(I,J-1,K)*(W_N(I+1,J-1,K)-W_N(I-1,J-1,K))&
150          +Q23(I,J+1,K)*DJAC(I,J+1,K)*(W_N(I,J+1,K+1)-W_N(I,J+1,K-1)) &
151          -Q23(I,J-1,K)*DJAC(I,J-1,K)*(W_N(I,J-1,K+1)-W_N(I,J-1,K-1)) &
152          +Q31(I,J,K+1)*DJAC(I,J,K+1)*(W_N(I+1,J,K+1)-W_N(I-1,J,K+1)) &
```

```
153             -Q31(I,J,K-1)*DJAC(I,J,K-1)*(W_N(I+1,J,K-1)-W_N(I-1,J,K-1)) &
154             +Q32(I,J,K+1)*DJAC(I,J,K+1)*(W_N(I,J+1,K+1)-W_N(I,J-1,K+1)) &
155             -Q32(I,J,K-1)*DJAC(I,J,K-1)*(W_N(I,J+1,K-1)-W_N(I,J-1,K-1)))
156                           ENDDO
157              78           ENDDO
158                           ENDDO
```

<최적화 코드>

```
27               1           DO K=SZ,EZ
28              12           DO J=SY1,EY1
29               8           DO I=SX1,EX1
30            1929           W_conts(I,J,K) = (ZTX(I,J,K)*U_N(I,J,K)
                             +ZTY(I,J,K)*V_N(I,J,K)+
31               8           ENDDO
32               1           ENDDO
33                           ENDDO
34
35               2           DO K=SZ1,EZ1
36
37               9           DO J=SY,EY
38               4           DO I=SX1,EX1
39            1090             V_conts(I,J) = (ETX(I,J,K)*U_N(I,J,K)
                                 +ETY(I,J,K)*V_N(I,J,K)+
40                           ENDDO
41               3           ENDDO
42
43              21           DO J=SY1,EY1
44
45                           DO I=SX,EX
46             831             U_conts(I) = (XIX(I,J,K)*U_N(I,J,K)
                              +XIY(I,J,K)*V_N(I,J,K)+
47                           ENDDO
48
49             157           DO I=SX1,EX1
50            1046                U_cont_Es = U_conts(I)+U_conts(I+1)
51              34                U_cont_Ws = U_conts(I)+U_conts(I-1)
52            1711                V_cont_Ns = V_conts(I,J)+V_conts(I,J+1)
53             144                V_cont_Ss = V_conts(I,J)+V_conts(I,J-1)
54             581                W_cont_Ts = W_conts(I,J,K)
                                    +W_conts(I,J,K+1)
55              60                W_cont_Bs = W_conts(I,J,K)+W_conts(I,J,K-1)
56
57            1276               DEs = 0.5D0*(DJAC(I,J,K)*Q11(I,J,K)
                                  +DJAC(I+1,J,K)*Q11(I+1,J,K))
58             490               DWs = 0.5D0*(DJAC(I,J,K)*Q11(I,J,K)
                                  +DJAC(I-1,J,K)*Q11(I-1,J,K))
```

```
 59          208                DNs = 0.5D0*(DJAC(I,J,K)*Q22(I,J,K)
                                +DJAC(I,J+1,K)*Q22(I,J+1,K))
 60          173                DSs = 0.5D0*(DJAC(I,J,K)*Q22(I,J,K)
                                +DJAC(I,J-1,K)*Q22(I,J-1,K))
 61         2489                DTs = 0.5D0*(DJAC(I,J,K)*Q33(I,J,K)
                                +DJAC(I,J,K+1)*Q33(I,J,K+1))
 62          309                DBs = 0.5D0*(DJAC(I,J,K)*Q33(I,J,K)
                                +DJAC(I,J,K-1)*Q33(I,J,K-1))
 63
 64          812               AE(I,J,K)=DEs
 65          133               AW(I,J,K)=DWs
 66          223               AN(I,J,K)=DNs
 67          164               AS(I,J,K)=DSs
 68           64               AT(I,J,K)=DTs
 69          322               AB(I,J,K)=DBs
 70           66               APS(I,J,K)=DNs+DSs+DEs+DWs+DTs+DBs
 71
 72          275                    NLX_N2(I,J,K)=NLX_N1(I,J,K)
 73           73                    NLY_N2(I,J,K)=NLY_N1(I,J,K)
 74          436                    NLZ_N2(I,J,K)=NLZ_N1(I,J,K)
 75
 76            3                   NLX_N1(I,J,K)=NLX_N(I,J,K)
 77                                NLY_N1(I,J,K)=NLY_N(I,J,K)
 78          342                    NLZ_N1(I,J,K)=NLZ_N(I,J,K)
 79
 80         1921                    NLX_N(I,J,K)=0.25D0/DJAC(I,J,K)    &
 81                                *( U_cont_Es*(U_N(I,J,K)+U_N(I+1,J,K))&
 82                                -U_cont_Ws*(U_N(I,J,K)+U_N(I-1,J,K)) &
 83                                +V_cont_Ns*(U_N(I,J,K)+U_N(I,J+1,K))&
 84                                -V_cont_Ss*(U_N(I,J,K)+U_N(I,J-1,K))&
 85                                +W_cont_Ts*(U_N(I,J,K)+U_N(I,J,K+1))&
 86                                -W_cont_Bs*(U_N(I,J,K)+U_N(I,J,K-1)))
 87
 88         3303                    NLY_N(I,J,K)=0.25D0/DJAC(I,J,K) &
 89                                *( U_cont_Es*(V_N(I,J,K)+V_N(I+1,J,K))&
 90                                -U_cont_Ws*(V_N(I,J,K)+V_N(I-1,J,K))&
 91                                +V_cont_Ns*(V_N(I,J,K)+V_N(I,J+1,K))&
 92                                -V_cont_Ss*(V_N(I,J,K)+V_N(I,J-1,K))&
 93                                +W_cont_Ts*(V_N(I,J,K)+V_N(I,J,K+1))&
 94                                -W_cont_Bs*(V_N(I,J,K)+V_N(I,J,K-1)))


146         4701                    DIFZ(I,J,K)=1.0D0/RE/DJAC(I,J,K)&
147                                       *(DEs*(W_N(I+1,J,K)-W_N(I,J,K))&
148                                      -DWs*(W_N(I,J,K)    -W_N(I-1,J,K))&
149                                      +DNs*(W_N(I,J+1,K)-W_N(I,J,K))&
150                                      -DSs*(W_N(I,J,K)    -W_N(I,J-1,K))&
151                                      +DTs*(W_N(I,J,K+1)-W_N(I,J,K))&
152                                      -DBs*(W_N(I,J,K)    -W_N(I,J,K-1)))&
```

```
153                                                    +0.25D0/RE/DJAC(I,J,K)
154           *((Q12(I+1,J,K)*(W_N(I+1,J+1,K)-W_N(I+1,J-1,K))
155           +Q13(I+1,J,K)*(W_N(I+1,J,K+1)-W_N(I+1,J,K-1)) )*DJAC(I+1,J,K)&
156           -(Q12(I-1,J,K)*(W_N(I-1,J+1,K)-W_N(I-1,J-1,K))        &
157           +Q13(I-1,J,K)*(W_N(I-1,J,K+1)-W_N(I-1,J,K-1)) )*DJAC(I-1,J,K)&
158           +(Q21(I,J+1,K)*(W_N(I+1,J+1,K)-W_N(I-1,J+1,K))        &
159           +Q23(I,J+1,K)*(W_N(I,J+1,K+1)-W_N(I,J+1,K-1)) )*DJAC(I,J+1,K)&
160           -(Q21(I,J-1,K)*(W_N(I+1,J-1,K)-W_N(I-1,J-1,K))        &
161           +Q23(I,J-1,K)*(W_N(I,J-1,K+1)-W_N(I,J-1,K-1)) )*DJAC(I,J-1,K)&
162           +(Q31(I,J,K+1)*(W_N(I+1,J,K+1)-W_N(I-1,J,K+1))        &
163           +Q32(I,J,K+1)*(W_N(I,J+1,K+1)-W_N(I,J-1,K+1)) )*DJAC(I,J,K+1)&
164           -(Q31(I,J,K-1)*(W_N(I+1,J,K-1)-W_N(I-1,J,K-1))        &
165           +Q32(I,J,K-1)*(W_N(I,J+1,K-1)-W_N(I,J-1,K-1)) )*DJAC(I,J,K-1))
166    273            ENDDO
167      2            ENDDO
168                   ENDDO
```

① Original 코드에서 비연속적인 메모리 access를 피하기 위해서 inner loop와 outer loop의 수행순서를 바꾸어 연속적인 메모리 access를 하도록 수정하였다.

② 최적화 코드에서는 동일한 영역에 대해서 반복되는 do-loop를 하나로 통합하면서 저장할 필요가 없는 array(U_cont, V_cont, U_cont_E, U_cont_W, V_cont_N, V_cont_S, W_cont_T, W_cont_B, DE, DW, DN, DS, DT, DB)에 대해서는 scalar 변수를 사용하거나 혹은 줄어든 dimension의 array로 선언하여 사용함으로써 store/load 수행 횟수를 줄였다.

③ 불필요한 대형 array를 가능한 사용하지 않아 전체적으로 메모리 요구량을 감소시킬 수 있다.

④ 최적화 코드에서 중간에 APS array를 구하는 부분이 추가되어 있는데 이는 calc_u, calc_v, calc_w, calc_p 루틴에서 AP array를 계산할 때 반복적으로 계산되어 지는 AN(I,J,K)+AS(I,J,K)+AE(I,J,K)+AW(I,J,K)+AT(I,J,K)+AB(I,J,K) 연산부분을 이곳에 미리 계산해 둔 것이다.

⑤ 마지막 부분에서는 동일한 index에 대해 반복되는 DJAC array를 괄호 밖으로 빼서 전체적으로 연산횟수를 줄이도록 했다.

6.7 calc_u 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
 21            3              DO I=SX1,EX1
 22            6              DO J=SY1,EY1
 23            9              DO K=SZ1,EZ1
 24
 25         2343                 SU(I,J,K)= 0.0D0
 26         5885                 SP(I,J,K)= -RE*DJAC(I,J,K)*2.0D0/DEL_T
 27
 28        48231                 RHS_X(I,J,K)=DJAC(I,J,K)*RE
                                   *(2.0D0*U_N(I,J,K)/DEL_T-23.0D0/6.0D0
                                   *NLX_N(I,J,K) +16.0D0/6.0D0
                                   *NLX_N1(I,J,K)-5.0D0/6.0D0*NLX_N2(I,J,K) &
 30                                +DIFX(I,J,K))
 31         2754                 SU0(I,J,K)= RHS_X(I,J,K)
 32
 33         4442              ENDDO
 34           72              ENDDO
 35                           ENDDO
 36
 37                           CALL BCU
 38
 39                           DO I=SX1,EX1
 40            1              DO J=SY1,EY1
 41           10              DO K=SZ1,EZ1
 42        54720             AP(I,J,K) =AN(I,J,K)+AS(I,J,K)+AE(I,J,K)
                             +AW(I,J,K)+AT(I,J,K)+AB(I,J,K)-SP(I,J,K)
 43           68              ENDDO
 44           59              ENDDO
 45            3              ENDDO
 46
 47                           DO I=SX1,EX1
 48           48              DO J=SY1,EY1
 49                           DO K=SZ1,EZ1
 50
 51        97872             SU(I,J,K)= SU0(I,J,K) +0.25D0      &
 52              *(Q12(I+1,J,K)*DJAC(I+1,J,K)*(U(I+1,J+1,K)-U(I+1,J-1,K))   &
 53              -Q12(I-1,J,K)*DJAC(I-1,J,K)*(U(I-1,J+1,K)-U(I-1,J-1,K))   &
 54              +Q13(I+1,J,K)*DJAC(I+1,J,K)*(U(I+1,J,K+1)-U(I+1,J,K-1))   &
 55              -Q13(I-1,J,K)*DJAC(I-1,J,K)*(U(I-1,J,K+1)-U(I-1,J,K-1))   &
 56              +Q21(I,J+1,K)*DJAC(I,J+1,K)*(U(I+1,J+1,K)-U(I-1,J+1,K))   &
 57              -Q21(I,J-1,K)*DJAC(I,J-1,K)*(U(I+1,J-1,K)-U(I-1,J-1,K))   &
 58              +Q23(I,J+1,K)*DJAC(I,J+1,K)*(U(I,J+1,K+1)-U(I,J+1,K-1))   &
 59              -Q23(I,J-1,K)*DJAC(I,J-1,K)*(U(I,J-1,K+1)-U(I,J-1,K-1))   &
```

```
   60                +Q31(I,J,K+1)*DJAC(I,J,K+1)*(U(I+1,J,K+1)-U(I-1,J,K+1))   &
   61                -Q31(I,J,K-1)*DJAC(I,J,K-1)*(U(I+1,J,K-1)-U(I-1,J,K-1))   &
   62                +Q32(I,J,K+1)*DJAC(I,J,K+1)*(U(I,J+1,K+1)-U(I,J-1,K+1))   &
   63                -Q32(I,J,K-1)*DJAC(I,J,K-1)*(U(I,J+1,K-1)-U(I,J-1,K-1)))
   64                           ENDDO
   65              7            ENDDO
   66                           ENDDO
   67
   68                           CALL BCU
```

<최적화 코드>

```
21
   22                           CALL BCU
   23
   24                           DO K=SZ1,EZ1
   25              4            DO J=SY1,EY1
   26                           DO I=SX1,EX1
   27
   28            168              SPs= -RE*DJAC(I,J,K)*2.0D0/DEL_T
   29
   30           4849             RHS_Xs=DJAC(I,J,K)*RE*(2.0D0*U_N(I,J,K)
                                   /DEL_T-23.0D0/6.0D0*NLX_N(I,J,K)
   31                           +16.0D0/6.0D0*NLX_N1(I,J,K)-5.0D0/6.0D0
                               *NLX_N2(I,J,K)
   32                           +DIFX(I,J,K))
   33                           SU0s= RHS_Xs
   34
   35              62           AP(I,J,K) =APS(I,J,K)-SPs
   36
   37           20832                SU(I,J,K)= SU0s +0.25D0
   38                             *((Q12(I+1,J,K)*(U(I+1,J+1,K)-U(I+1,J-1,K))   &
   39              +Q13(I+1,J,K)*(U(I+1,J,K+1)-U(I+1,J,K-1))*DJAC(I+1,J,K)   )&
   40              -(Q12(I-1,J,K)*(U(I-1,J+1,K)-U(I-1,J-1,K))   &
   41              +Q13(I-1,J,K)*(U(I-1,J,K+1)-U(I-1,J,K-1))*DJAC(I-1,J,K)   )&
   42              +(Q21(I,J+1,K)*(U(I+1,J+1,K)-U(I-1,J+1,K))   &
   43              +Q23(I,J+1,K)*(U(I,J+1,K+1)-U(I,J+1,K-1))*DJAC(I,J+1,K)   )&
   44              -(Q21(I,J-1,K)*(U(I+1,J-1,K)-U(I-1,J-1,K))   &
   45              +Q23(I,J-1,K)*(U(I,J-1,K+1)-U(I,J-1,K-1))*DJAC(I,J-1,K)   )&
   46              +(Q31(I,J,K+1)*(U(I+1,J,K+1)-U(I-1,J,K+1))   &
   47              +Q32(I,J,K+1)*(U(I,J+1,K+1)-U(I,J-1,K+1))*DJAC(I,J,K+1))&
   48              -(Q31(I,J,K-1)*(U(I+1,J,K-1)-U(I-1,J,K-1))   &
   49              +Q32(I,J,K-1)*(U(I,J+1,K-1)-U(I,J-1,K-1))*DJAC(I,J,K-1)) )
   50                           ENDDO
   51              80           ENDDO
   52                           ENDDO
```

① Original 코드에서 비연속적인 메모리 access를 피하기 위해서 inner loop와 outer loop의 수행순서를 바꾸어 연속적인 메모리 access를 하도록 수정하였다.

② 3개의 do-loop를 하나의 do-loop로 통합하였고, 그러면서 저장할 필요가 없는 SP, RHS_X, SU0 array를 사용하는 대신 scalar 변수를 사용함으로써 store/load 횟수를 감소시켰다.

③ 불필요한 대형 array를 가능한한 사용하지 않아 전체적으로 메모리 요구량을 감소시킬 수 있다.

④ AP를 계산할 때 수행하는 많은 연산을 coeff 루틴에서 미리 계산된 APS array를 이용하여 간소화하였다.

⑤ 동일한 index에 대해 반복되는 DJAC array를 괄호 밖으로 빼서 전체적으로 연산횟수를 줄이도록 했다.

⑥ Original 코드에서 마지막에 불필요하게 한번 더 bcu 루틴을 call하는 부분을 최적화 코드에서는 삭제하였다.

6.8 calc_v, calc_w 루틴

calc_u 루틴 변수이름 정도만 차이 날뿐 거의 같은 루틴이기 때문에 calc_u 루틴 최적화 방법과 동일한 방법으로 최적화하였다.

6.9 calc_p 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
 23          1          DO I=SX,EX
 24         20          DO J=SY,EY
 25        119          DO K=SZ,EZ
 26
 27     148846           U_CONT(I,J,K)=(U_S(I,J,K)
```

```
                                *XIX(I,J,K)+V_S(I,J,K)*XIY(I,J,K)
 28          172113              V_CONT(I,J,K)=(U_S(I,J,K)
                                *ETX(I,J,K)+V_S(I,J,K)*ETY(I,J,K)
 29          216433              W_CONT(I,J,K)=(U_S(I,J,K)
                                *ZTX(I,J,K)+V_S(I,J,K)*ZTY(I,J,K)
 30
 31                              ENDDO
 32              170             ENDDO
 33                              ENDDO
 34
 35                5             DO I=SX1,EX1
 36               21             DO J=SY1,EY1
 37               39             DO K=SZ1,EZ1
 38
 39          32075               U_CONT_E(I,J,K)=0.5D0
                                *(U_CONT(I,J,K)+U_CONT(I+1,J,K))
 40          25278               U_CONT_W(I,J,K)=0.5D0
                                *(U_CONT(I,J,K)+U_CONT(I-1,J,K))
 41          15855               V_CONT_N(I,J,K)=0.5D0
                                *(V_CONT(I,J,K)+V_CONT(I,J+1,K))
 42           1240               V_CONT_S(I,J,K)=0.5D0
                                *(V_CONT(I,J,K)+V_CONT(I,J-1,K))
 43          51286               W_CONT_T(I,J,K)=0.5D0
                                *(W_CONT(I,J,K)+W_CONT(I,J,K+1))
 44           1551               W_CONT_B(I,J,K)=0.5D0
                                *(W_CONT(I,J,K)+W_CONT(I,J,K-1))
 45
 46          15579               SP(I,J,K)=0.0D0
 47           6771               SU(I,J,K)=0.0D0
 48
 49          36464               SU0(I,J,K)=-1.0D0/DEL_T     &
 50                                        *((U_CONT_E(I,J,K)-U_CONT_W(I,J,K))&
 51                                        +(V_CONT_N(I,J,K)-V_CONT_S(I,J,K))   &
 52                                        +(w_CONT_t(I,J,K)-w_CONT_b(I,J,K))   )
 53                              ENDDO
 54              290             ENDDO
 55                              ENDDO
 56
 57                              CALL BCP
 58
 59                3             DO I=SX1,EX1
 60                9             DO J=SY1,EY1
 61               14             DO K=SZ1,EZ1
 62          152662              AP(I,J,K)=AN(I,J,K)+AS(I,J,K)+AE(I,J,K)
                                +AW(I,J,K)+AT(I,J,K)+AB(I,J,K)-SP(I,J,K)
 63              219             ENDDO
 64              157             ENDDO
 65                              ENDDO
 66
```

```
67                          CALL BCP
68
69              3           DO I=SX1,EX1
70            134           DO J=SY1,EY1
71                          DO K=SZ1,EZ1
72
73         280195             SU(I,J,K) =   SU0(I,J,K)+0.25D0      &
74                  *(Q12(I+1,J,K)*DJAC(I+1,J,K)*(P(I+1,J+1,K)-P(I+1,J-1,K))   &
75                  -Q12(I-1,J,K)*DJAC(I-1,J,K)*(P(I-1,J+1,K)-P(I-1,J-1,K))   &
76                  +Q13(I+1,J,K)*DJAC(I+1,J,K)*(P(I+1,J,K+1)-P(I+1,J,K-1))   &
77                  -Q13(I-1,J,K)*DJAC(I-1,J,K)*(P(I-1,J,K+1)-P(I-1,J,K-1))   &
78                  +Q21(I,J+1,K)*DJAC(I,J+1,K)*(P(I+1,J+1,K)-P(I-1,J+1,K))   &
79                  -Q21(I,J-1,K)*DJAC(I,J-1,K)*(P(I+1,J-1,K)-P(I-1,J-1,K))   &
80                  +Q23(I,J+1,K)*DJAC(I,J+1,K)*(P(I,J+1,K+1)-P(I,J+1,K-1))   &
81                  -Q23(I,J-1,K)*DJAC(I,J-1,K)*(P(I,J-1,K+1)-P(I,J-1,K-1))   &
82                  +Q31(I,J,K+1)*DJAC(I,J,K+1)*(P(I+1,J,K+1)-P(I-1,J,K+1))   &
83                  -Q31(I,J,K-1)*DJAC(I,J,K-1)*(P(I+1,J,K-1)-P(I-1,J,K-1))   &
84                  +Q32(I,J,K+1)*DJAC(I,J,K+1)*(P(I,J+1,K+1)-P(I,J-1,K+1))   &
85                  -Q32(I,J,K-1)*DJAC(I,J,K-1)*(P(I,J+1,K-1)-P(I,J-1,K-1))   )
86                          ENDDO
87              20          ENDDO
88                          ENDDO
```

< 최적화 코드 >

```
30                          CALL BCP
31
32              3           DO K=SZ,EZ
33             97           DO J=SY1,EY1
34             39           DO I=SX1,EX1
35          10628            W_CONTs(I,J,K)=(U_S(I,J,K)
                            *ZTX(I,J,K)+V_S(I,J,K)*ZTY(I,J,K)+
36                          ENDDO
37             19           ENDDO
38              1           ENDDO
39
40              2           DO K=SZ1,EZ1
41
42             84           DO J=SY,EY
43             10           DO I=SX1,EX1
44           7735            V_CONTs(I,J)=(U_S(I,J,K)
                            *ETX(I,J,K)+V_S(I,J,K)*ETY(I,J,K)+
45             13           ENDDO
46             18           ENDDO
47
48             58           DO J=SY1,EY1
49
```

```
50                          DO I=SX,EX
51          5599             U_CONTs(I)=(U_S(I,J,K)*XIX(I,J,K)
                               +V_S(I,J,K)*XIY(I,J,K)+
52                          ENDDO
53
54           154            DO I=SX1,EX1
55
56          1762            U_CONT_Es=U_CONTs(I)+U_CONTs(I+1)
57             6            U_CONT_Ws=U_CONTs(I)+U_CONTs(I-1)
58          2038            V_CONT_Ns=V_CONTs(I,J)+V_CONTs(I,J+1)
59           817            V_CONT_Ss=V_CONTs(I,J)+V_CONTs(I,J-1)
60           214            W_CONT_Ts=W_CONTs(I,J,K)+W_CONTs(I,J,K+1)
61           148            W_CONT_Bs=W_CONTs(I,J,K)+W_CONTs(I,J,K-1)
62
63          5464            SU0s=-0.5D0/DEL_T    &
64                               *((U_CONT_Es-U_CONT_Ws)   &
65                              +(V_CONT_Ns-V_CONT_Ss)   &
66                              +(w_CONT_ts-w_CONT_bs)   )
67           513            AP(I,J,K)=APS(I,J,K)
68
69         64758            SU(I,J,K) =   SU0s+0.25D0&
70                            *(   ( Q12(I+1,J,K)*(P(I+1,J+1,K)-P(I+1,J-1,K))&
71              +Q13(I+1,J,K)*(P(I+1,J,K+1)-P(I+1,J,K-1)) )*DJAC(I+1,J,K) &
72               -( Q12(I-1,J,K)*(P(I-1,J+1,K)-P(I-1,J-1,K))   &
73               +Q13(I-1,J,K)*(P(I-1,J,K+1)-P(I-1,J,K-1)) )*DJAC(I-1,J,K) &
74               +( Q21(I,J+1,K)*(P(I+1,J+1,K)-P(I-1,J+1,K))   &
75               +Q23(I,J+1,K)*(P(I,J+1,K+1)-P(I,J+1,K-1)) )*DJAC(I,J+1,K) &
76               -( Q21(I,J-1,K)*(P(I+1,J-1,K)-P(I-1,J-1,K))   &
77               +Q23(I,J-1,K)*(P(I,J-1,K+1)-P(I,J-1,K-1)) )*DJAC(I,J-1,K) &
78               +( Q31(I,J,K+1)*(P(I+1,J,K+1)-P(I-1,J,K+1))   &
79               +Q32(I,J,K+1)*(P(I,J+1,K+1)-P(I,J-1,K+1)) )*DJAC(I,J,K+1) &
80               -( Q31(I,J,K-1)*(P(I+1,J,K-1)-P(I-1,J,K-1))   &
81               +Q32(I,J,K-1)*(P(I,J+1,K-1)-P(I,J-1,K-1)) )*DJAC(I,J,K-1) )
82                          ENDDO
83            12            ENDDO
84             3            ENDDO
```

① Original 코드에서 비연속적인 메모리 access를 피하기 위해서 inner loop와 outer loop의 수행순서를 바꾸어 연속적인 메모리 access를 하도록 수정하였다.

② 최적화 코드에서는 동일한 영역에 대해서 반복되는 do-loop를 하나로 통합하면서 저장할 필요가 없는 array(U_cont, V_cont, U_cont_E, U_cont_W, V_cont_N, V_cont_S, W_cont_T, W_cont_B, SU0)에 대해서는 scalar 변수를 사용하거나 혹은 줄

어든 dimension의 array로 선언하여 사용함으로써 store/load 수행 횟수를 줄였다.

③ 불필요한 대형 array를 가능한한 사용하지 않아 전체적으로 메모리 요구량을 감소시킬 수 있다.

④ AP를 계산할 때 수행하는 많은 연산을 coeff 루틴에서 미리 계산된 APS array를 이용하여 간소화하였다.

⑤ 동일한 index에 대해 반복되는 DJAC array를 괄호 밖으로 빼서 전체적으로 연산횟수를 줄이도록 했다.

⑥ Original 코드에서 마지막에 불필요하게 한번 더 bcu 루틴을 call하는 부분을 최적화 코드에서는 삭제하였다.

6.10 tdma 루틴

A. 불필요한 계산 부분 삭제

<Original 코드>

```
45          100                TDMA_P(SX1:EX1,SY,SZ1:EZ1)=0.0D0
46          136                TDMA_Q(SX1:EX1,SY,SZ1:EZ1)
                                 =PHI(SX1:EX1,SY,SZ1:EZ1)
50            1              Do K=SZ1,EZ1
52       196088               PHI_OLD(:,:,:)=PHI(:,:,:)
54          110               DO J=SY1,EY1
55           22               DO I=SX1,EX1
56        38963               D(I,J,K) = Sterm(I,J,K)&
57                             +A_E(I,J,K)*PHI_OLD(I+1,J,K) &
58                             +A_W(I,J,K)*PHI_OLD(I-1,J,K) &
59                             +A_T(I,J,K)*PHI_OLD(I,J,K+1) &
60                             +A_B(I,J,K)*PHI_OLD(I,J,K-1)
62        11362               TDMA_P(I,J,K)=A_N(I,J,K)/(A_P(I,J,K)-A_S(I,J,K)
                                *TDMA_P(I,J-1,K))
63         1390               TDMA_Q(I,J,K)=(D(I,J,K)+A_S(I,J,K)
                                *TDMA_Q(I,J-1,K))/(A_P(I,J,K)
                                -A_S(I,J,K)*TDMA_P(I,J-1,K))
64                               ENDDO
65          144                  ENDDO
67           40                  DO J=EY1,SY1,-1
68            8                  DO I=EX1,SX1,-1
69         4226                       PHI(I,J,K)=TDMA_P(I,J,K)
                                     *PHI(I,J+1,K)+TDMA_Q(I,J,K)
```

```
 70          17                ENDDO
 71          19                ENDDO
 73           2             ENDDO
```

<최적화 코드>

```
 49           2             Do K=SZ1,EZ1
 51          16                TDMA_P(SX1:EX1,SY)=0.0D0
 52          30                TDMA_Q(SX1:EX1,SY)=PHI(SX1:EX1,SY,K)
 54          12                DO J=SY1,EY1
 55           5                DO I=SX1,EX1
 56       13988                 R=1./(A_P(I,J,K)-A_S(I,J,K)*TDMA_P(I,J-1))
 57       16213                 D = Sterm(I,J,K)                &
 58                                    +A_E(I,J,K)*PHI(I+1,J,K) &
 59                                    +A_W(I,J,K)*PHI(I-1,J,K) &
 60                                    +A_T(I,J,K)*PHI(I,J,K+1) &
 61                                    +A_B(I,J,K)*PHI(I,J,K-1)
 63       15980                 TDMA_P(I,J)=A_N(I,J,K)*R
 64         839                 TDMA_Q(I,J)=(D+A_S(I,J,K)*TDMA_Q(I,J-1))*R
 65                           ENDDO
 66          82                ENDDO
 68          16                DO J=EY1,SY1,-1
 69          25                DO I=SX1,EX1
 70        4376                      PHI(I,J,K)=TDMA_P(I,J)
                                       *PHI(I,J+1,K)+TDMA_Q(I,J)
 71           7                ENDDO
 72          16                ENDDO
 74                         ENDDO
```

① Original 코드에서 불필요한 연산 부분 (PHI_OLD(:,:,:)=PHI(:,:,:))을 삭제하였다.

② Original 코드에서 3차원 array로 선언한 후 K loop 바깥에서 초기화하여 사용하는 TDMA_P 및 TDMA_Q array를 최적화 코드에서는 2차원 array로 선언한 후 K loop 안쪽에서 초기화하여 사용하면서 전체적으로 3개의 K loop를 하나의 K loop로 통합한 결과를 얻게 된다.

③ 반복되는 연산(1./(A_P(I,J,K)-A_S(I,J,K)*TDMA_P(I,J-1)))을 최적화 코드에서는 미리 계산하여 계산부하가 높은 나눗셈을 조금이나마 피하도록 하였다.

7. Summary

7.1 최적화 결과

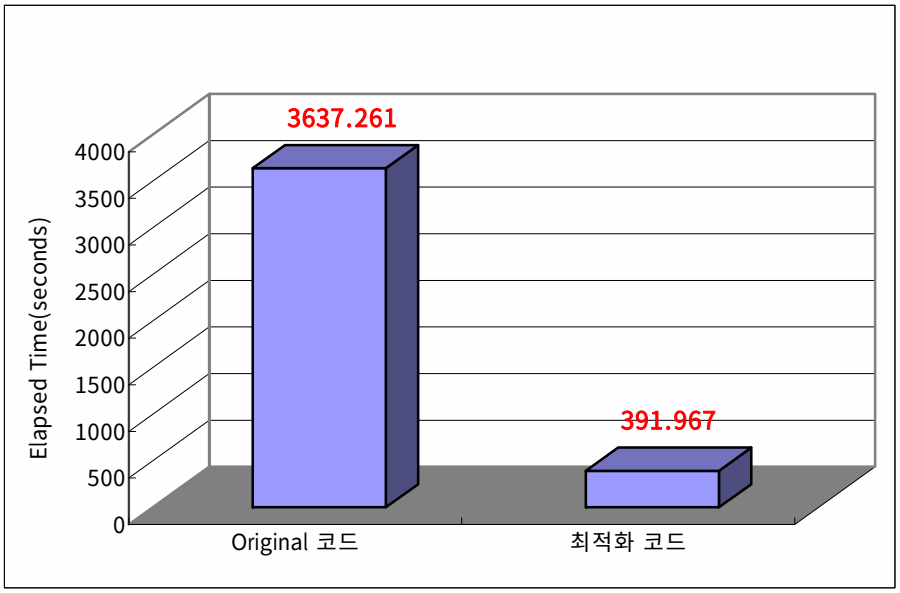
코드 전체적으로 봤을 때 Original 코드에서 잘못 사용한 Do-loop 순서를 수정하여 상당한 성능향상이 있었으며, exchange 부분의 MPI_Sendrecv 통신부분을 최적화하여 통신시간을 줄임으로써 전체적인 수행시간이 많이 단축되었다.

반복계산되어지는 부분 및 불필요한 계산부분을 삭제하였으며, Original 코드에서 해당 subroutine에서만 사용되던 큰 사이즈의 array를 많이 선언하여 사용했는데, 최적화 코드에서는 Do-loop를 Merging한 후 해당 array를 값에 대해 scalar 변수를 사용하면서 load/store 횟수를 줄임과 동시에 메모리가 낭비되는 것을 막았다.

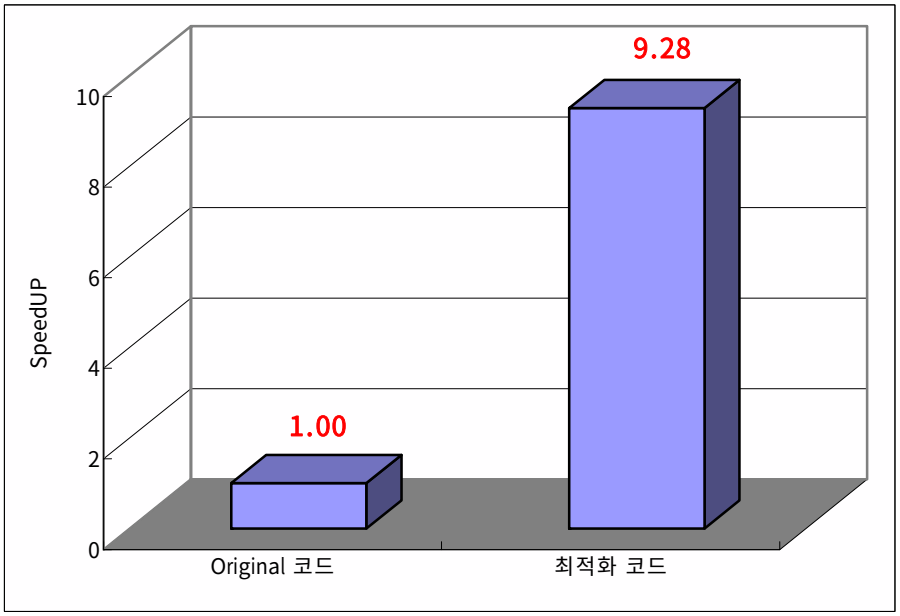
다음 결과는 MaxTIME=0.001, DEL_T=0.00001로 하여 전체적으로 100번의 Iteration을 하도록 했을 때의 성능이다.

< 표 II. 2 Original 코드와 최적화 코드의 수행시간 비교 및 rank별 평균 통신시간 비교 >

p690 1.7GHz 10CPU	전체 수행시간	Rank별 평균 통신 시간
Original 코드	3637.261	444.02
최적화 코드	391.967	20.313
Speedup	9.28	21.86



< 그림 II. 9 100IT에 대한 전체 수행 시간 비교 >



< 그림 II.10 100IT에 대한 최적화 코드의 Speedup >

8. Appendix

Mpitrace 결과 비교 – 10tasks 작업

<Original 코드>

(rank = 0)

```
-----------------------------------------------------------------
MPI Routine                        #calls        avg. bytes          time(sec)
-----------------------------------------------------------------
MPI_Comm_size                           1                0.0              0.000
MPI_Comm_rank                           2                0.0              0.002
MPI_Sendrecv                      2379600              576.0            302.640
MPI_Bcast                             101                8.0              0.015
MPI_Allreduce                        1280                4.3             13.892
-----------------------------------------------------------------
total communication time = 316.550 seconds.
total elapsed time       = 3637.261 seconds.
user cpu time            = 3630.350 seconds.
system time              = 2.500 seconds.
maximum memory size      = 480872 KBytes.

-----------------------------------------------------------------
Message size distributions:

MPI_Sendrecv                #calls        avg. bytes          time(sec)
                           1189800              448.0            190.770
                           1189800              704.0            111.870

MPI_Bcast                   #calls        avg. bytes          time(sec)
                              101                8.0              0.015

MPI_Allreduce               #calls        avg. bytes          time(sec)
                              1179                4.0             13.060
                               101                8.0              0.833
```

(rank = 4)

```
-----------------------------------------------------------------
MPI Routine                        #calls        avg. bytes          time(sec)
-----------------------------------------------------------------
MPI_Comm_size                           1                0.0              0.000
MPI_Comm_rank                           2                0.0              0.002
```

```
MPI_Sendrecv                2379600             560.0          430.532
MPI_Bcast                       101               8.0            0.023
MPI_Allreduce                  1280               4.3           17.378
-----------------------------------------------------------------
total communication time = 447.936 seconds.
total elapsed time       = 3637.261 seconds.
user cpu time             = 3628.800 seconds.
system time               = 2.700 seconds.
maximum memory size       = 464568 KBytes.

-----------------------------------------------------------------
Message size distributions:

MPI_Sendrecv              #calls       avg. bytes        time(sec)
                         1189800            448.0          321.142
                         1189800            672.0          109.390

MPI_Bcast                 #calls       avg. bytes        time(sec)
                             101            8.0            0.023

MPI_Allreduce             #calls       avg. bytes        time(sec)
                            1179            4.0           15.540
                             101            8.0            1.837
```

(rank = 9)

```
-----------------------------------------------------------------
MPI Routine                 #calls       avg. bytes        time(sec)
-----------------------------------------------------------------
MPI_Comm_size                    1              0.0            0.000
MPI_Comm_rank                    2              0.0            0.002
MPI_Sendrecv               2379600            560.0          493.000
MPI_Bcast                      101              8.0            0.026
MPI_Allreduce                 1280              4.3           17.974
-----------------------------------------------------------------
total communication time = 511.002 seconds.
total elapsed time       = 3637.261 seconds.
user cpu time             = 3627.290 seconds.
system time               = 3.330 seconds.
maximum memory size       = 464560 KBytes.

-----------------------------------------------------------------
Message size distributions:

MPI_Sendrecv              #calls       avg. bytes        time(sec)
                         1189800            448.0          357.152
                         1189800            672.0          135.848
```

```
MPI_Bcast                 #calls       avg. bytes        time(sec)
                            101              8.0             0.026

MPI_Allreduce             #calls       avg. bytes        time(sec)
                           1179              4.0            16.126
                            101              8.0             1.848
```

<최적화 코드>

(rank = 0)

```
-----------------------------------------------------------------
MPI Routine                     #calls       avg. bytes        time(sec)
-----------------------------------------------------------------
MPI_Comm_size                        1              0.0             0.000
MPI_Comm_rank                        2              0.0             0.000
MPI_Sendrecv                     18768          61160.0            10.571
MPI_Bcast                          101              8.0             0.017
MPI_Allreduce                     1254              4.3             0.490
-----------------------------------------------------------------
total communication time = 11.078 seconds.
total elapsed time       = 391.967 seconds.
user cpu time            = 387.230 seconds.
system time              = 2.720 seconds.
maximum memory size      = 397676 KBytes.

-----------------------------------------------------------------
Message size distributions:

MPI_Sendrecv              #calls       avg. bytes        time(sec)
                            9384         44880.0             6.110
                            9384         77440.0             4.461

MPI_Bcast                 #calls       avg. bytes        time(sec)
                            101              8.0             0.017

MPI_Allreduce             #calls       avg. bytes        time(sec)
                           1153              4.0             0.247
                            101              8.0             0.243
```

(rank = 4)

```
-----------------------------------------------------------------
MPI Routine                     #calls       avg. bytes        time(sec)
```

```
-----------------------------------------------------------------
MPI_Comm_size                1          0.0          0.000
MPI_Comm_rank                2          0.0          0.000
MPI_Sendrecv             18768      59400.0         19.150
MPI_Bcast                  101          8.0          0.041
MPI_Allreduce             1254          4.3          1.482
-----------------------------------------------------------------
total communication time = 20.673 seconds.
total elapsed time       = 391.967 seconds.
user cpu time            = 386.930 seconds.
system time              = 2.760 seconds.
maximum memory size      = 385184 KBytes.

-----------------------------------------------------------------
Message size distributions:

MPI_Sendrecv             #calls    avg. bytes      time(sec)
                           9384       44880.0         14.799
                           9384       73920.0          4.351

MPI_Bcast                #calls    avg. bytes      time(sec)
                            101          8.0          0.041

MPI_Allreduce            #calls    avg. bytes      time(sec)
                           1153          4.0          1.139
                            101          8.0          0.343
```

(rank = 9)

```
-----------------------------------------------------------------
MPI Routine                 #calls    avg. bytes      time(sec)
-----------------------------------------------------------------
MPI_Comm_size                1          0.0          0.000
MPI_Comm_rank                2          0.0          0.000
MPI_Sendrecv             18768      59400.0         21.660
MPI_Bcast                  101          8.0          0.037
MPI_Allreduce             1254          4.3          1.259
-----------------------------------------------------------------
total communication time = 22.956 seconds.
total elapsed time       = 391.967 seconds.
user cpu time            = 386.910 seconds.
system time              = 2.820 seconds.
maximum memory size      = 385216 KBytes.

-----------------------------------------------------------------
Message size distributions:
```

```
MPI_Sendrecv                #calls       avg. bytes       time(sec)
                              9384          44880.0          15.840
                              9384          73920.0           5.820

MPI_Bcast                   #calls       avg. bytes       time(sec)
                               101              8.0           0.037

MPI_Allreduce               #calls       avg. bytes       time(sec)
                              1153              4.0           0.973
                               101              8.0           0.286
```

CHAPTER III. KAIST PipeFlow 코드

1. Makefile

<Original inflow Makefile>

```
FC = xlf
FCFLAGS = -O3 -q64 -qrealsize=8 -qdpc=e -qarch=pwr4 -qtune=pwr4 -pg -g
FCFLAG = -q64 -pg -L/applic/vni/CTT4.0/lib/lib.rs6000_64 -limsl -limslsup ₩
         -limsls_err -limslmpistub
```

<최적화 inflow makefile>

```
FFLAGS= -O3 -pg -g -q64 -qnosave -qsmp=noauto -qreport=smplist -qrealsize=8
FFLAG = -q64 -pg -qsmp=noauto
FC= xlf_r
LIB=   -lessl -L/applic/vni/CTT4.0/lib/lib.rs6000_64 -limsl -limslsup ₩
       -limsls_err -limslmpistub
```

<Original Main Makefile>

```
FFLAGS= -O3 -q64 -qarch=pwr4 -qtune=pwr4 -qrealsize=8 -qdpc=e -pg -g
LDFLAG = -q64 -pg
LDFLAG_GRID = -q64
FC= xlf
LIB= -L/applic/vni/CTT4.0/lib/lib.rs6000_64 -limsl -limslsup -limsls_err -limslmpistub
```

<최적화 Main Makefile>

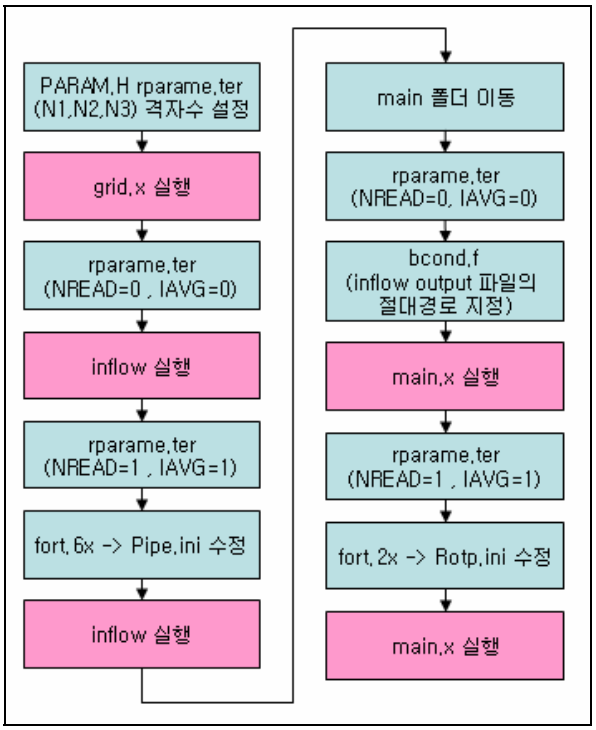
```
FFLAGS= -pg -g -O3 -q64 -qnosave -qsmp=noauto -qreport=smplist ₩
        -qarch=pwr4 -qtune=pwr4   -qrealsize=8 -qdpc=e -qsource -qlist
LDFLAG = -pg -q64 -qsmp=noauto
FC= xlf_r
LIB=   -lessl -L/applic/vni/CTT4.0/lib/lib.rs6000_64 -limsl -limslsup ₩
       -limsls_err -limslmpistub
```

2. Profiling 결과

격자를 생성하는 grid.x의 실행을 제외하면 최종 결과를 얻기까지 두 개의 실행파일(inflow, main.x)이 각 두 번씩 실행이 된다. 그리고 각 실행파일이 두 번 실행되는 동안 호출하는 서브루틴에 약간의 차이가 있다. 이런 이유로 두 개의 실행파일이 두 번 실행되는 것에 대해 각각의 profile결과를 얻었다. 최적화된 순차코드가 따로 없어서, OpenMP 병렬코드를 하나의 thread를 이용해 실행한 profiling 결과를 최적화 순차코드의 profiling 결과로 대신 하였다.

이 코드는 이미 한 차례 최적화 과정을 거쳤던 것으로 original 코드와 비교하여 최적화 코드의 성능 증가 정도가 그렇게 높지는 않다.

아래 그림은 전체 코드의 실행 순서를 나타낸 것이다.



<그림 III.1 전체 코드 실행 history >

2.1 inflow 실행파일의 profiling 결과 (실행 1)

<Original 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 300.68 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
17.6	52.83	52.83	100	528.30	672.39	.getuh3 [5]
13.0	91.85	39.02	100	390.20	390.20	.rhs3 [6]
11.6	126.68	34.83	100	348.30	348.30	.rhs1 [7]
9.3	154.62	27.94	100	279.40	279.40	.rhs2 [10]
6.7	174.86	20.24	38400	0.53	0.53	.ctdma1j [11]
5.5	191.41	16.55	28700	0.58	0.58	.ctdma3i [12]
4.9	206.25	14.84	100	148.40	291.92	.getuh2 [8]
4.8	220.61	14.36	100	143.60	287.69	.getuh1 [9]
3.1	229.92	9.31				.memmove [15]
2.7	238.00	8.08	100	80.80	130.10	.getdp [14]
2.1	244.38	6.38	38400	0.17	0.17	.tdmai [16]
2.0	250.30	5.92	100	59.20	59.20	.upcalc [17]
1.6	255.23	4.93	1664000	0.00	0.00	.tdmap [19]
1.3	259.26	4.03				.df6tcb [21]
1.2	262.82	3.56				.df6tcf [23]
1.1	266.25	3.43	100	34.30	2515.10	.getup [3]
1.1	269.62	3.37	200	16.85	16.85	.cfl [24]
0.9	272.44	2.82				.FormatControl [25]
0.7	274.53	2.09	100	20.90	20.90	.rhsdp [26]
0.6	276.42	1.89	101	18.71	18.71	.divcheck [27]
0.6	278.27	1.85	10666789	0.00	0.00	.cvtloop [28]
0.6	280.05	1.78				.df2t2d [29]
0.6	281.75	1.70				.WriteUnit [30]
0.5	283.39	1.64				.__mcount [31]
0.5	285.02	1.63				.df2t2b [32]
0.5	286.61	1.59				.df3tcb [33]
0.5	288.19	1.58	11167476	0.00	0.00	.__cvt_r [22]
0.4	289.46	1.27				.FmtRToQED [18]
0.4	290.72	1.26				.df3tcf [34]
0.4	291.93	1.21	10	121.00	124.22	.insfield [35]
0.4	293.03	1.10				.IOWrite [36]
0.3	293.89	0.86				._xlfWriteUfmt [37]
0.3	294.73	0.84				._xlfWriteFmt [38]
0.2	295.39	0.66	100	6.60	6.60	.nutcheck [39]
0.2	295.86	0.47				.IOFill [40]

<최적화 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 265.16 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
15.3	40.66	40.66	100	406.60	406.60	.rhs3@OL@1 [2]
12.0	72.41	31.75	100	317.50	317.50	.rhs1@OL@1 [3]
9.6	97.81	25.40	100	254.00	254.00	.rhs2@OL@1 [4]
7.5	117.63	19.82	38400	0.52	0.52	.ctdma1j1 [6]
7.2	136.64	19.01	100	190.10	212.07	.getuh3@OL@1 [5]
5.9	152.31	15.67	28700	0.55	0.55	.ctdma3i1 [7]
4.4	163.89	11.58	100	115.80	115.80	.getuh3@OL@5 [8]
3.2	172.46	8.57				.memmove [12]
3.1	180.74	8.28	100	82.80	104.77	.getuh1@OL@1 [9]
3.0	188.68	7.94	100	79.40	79.40	.getuh3@OL@4 [17]
2.5	195.27	6.59	38400	0.17	0.17	.tdmai1 [19]
2.2	201.15	5.88	100	58.80	80.77	.getuh2@OL@1 [16]
1.3	204.54	3.39	100	33.90	33.90	.upcalc@OL@2 [21]
1.2	207.80	3.26	100	32.60	85.02	.getuh3@OL@2 [14]
1.2	210.95	3.15	100	31.50	83.37	.getuh2@OL@3 [15]
1.2	214.06	3.11	200	15.55	15.55	.cfl@OL@1 [22]
1.1	217.09	3.03				.FormatControl [23]
1.0	219.72	2.63	100	26.30	26.30	.getup@OL@1 [26]
1.0	222.34	2.62	100	26.20	92.27	.getuh2@OL@2 [10]
0.9	224.70	2.36	100	23.60	89.67	.getuh3@OL@3 [11]
0.9	227.05	2.35	100	23.50	75.92	.getuh1@OL@3 [18]
0.9	229.36	2.31	6300	0.37	0.37	.tdmap2 [30]
0.7	231.26	1.90	100	19.00	85.07	.getuh1@OL@2 [13]
0.7	233.11	1.85	100	18.50	18.50	.rhsdp@OL@1 [31]
0.7	234.93	1.82				.__mcount [32]
0.7	236.73	1.80	100	18.00	18.00	.getuh3@OL@6 [33]
0.7	238.47	1.74				.d8lw4 [34]
0.6	240.17	1.70				.WriteUnit [35]
0.6	241.72	1.55	11167091	0.00	0.00	.__cvt_r [25]
0.6	243.18	1.46				.__mcount [36]
0.5	244.58	1.40	10	140.00	140.00	.insfield [37]
0.5	245.97	1.39	100	13.90	13.90	.upcalc@OL@1 [38]
0.5	247.31	1.34				.IOWrite [39]
0.5	248.61	1.30	101	12.87	12.87	.divcheck@OL@1 [40]
0.5	249.86	1.25				.d16f2$ [41]
0.4	251.03	1.17	100	11.70	11.70	.getdp@OL@3 [42]
0.4	252.09	1.06				.FmtRToQED [20]
0.4	253.14	1.05	100	10.50	10.50	.getdp@OL@1 [43]
0.4	254.16	1.02	6027747	0.00	0.00	.cvtloop [44]

0.3	255.01	0.85				._xlfWriteUfmt [45]
0.3	255.80	0.79				._xlfWriteFmt [46]
0.3	256.47	0.67	100	6.70	6.70	.nutcheck [49]

<OpenMP 병렬코드(32 threads) profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 1506.95 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
76.2	1148.52	1148.52				.ThdCode [1]
2.9	1192.19	43.67	651	67.08	67.08	.rhs3@OL@1 [3]
2.3	1227.53	35.34	757	46.68	46.68	.rhs1@OL@1 [4]
2.2	1261.10	33.57	27288	1.23	1.23	.ctdma3i1 [5]
1.8	1288.45	27.35	681	40.16	40.16	.rhs2@OL@1 [6]
1.6	1312.19	23.74	35409	0.67	0.67	.ctdma1j1 [8]
1.4	1333.68	21.49	692	31.05	35.67	.getuh3@OL@1 [7]
0.8	1346.39	12.71	629	20.21	20.21	.getuh3@OL@5 [13]
0.7	1356.39	10.00	652	15.34	15.34	.getuh3@OL@4 [18]
0.6	1365.85	9.46	646	14.64	14.64	.getup@OL@1 [19]
0.6	1375.30	9.45				.memmove [20]
0.6	1384.55	9.25	661	13.99	18.43	.getuh1@OL@1 [14]
0.6	1393.74	9.19	36167	0.25	0.25	.tdmai1 [21]
0.6	1402.03	8.29	624	13.29	13.29	.upcalc@OL@2 [22]
0.5	1409.98	7.95	616	12.91	31.27	.getuh3@OL@2 [9]
0.5	1417.72	7.74	612	12.65	30.79	.getuh2@OL@3 [10]
0.5	1425.06	7.34	581	12.63	17.91	.getuh2@OL@1 [17]
0.4	1431.25	6.19	614	10.08	28.25	.getuh1@OL@3 [11]
0.4	1436.56	5.31	625	8.50	8.50	.rhsdp@OL@1 [23]
0.3	1441.53	4.97	1205	4.12	4.12	.cfl@OL@1 [24]
0.3	1446.39	4.86	603	8.06	21.39	.getuh3@OL@3 [12]
0.3	1451.16	4.77	560	8.52	8.52	.getuh3@OL@6 [25]
0.2	1454.64	3.48	849	4.10	4.10	.divcheck@OL@1 [27]
0.2	1457.96	3.32	591	5.62	5.62	.getdp@OL@1 [28]
0.2	1461.19	3.23	611	5.29	5.29	.getdp@OL@3 [29]
0.2	1464.39	3.20	644	4.97	16.77	.getuh2@OL@2 [16]
0.2	1467.57	3.18				.d8lw4 [30]
0.2	1470.35	2.78				.FormatControl [31]
0.2	1473.12	2.77	548	5.05	19.84	.getuh1@OL@2 [15]
0.2	1475.87	2.75	1146	2.40	2.40	.upcalc@OL@1 [34]
0.2	1478.39	2.52				.d16f2$ [37]
0.2	1480.66	2.27	5346	0.42	0.42	.tdmap2 [39]

2.2 inflow 실행파일의 profiling 결과 (실행 2)

실행 1에서 사용되지 않았던 energy 루틴이 실행 2에서 호출되고 있으며 이것이 profiling 결과에서 최상단을 차지하고 있다.

<Original 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 365.77 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
16.9	61.73	61.73	100	617.30	617.30	.energy [6]
13.4	110.58	48.85	100	488.50	626.85	.getuh3 [5]
10.2	148.00	37.42	100	374.20	374.20	.rhs3 [7]
8.9	180.62	32.62	100	326.20	326.20	.rhs1 [8]
7.2	206.84	26.22	100	262.20	262.20	.rhs2 [11]
5.3	226.09	19.25	38400	0.50	0.50	.ctdma1j [13]
4.6	243.05	16.96	9600	1.77	1.77	.strain [14]
4.4	259.04	15.99	28700	0.56	0.56	.ctdma3i [15]
3.9	273.24	14.20	100	142.00	280.35	.getuh1 [9]
3.8	286.99	13.75	100	137.50	275.30	.getuh2 [10]
2.6	296.53	9.54				.memmove [18]
2.0	303.93	7.40	100	74.00	120.30	.getdp [17]
1.7	310.14	6.21	38400	0.16	0.16	.tdmai [19]
1.3	315.06	4.92	100	49.20	49.20	.upcalc [21]
1.3	319.69	4.63	1664000	0.00	0.00	.tdmap [22]
1.0	323.53	3.84				.df6tcb [24]
1.0	327.06	3.53				.df6tcf [26]
0.9	330.29	3.23	200	16.15	16.15	.cfl [27]
0.8	333.27	2.98	100	29.80	2365.30	.getup [3]
0.7	335.95	2.68				.FormatControl [28]
0.6	338.26	2.31	100	23.10	192.70	.sgstress [12]
0.6	340.36	2.10				.__mcount [29]
0.6	342.40	2.04	100	20.40	20.40	.rhsdp [30]
0.5	344.23	1.83	11167476	0.00	0.00	.__cvt_r [25]
0.5	345.99	1.76	101	17.43	17.43	.divcheck [31]
0.5	347.72	1.73				.df2t2d [32]
0.4	349.36	1.64				.df2t2b [33]
0.4	350.87	1.51				.WriteUnit [34]
0.4	352.35	1.48	10666778	0.00	0.00	.cvtloop [35]
0.4	353.78	1.43				.df3tcb [36]
0.4	355.21	1.43				.df3tcf [37]
0.3	356.43	1.22	10	122.00	122.80	.insfield [38]

0.3	357.65	1.22				.IOWrite [39]
0.3	358.84	1.19				.FmtRToQED [20]
0.3	359.76	0.92				._xlfWriteUfmt [40]
0.2	360.51	0.75				._xlfWriteFmt [41]
0.2	361.15	0.64	100	6.40	6.40	.nutcheck [42]

<최적화 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 341.43 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
16.3	55.82	55.82	100	558.20	558.20	.energy@OL@1 [2]
12.0	96.66	40.84	100	408.40	408.40	.rhs3@OL@1 [3]
9.2	128.00	31.34	100	313.40	313.40	.rhs1@OL@1 [4]
7.4	153.39	25.39	100	253.90	253.90	.rhs2@OL@1 [5]
5.8	173.08	19.69	38400	0.51	0.51	.ctdma1j1 [7]
5.5	191.76	18.68	100	186.80	209.77	.getuh3@OL@1 [6]
4.8	208.11	16.35	9600	1.70	1.70	.strain [9]
4.7	224.10	15.99	28700	0.56	0.56	.ctdma3i1 [10]
3.4	235.59	11.49	100	114.90	114.90	.getuh3@OL@5 [11]
2.5	244.29	8.70				.memmove [15]
2.5	252.73	8.44	100	84.40	107.37	.getuh1@OL@1 [12]
2.3	260.70	7.97	100	79.70	79.70	.getuh3@OL@4 [20]
2.0	267.59	6.89	38400	0.18	0.18	.tdmai1 [22]
1.7	273.35	5.76	100	57.60	80.57	.getuh2@OL@1 [19]
1.0	276.65	3.30	100	33.00	33.00	.upcalc@OL@2 [24]
0.9	279.88	3.23	100	32.30	85.79	.getuh3@OL@2 [17]
0.9	282.94	3.06	100	30.60	194.10	.sgstress@OL@1 [8]
0.9	285.99	3.05	100	30.50	83.43	.getuh2@OL@3 [18]
0.9	289.03	3.04	200	15.20	15.20	.cfl@OL@1 [25]
0.8	291.89	2.86	100	28.60	28.60	.getup@OL@1 [26]
0.8	294.70	2.81				.FormatControl [28]
0.8	297.50	2.80	100	28.00	93.63	.getuh2@OL@2 [13]
0.7	299.79	2.29	100	22.90	76.39	.getuh1@OL@3 [21]
0.7	302.02	2.23	100	22.30	87.93	.getuh3@OL@3 [14]
0.6	304.17	2.15	6300	0.34	0.34	.tdmap2 [33]
0.6	306.19	2.02	100	20.20	85.83	.getuh1@OL@2 [16]
0.6	308.18	1.99				.__mcount [34]
0.6	310.08	1.90	100	19.00	19.00	.rhsdp@OL@1 [35]
0.5	311.89	1.81				.d8lw4 [36]
0.5	313.67	1.78	100	17.80	17.80	.getuh3@OL@6 [37]
0.5	315.35	1.68				.WriteUnit [38]
0.4	316.76	1.41	100	14.10	14.10	.upcalc@OL@1 [39]
0.4	318.16	1.40	101	13.86	13.86	.divcheck@OL@1 [40]

0.4	319.56	1.40				.IOWrite [41]
0.4	320.95	1.39				.__mcount [42]
0.4	322.29	1.34	11167091	0.00	0.00	.__cvt_r [29]
0.4	323.60	1.31	10	131.00	131.00	.insfield [43]
0.4	324.84	1.24	100	12.40	12.40	.getdp@OL@3 [44]
0.3	326.02	1.18				.d16f2$ [45]
0.3	327.17	1.15				._xlfWriteUfmt [46]
0.3	328.26	1.09	6027747	0.00	0.00	.cvtloop [47]
0.3	329.32	1.06	100	10.60	10.60	.getdp@OL@1 [48]
0.3	330.30	0.98				.local_unlock_ppc_mp [49]
0.3	331.22	0.92				.FmtRToQED [23]
0.2	331.96	0.74				._xlfWriteFmt [50]
0.2	332.69	0.73				.d8f2$ [51]
0.2	333.34	0.65	100	6.50	6.50	.nutcheck [53]
0.2	333.91	0.57				.IOFill [55]
0.2	334.47	0.56				.drcb$ [56]

<OpenMP 병렬코드(32 threads) profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 1636.86 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
72.4	1185.46	1185.46				.ThdCode [1]
4.4	1257.51	72.05	736	97.89	97.89	.energy@OL@1 [3]
2.7	1301.08	43.57	591	73.72	73.72	.rhs3@OL@1 [4]
2.1	1335.67	34.59	606	57.08	57.08	.rhs1@OL@1 [5]
2.1	1369.32	33.65	27388	1.23	1.23	.ctdma3i1 [6]
1.7	1396.63	27.31	603	45.29	45.29	.rhs2@OL@1 [7]
1.5	1420.51	23.88	35486	0.67	0.67	.ctdma1j1 [9]
1.3	1442.00	21.49	624	34.44	39.47	.getuh3@OL@1 [8]
1.2	1461.35	19.35	6923	2.80	2.80	.strain [11]
0.8	1474.05	12.70	575	22.09	22.09	.getuh3@OL@5 [16]
0.6	1484.07	10.02	619	16.19	16.19	.getuh3@OL@4 [21]
0.6	1493.80	9.73				.memmove [22]
0.6	1503.29	9.49	582	16.31	16.31	.getup@OL@1 [23]
0.6	1512.55	9.26	587	15.78	20.64	.getuh1@OL@1 [17]
0.5	1521.53	8.98	35861	0.25	0.25	.tdmai1 [24]
0.5	1529.97	8.44	596	14.16	14.16	.upcalc@OL@2 [25]
0.4	1537.33	7.36	595	12.37	17.39	.getuh2@OL@1 [20]
0.4	1544.61	7.28	627	11.61	29.41	.getuh2@OL@3 [12]
0.4	1551.73	7.12	629	11.32	29.22	.getuh3@OL@2 [13]
0.3	1557.27	5.54	653	8.48	25.68	.getuh1@OL@3 [14]
0.3	1562.56	5.29	534	9.91	9.91	.rhsdp@OL@1 [26]
0.3	1567.31	4.75	542	8.76	8.76	.getuh3@OL@6 [27]

0.3	1571.92	4.61	606	7.61	20.97	.getuh3@OL@3 [15]
0.3	1576.24	4.32	1194	3.62	3.62	.cfl@OL@1 [28]
0.3	1580.45	4.21	704	5.98	33.47	.sgstress@OL@1 [10]
0.2	1583.92	3.47	856	4.05	4.05	.divcheck@OL@1 [30]
0.2	1587.31	3.39	644	5.26	5.26	.getdp@OL@3 [31]
0.2	1590.69	3.38	626	5.40	5.40	.getdp@OL@1 [32]
0.2	1593.83	3.14				.d8lw4 [33]
0.2	1596.93	3.10				.FormatControl [34]
0.2	1599.96	3.03	586	5.17	18.36	.getuh2@OL@2 [19]
0.2	1602.81	2.85	1171	2.43	2.43	.upcalc@OL@1 [35]
0.2	1605.64	2.83	625	4.53	17.41	.getuh1@OL@2 [18]
0.2	1608.21	2.57				.d16f2$ [40]

2.3 main.x 실행파일의 profiling 결과 (실행 1)

<Original 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 7416.37 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
15.3	1138.08	1138.08	1037428	1.10	1.20	.tribi [10]
12.9	2094.92	956.84	518714	1.84	4.63	.itsolv [8]
11.6	2952.38	857.46	100	8574.60	9429.59	.getuh3 [12]
7.3	3494.99	542.61	100	5426.10	5426.10	.rhs3 [13]
6.3	3963.73	468.74	320321	1.46	1.52	.resca [14]
6.0	4407.83	444.10	100	4441.00	4441.00	.rhs1 [15]
6.0	4851.76	443.93	100	4439.30	4439.30	.rhs2 [16]
4.8	5206.72	354.96	100	3549.60	4402.92	.getuh2 [17]
4.5	5539.56	332.84	149989	2.22	2.27	.resma [18]
3.2	5776.92	237.36	100	2373.60	3228.59	.getuh1 [20]
2.4	5951.41	174.49	1186062533	0.00	0.00	._pow [19]
1.7	6078.94	127.53	137349	0.93	0.93	.tfila [21]
1.6	6196.49	117.55				.memcpy [22]
1.5	6304.58	108.09				.__mcount [23]
1.5	6412.38	107.80	137349	0.78	0.79	.tlafi [24]
1.3	6508.17	95.79	57500	1.67	1.67	.ctdma3i [25]
1.2	6597.87	89.70	38400	2.34	2.34	.tdmaj [26]
1.2	6684.68	86.81	1186007685	0.00	0.00	.expinner2 [27]
1.0	6755.52	70.84	38400	1.84	1.84	.tdmai [28]
0.9	6823.94	68.42	1186007685	0.00	0.00	.loginner2 [29]
0.9	6889.47	65.53	100	655.30	655.30	.upcalc [30]

0.8	6948.88	59.41	100	594.10	35444.97	.takedp [5]
0.8	7006.43	57.55	100	575.50	575.50	.rhsdp [31]
0.7	7060.99	54.56	200	272.80	272.80	.cfl [32]
0.6	7107.37	46.38	100	463.80	68814.37	.getup [3]
0.4	7138.30	30.93	101	306.24	306.24	.divcheck [34]
0.4	7166.68	28.38	100	283.80	31651.30	.uhcalc [7]
0.3	7190.24	23.56	182972	0.13	11.58	.cfila [9]
0.3	7211.72	21.48	10	2148.00	2649.78	.insfield [35]
0.3	7232.97	21.25				.FormatControl [38]
0.2	7245.77	12.80				.dcopy [39]
0.2	7257.98	12.21				.WriteUnit [40]
0.2	7269.61	11.63	62354129	0.00	0.00	.cvtloop [41]
0.2	7281.05	11.44				.qincrement [42]

<최적화 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 3965.10 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
13.9	549.77	549.77	100	5497.70	5497.70	.rhs3@OL@1 [3]
11.7	1014.33	464.56	100	4645.60	4645.60	.rhs2@OL@1 [4]
11.4	1465.34	451.01	100	4510.10	4510.10	.rhs1@OL@1 [5]
6.4	1719.18	253.84	100	2538.40	2716.10	.getuh3@OL@1 [11]
6.1	1961.98	242.80	100	2428.00	2907.63	.getuh2@OL@1 [8]
5.4	2175.83	213.85	145081	1.47	2.41	.itsolv [6]
4.2	2343.56	167.73	100	1677.30	1677.30	.getuh3@OL@5 [12]
3.4	2479.00	135.44	290162	0.47	0.47	.tribi [15]
3.1	2600.27	121.27				._moveeq [16]
2.9	2717.18	116.91	100	1169.10	1169.10	.getuh3@OL@4 [17]
2.6	2821.40	104.22	57500	1.81	1.81	.ctdma3i [18]
2.5	2919.89	98.49	100	984.90	1464.53	.getuh1@OL@1 [13]
2.3	3011.66	91.77	88778	1.03	1.03	.resca [20]
2.3	3102.24	90.58	38400	2.36	2.36	.tdmaj [21]
1.9	3177.61	75.37	49606	1.52	1.52	.resma [22]
1.8	3250.90	73.29	100	732.90	1034.83	.getuh3@OL@3 [19]
1.7	3318.33	67.43	100	674.30	674.30	.upcalc@OL@1 [23]
1.5	3377.70	59.37	200	296.85	296.85	.cfl@OL@1 [27]
1.4	3434.36	56.66	100	566.60	566.60	.rhsdp@OL@1 [28]
1.3	3487.67	53.31	38400	1.39	1.39	.tdmai [29]
0.8	3519.98	32.31	100	323.10	323.10	.getup@OL@1 [33]
0.8	3552.21	32.23	100	322.30	322.30	.uhcalc@OL@1 [34]
0.7	3581.60	29.39	101	290.99	290.99	.divcheck@OL@1 [35]
0.7	3609.81	28.21	100	282.10	630.10	.getuh1@OL@2 [24]
0.7	3636.82	27.01	100	270.10	616.29	.getuh2@OL@2 [25]

0.6	3662.57	25.75	100	257.50	605.50	.getuh3@OL@2 [26]
0.6	3687.02	24.45	38979	0.63	0.63	.tfila [38]
0.6	3710.52	23.50	10	2350.00	2350.00	.insfield [39]
0.5	3729.11	18.59				.FormatControl [40]
0.4	3745.45	16.34				.__mcount [41]
0.3	3757.18	11.73	62317366	0.00	0.00	.cvtloop [42]
0.3	3768.13	10.95	100	109.50	109.50	.takedp@OL@1 [43]
0.3	3778.90	10.77				.WriteUnit [44]
0.3	3789.56	10.66	10	1066.00	1066.00	.profile [45]
0.3	3800.13	10.57	65108196	0.00	0.00	.__cvt_r [37]
0.3	3810.17	10.04				.__mcount [46]
0.2	3819.87	9.70				.IOWrite [47]
0.2	3829.32	9.45	100	94.50	94.50	.takedp@OL@4 [48]
0.2	3837.75	8.43	100	84.30	84.30	.nutcheck@OL@1 [49]
0.2	3846.15	8.40	100	84.00	2817.66	.takedp@OL@3 [10]
0.2	3854.49	8.34	100	83.40	2903.84	.takedp@OL@2 [9]
0.2	3862.35	7.86	38979	0.20	0.20	.tlafi [50]
0.2	3870.02	7.67				._xlfWriteUfmt [51]
0.2	3876.91	6.89				.local_unlock_ppc_mp [52]
0.2	3883.73	6.82				.FmtRToQED [32]
0.2	3890.40	6.67	49799	0.13	6.76	.cfila [7]

<OpenMP 병렬코드(32 threads) profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 7618.52 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
34.4	2621.63	2621.63				.ThdCode [2]
7.7	3209.37	587.74	3298	178.21	178.21	.rhs3@OL@1 [4]
6.5	3707.59	498.22	2729	182.57	182.57	.rhs2@OL@1 [6]
6.5	4201.43	493.84	2959	166.89	166.89	.rhs1@OL@1 [7]
4.8	4566.40	364.97	111287	3.28	4.84	.itsolv [5]
3.8	4855.55	289.15	9915	29.16	32.36	.getuh3@OL@1 [12]
3.5	5119.58	264.03	54696	4.83	4.83	.ctdma3i [13]
3.3	5369.66	250.08	9754	25.64	33.83	.getuh2@OL@1 [11]
2.3	5543.11	173.45	218582	0.79	0.79	.tribi [16]
2.1	5705.79	162.68	7413	21.95	21.95	.getuh3@OL@5 [17]
1.9	5847.34	141.55	9678	14.63	25.94	.getuh1@OL@1 [14]
1.9	5988.63	141.29	13375	10.56	10.56	.tdmai [19]
1.8	6125.74	137.11	6307	21.74	21.74	.getuh3@OL@4 [21]
1.8	6260.18	134.44	36446	3.69	3.69	.resma [22]
1.7	6391.21	131.03	69600	1.88	1.88	.resca [24]
1.5	6505.22	114.01	10025	11.37	11.37	.upcalc@OL@1 [25]
1.4	6614.58	109.36				._moveeq [26]

1.3	6712.18	97.60	13568	7.19	7.19	.tdmaj [28]
1.1	6796.47	84.29	9762	8.63	10.46	.getuh3@OL@3 [27]
0.8	6858.34	61.87	10191	6.07	6.07	.getup@OL@1 [29]
0.8	6918.88	60.54	3899	15.53	15.53	.rhsdp@OL@1 [30]
0.8	6977.24	58.36	1211	48.19	48.19	.cfl@OL@1 [31]
0.7	7032.60	55.36	16432	3.37	8.81	.getuh3@OL@2 [18]
0.7	7087.75	55.15	16256	3.39	8.52	.getuh2@OL@2 [20]
0.6	7136.23	48.48	10095	4.80	4.80	.uhcalc@OL@1 [32]
0.6	7179.27	43.04	16439	2.62	8.18	.getuh1@OL@2 [23]
0.4	7213.13	33.86	33566	1.01	1.01	.tfila [35]
0.4	7244.81	31.68	16345	1.94	1.94	.takedp@OL@1 [37]
0.4	7275.86	31.05	657	47.26	47.26	.divcheck@OL@1 [38]
0.3	7299.75	23.89	3782	6.32	121.31	.takedp@OL@2 [9]
0.3	7323.46	23.71	16361	1.45	1.45	.takedp@OL@4 [41]
0.3	7346.01	22.55	3641	6.19	122.07	.takedp@OL@3 [10]
0.3	7367.36	21.35	10	2135.00	2135.00	.insfield [42]
0.3	7386.85	19.49				.FormatControl [43]
0.2	7401.78	14.93				.__mcount [44]
0.2	7413.82	12.04				.WriteUnit [45]
0.2	7425.54	11.72	62316810	0.00	0.00	.cvtloop [46]

2.4 main.x 실행파일의 profiling 결과 (실행 2)

Inflow에서와 마찬가지로 실행 1에서 사용되지 않았던 energy 루틴이 실행 2에서 호출되고 있으며 이것이 profiling 결과에서 최상단을 차지하고 있다.

<Original 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 9294.91 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
18.1	1684.58	1684.58	100	16845.80	16845.80	.energy [10]
11.8	2780.52	1095.94	1013574	1.08	1.19	.tribi [11]
10.0	3708.45	927.93	506787	1.83	4.58	.itsolv [8]
9.0	4545.71	837.26	100	8372.60	9230.23	.getuh3 [13]
5.8	5084.36	538.65	100	5386.50	5386.50	.rhs3 [14]
4.8	5535.15	450.79	311384	1.45	1.50	.resca [15]
4.7	5975.36	440.21	100	4402.10	4402.10	.rhs2 [16]
4.7	6415.40	440.04	100	4400.40	4400.40	.rhs1 [17]
3.8	6765.79	350.39	100	3503.90	4359.85	.getuh2 [18]

3.5	7088.16	322.37	143812	2.24	2.30	.resma [19]
2.5	7323.91	235.75	100	2357.50	3215.13	.getuh1 [22]
2.1	7515.56	191.65	19200	9.98	9.98	.strain [23]
1.8	7685.01	169.45	1157977458	0.00	0.00	._pow [21]
1.5	7820.06	135.05	100	1350.50	3267.00	.sgstress [20]
1.4	7949.59	129.53				.memcpy [24]
1.3	8072.86	123.27	135279	0.91	0.91	.tfila [25]
1.1	8177.83	104.97	135279	0.78	0.78	.tlafi [26]
1.1	8281.26	103.43				.__mcount [27]
1.0	8377.63	96.37	57500	1.68	1.68	.ctdma3i [28]
1.0	8467.61	89.98	38400	2.34	2.34	.tdmaj [29]
0.9	8552.14	84.53	1157922610	0.00	0.00	.expinner2 [30]
0.8	8622.91	70.77	38400	1.84	1.84	.tdmai [31]
0.8	8692.75	69.84	1157922610	0.00	0.00	.loginner2 [32]
0.7	8757.37	64.62	100	646.20	646.20	.upcalc [33]
0.6	8814.75	57.38	100	573.80	573.80	.rhsdp [34]
0.6	8871.65	56.90	100	569.00	34295.86	.takedp [5]
0.6	8927.18	55.53	200	277.65	277.65	.cfl [35]
0.5	8973.28	46.10	100	461.00	67264.06	.getup [3]
0.3	9003.86	30.58	101	302.77	302.77	.divcheck [37]
0.3	9030.97	27.11	100	271.10	31265.30	.uhcalc [7]
0.3	9054.59	23.62	176105	0.13	11.59	.cfila [9]
0.2	9076.38	21.79	10	2179.00	2668.07	.insfield [38]
0.2	9097.50	21.12				.FormatControl [41]
0.2	9113.49	15.99	100	159.90	159.90	.taver [42]

<최적화 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 6339.37 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
23.4	1482.97	1482.97	100	14829.70	14829.70	.energy@OL@1 [2]
8.5	2021.58	538.61	100	5386.10	5386.10	.rhs3@OL@1 [8]
7.2	2478.77	457.19	291092	1.57	2.56	.itsolv [5]
7.1	2931.51	452.74	100	4527.40	4527.40	.rhs2@OL@1 [9]
7.0	3372.78	441.27	100	4412.70	4412.70	.rhs1@OL@1 [10]
4.6	3661.31	288.53	582184	0.50	0.50	.tribi [14]
3.9	3906.71	245.40	100	2454.00	2626.30	.getuh3@OL@1 [17]
3.7	4142.98	236.27	100	2362.70	2831.77	.getuh2@OL@1 [15]
3.3	4351.62	208.64	189237	1.10	1.10	.resca [18]
2.9	4533.95	182.33	19200	9.50	9.50	.strain [19]
2.5	4694.67	160.72	108017	1.49	1.49	.resma [20]
2.5	4855.16	160.49	100	1604.90	1604.90	.getuh3@OL@5 [21]
2.2	4993.76	138.60	100	1386.00	3209.30	.sgstress [13]

2.1	5126.38	132.62				._moveeq [23]
1.7	5236.15	109.77	100	1097.70	1097.70	.getuh3@OL@4 [24]
1.6	5335.05	98.90	57500	1.72	1.72	.ctdma3i [26]
1.5	5432.56	97.51	100	975.10	1444.17	.getuh1@OL@1 [22]
1.4	5521.59	89.03	38400	2.32	2.32	.tdmaj [27]
1.1	5593.36	71.77	100	717.70	1014.47	.getuh3@OL@3 [25]
1.0	5658.47	65.11	100	651.10	651.10	.upcalc@OL@1 [28]
0.9	5716.76	58.29	200	291.45	291.45	.cfl@OL@1 [30]
0.9	5771.92	55.16	100	551.60	551.60	.rhsdp@OL@1 [33]
0.8	5823.61	51.69	38400	1.35	1.35	.tdmai [34]
0.7	5868.22	44.61	70515	0.63	0.63	.tfila [35]
0.5	5899.98	31.76	100	317.60	317.60	.getup@OL@1 [37]
0.5	5930.61	30.63	100	306.30	306.30	.uhcalc@OL@1 [38]
0.5	5959.27	28.66	101	283.76	283.76	.divcheck@OL@1 [39]
0.4	5986.26	26.99	100	269.90	600.14	.getuh1@OL@2 [29]
0.4	6011.39	25.13	100	251.30	579.82	.getuh2@OL@2 [31]
0.4	6035.72	24.33	100	243.30	573.54	.getuh3@OL@2 [32]
0.4	6059.25	23.53	10	2353.00	2353.00	.insfield [42]
0.3	6078.10	18.85				.FormatControl [43]
0.3	6094.12	16.02	100	160.20	160.20	.taver [44]
0.2	6109.95	15.83				.__mcount [45]
0.2	6124.05	14.10	118722	0.12	6.34	.cfila [4]
0.2	6138.10	14.05	70515	0.20	0.20	.tlafi [46]
0.2	6150.35	12.25				.WriteUnit [47]
0.2	6161.48	11.13	62358401	0.00	0.00	.cvtloop [48]
0.2	6171.91	10.43	65108196	0.00	0.00	.__cvt_r [41]
0.2	6182.25	10.34				.__mcount [49]
0.2	6192.47	10.22	10	1022.00	1022.00	.profile [50]
0.2	6202.50	10.03				.IOWrite [51]
0.2	6212.25	9.75	100	97.50	97.50	.takedp@OL@1 [52]

<OpenMP 병렬코드(32 threads) profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 11798.34 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
22.2	2621.70	2621.70				.ThdCode [3]
14.4	4318.19	1696.49	16551	102.50	102.50	.energy@OL@1 [6]
11.0	5620.72	1302.53	258314	5.04	7.45	.itsolv [4]
5.3	6242.70	621.98	521377	1.19	1.19	.tribi [10]
5.2	6851.04	608.34	2405	252.95	252.95	.rhs3@OL@1 [11]
4.3	7357.66	506.62	2194	230.91	230.91	.rhs2@OL@1 [12]
4.2	7858.92	501.26	2502	200.34	200.34	.rhs1@OL@1 [13]
4.0	8329.70	470.78	188383	2.50	2.50	.resca [14]

3.3	8723.61	393.91	104943	3.75	3.75	.resma [15]
2.5	9017.85	294.24	9939	29.60	32.13	.getuh3@OL@1 [19]
2.2	9281.93	264.08	55091	4.79	4.79	.ctdma3i [21]
2.1	9533.28	251.35	9743	25.80	34.26	.getuh2@OL@1 [18]
1.5	9708.32	175.04	19200	9.12	9.12	.strain [23]
1.4	9873.62	165.30	7254	22.79	22.79	.getuh3@OL@5 [24]
1.2	10014.45	140.83	13775	10.22	10.22	.tdmai [26]
1.2	10155.01	140.56	9725	14.45	25.92	.getuh1@OL@1 [22]
1.2	10295.38	140.37	3544	39.61	39.61	.getuh3@OL@4 [27]
1.1	10428.43	133.05	100	1330.50	3080.90	.sgstress [20]
1.0	10550.59	122.16				._moveeq [30]
1.0	10665.01	114.42	9809	11.66	11.66	.upcalc@OL@1 [31]
0.8	10762.63	97.62	69921	1.40	1.40	.tfila [33]
0.8	10860.11	97.48	12973	7.51	7.51	.tdmaj [34]
0.7	10944.94	84.83	9573	8.86	10.88	.getuh3@OL@3 [32]
0.5	11008.49	63.55	1030	61.70	61.70	.cfl@OL@1 [35]
0.5	11070.32	61.83	10251	6.03	6.03	.getup@OL@1 [36]
0.5	11130.31	59.99	3418	17.55	17.55	.rhsdp@OL@1 [37]
0.5	11186.28	55.97	16297	3.43	8.60	.getuh2@OL@2 [28]
0.5	11241.06	54.78	16328	3.35	8.79	.getuh3@OL@2 [25]
0.4	11289.13	48.07	10149	4.74	4.74	.uhcalc@OL@1 [38]
0.4	11331.83	42.70	16368	2.61	8.17	.getuh1@OL@2 [29]
0.3	11371.68	39.85	88972	0.45	20.02	.cfila [5]
0.3	11404.79	33.11	683	48.48	48.48	.divcheck@OL@1 [39]
0.2	11433.64	28.85	16348	1.76	1.76	.takedp@OL@1 [41]
0.2	11457.74	24.10	16408	1.47	1.47	.takedp@OL@4 [44]
0.2	11480.82	23.08	3871	5.96	392.95	.takedp@OL@2 [7]
0.2	11503.38	22.56	3670	6.15	401.29	.takedp@OL@3 [8]
0.2	11524.84	21.46	70473	0.30	0.30	.tlafi [45]
0.2	11545.82	20.98	10	2098.00	2098.00	.insfield [46]
0.2	11564.88	19.06				.FormatControl [47]

3. Hpmcount 결과

hpmcount는 사용자가 작성한 프로그램의 실제 실행 시간과 하드웨어 카운터에 의해 수집되는 정보, 사용자원 등의 전반적인 성능 정보를 제공하는 커맨드라인 유틸리티이다. 아래는 inflow와 main.x 두 실행파일의 두 번째 실행(실행 2)에 대한 hpmcount 결과이다.

3.1 inflow 실행파일의 hpmcount 결과

<Original 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

 Total execution time (wall clock time): 376.124105 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode                 : 365.130000 seconds
 Total amount of time in system mode               : 1.100000 seconds
 Maximum resident set size                         : 167756 Kbytes
 Average shared memory use in text segment         : 280565 Kbytes*sec
 Average unshared memory use in data segment       : 60716757 Kbytes*sec
 Number of page faults without I/O activity        : 41966
 Number of page faults with I/O activity           : 0
 Number of times process was swapped out           : 0
 Number of times file system performed INPUT       : 0
 Number of times file system performed OUTPUT      : 0
 Number of IPC messages sent                       : 0
 Number of IPC messages received                   : 0
 Number of signals delivered                       : 0
 Number of voluntary context switches              : 4189
 Number of involuntary context switches            : 375

 #######  End of Resource Statistics  ########

  PM_FPU_FDIV (FPU executed FDIV instruction)                 :       1828139926
  PM_FPU_FMA (FPU executed multiply-add instruction)          :     110534376239
  PM_FPU0_FIN (FPU0 produced a result)                        :     181224208688
  PM_FPU1_FIN (FPU1 produced a result)                        :     168286001068
  PM_CYC (Processor cycles)                                   :     620907017889
  PM_FPU_STF (FPU executed store instruction)                 :      45018338317
```

```
PM_INST_CMPL (Instructions completed)                          :      637123881978
PM_LSU_LDF (LSU executed Floating Point load instruction) :      188792776878

Utilization rate                                  :              96.878 %
Load and store operations                         :         233811.115 M
MIPS                                              :           1693.919
Instructions per cycle                            :              1.026
HW Float points instructions per Cycle            :              0.563
Floating point instructions + FMAs                :         415026.248 M
Float point instructions + FMA rate               :           1103.429 Mflip/s
FMA percentage                                    :              53.266 %
Computation intensity                             :              1.775
```

<최적화 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 361.616956 seconds

########  Resource Usage Statistics  ########

Total amount of time in user mode                 : 352.580000 seconds
Total amount of time in system mode               : 1.250000 seconds
Maximum resident set size                         : 190472 Kbytes
Average shared memory use in text segment         : 180664 Kbytes*sec
Average unshared memory use in data segment       : 66489627 Kbytes*sec
Number of page faults without I/O activity        : 47600
Number of page faults with I/O activity           : 0
Number of times process was swapped out           : 0
Number of times file system performed INPUT       : 0
Number of times file system performed OUTPUT      : 0
Number of IPC messages sent                       : 0
Number of IPC messages received                   : 0
Number of signals delivered                       : 0
Number of voluntary context switches              : 4355
Number of involuntary context switches            : 354

#######  End of Resource Statistics  ########

PM_FPU_FDIV (FPU executed FDIV instruction)                   :        1545081024
PM_FPU_FMA (FPU executed multiply-add instruction)            :      110869254361
PM_FPU0_FIN (FPU0 produced a result)                          :      168209727979
PM_FPU1_FIN (FPU1 produced a result)                          :      161086563149
PM_CYC (Processor cycles)                                     :      599636647856
PM_FPU_STF (FPU executed store instruction)                   :       35944511419
```

```
PM_INST_CMPL (Instructions completed)                          :      608483643976
PM_LSU_LDF (LSU executed Floating Point load instruction) :      179387846549

Utilization rate                                  :              97.313 %
Load and store operations                         :         215332.358 M
MIPS                                              :            1682.675
Instructions per cycle                            :               1.015
HW Float points instructions per Cycle            :               0.549
Floating point instructions + FMAs                :         404221.034 M
Float point instructions + FMA rate               :            1117.815 Mflip/s
FMA percentage                                    :              54.856 %
Computation intensity                             :               1.877
```

<OpenMP 병렬코드(32 threads) hpmcount 결과>

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 46.886394 seconds

########  Resource Usage Statistics  ########

Total amount of time in user mode                 : 1482.890000 seconds
Total amount of time in system mode               : 7.780000 seconds
Maximum resident set size                         : 203528 Kbytes
Average shared memory use in text segment         : 510148 Kbytes*sec
Average unshared memory use in data segment       : 293269080 Kbytes*sec
Number of page faults without I/O activity        : 51355
Number of page faults with I/O activity           : 136
Number of times process was swapped out           : 0
Number of times file system performed INPUT       : 0
Number of times file system performed OUTPUT      : 0
Number of IPC messages sent                       : 0
Number of IPC messages received                   : 0
Number of signals delivered                       : 0
Number of voluntary context switches              : 4582
Number of involuntary context switches            : 13839

#######  End of Resource Statistics  ########

PM_FPU_FDIV (FPU executed FDIV instruction)                    :      1537904137
PM_FPU_FMA (FPU executed multiply-add instruction)             :    110562910042
PM_FPU0_FIN (FPU0 produced a result)                           :    174231921490
PM_FPU1_FIN (FPU1 produced a result)                           :    160075489752
PM_CYC (Processor cycles)                                      :   2522028427143
PM_FPU_STF (FPU executed store instruction)                    :     36018378858
```

```
PM_INST_CMPL (Instructions completed)                          :    2384035190150
PM_LSU_LDF (LSU executed Floating Point load instruction) :      186119957410

Utilization rate                                  :          3156.702 %
Load and store operations                         :        222138.336 M
MIPS                                              :         50847.058
Instructions per cycle                            :             0.945
HW Float points instructions per Cycle            :             0.133
Floating point instructions + FMAs                :        408851.942 M
Float point instructions + FMA rate               :          8720.055 Mflip/s
FMA percentage                                    :            54.085 %
Computation intensity                             :             1.841
```

단위시간당 부동소수 연산회수를 나타내는 Float point instructions + FMA rate이 1103 Mflip/s(original 코드)에서 1117 Mflip/s(최적화 코드)로 변화하였다. 성능면에서 큰 차이가 없는 것은, 이는 앞에서 언급했듯 original 코드가 이미 한 번 최적화를 수행한 것이기 때문이다. Inflow는 main.x에서 사용될 입력 데이터를 발생시키는 코드로 main.x 보다는 적은 시간을 소비하며 주로 병렬화를 통해 속도향상을 얻고 있다.

3.2 main.x 실행파일의 hpmcount 결과

<Original 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

 Total execution time (wall clock time): 1149.428384 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode                 : 1139.680000 seconds
 Total amount of time in system mode               : 4.010000 seconds
 Maximum resident set size                         : 2306888 Kbytes
 Average shared memory use in text segment         : 466985 Kbytes*sec
 Average unshared memory use in data segment       : 2147483647 Kbytes*sec
 Number of page faults without I/O activity        : 577726
 Number of page faults with I/O activity           : 4
 Number of times process was swapped out           : 0
 Number of times file system performed INPUT       : 0
 Number of times file system performed OUTPUT      : 0
```

```
Number of IPC messages sent                    : 0
Number of IPC messages received                : 0
Number of signals delivered                    : 0
Number of voluntary context switches           : 2787
Number of involuntary context switches         : 1147

########  End of Resource Statistics  ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                    :      16047298383
 PM_FPU_FMA (FPU executed multiply-add instruction)             :     128603158648
 PM_FPU0_FIN (FPU0 produced a result)                           :     209616262223
 PM_FPU1_FIN (FPU1 produced a result)                           :     211815676244
 PM_CYC (Processor cycles)                                      :    1938095757256
 PM_FPU_STF (FPU executed store instruction)                    :      74619712206
 PM_INST_CMPL (Instructions completed)                          :    1206592810796
 PM_LSU_LDF (LSU executed Floating Point load instruction)      :     277576399315

 Utilization rate                               :          98.952 %
 Load and store operations                      :      352196.112 M
 MIPS                                           :         1049.733
 Instructions per cycle                         :           0.623
 HW Float points instructions per Cycle         :           0.217
 Floating point instructions + FMAs             :      475415.385 M
 Float point instructions + FMA rate            :      413.610 Mflip/s
 FMA percentage                                 :          54.101 %
 Computation intensity                          :           1.350
```

<최적화 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

 Total execution time (wall clock time): 640.616355 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode            : 624.950000 seconds
 Total amount of time in system mode          : 4.020000 seconds
 Maximum resident set size                    : 1804636 Kbytes
 Average shared memory use in text segment    : 264316 Kbytes*sec
 Average unshared memory use in data segment  : 1080410600 Kbytes*sec
 Number of page faults without I/O activity   : 451756
 Number of page faults with I/O activity      : 3
 Number of times process was swapped out      : 0
 Number of times file system performed INPUT  : 0
 Number of times file system performed OUTPUT : 0
```

```
Number of IPC messages sent                    : 0
Number of IPC messages received                : 0
Number of signals delivered                    : 0
Number of voluntary context switches           : 4612
Number of involuntary context switches         : 663

########  End of Resource Statistics  ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                    :        5999486837
 PM_FPU_FMA (FPU executed multiply-add instruction)             :       91995771627
 PM_FPU0_FIN (FPU0 produced a result)                           :      153252256623
 PM_FPU1_FIN (FPU1 produced a result)                           :      149468331086
 PM_CYC (Processor cycles)                                      :     1063066168207
 PM_FPU_STF (FPU executed store instruction)                    :       42962431557
 PM_INST_CMPL (Instructions completed)                          :      634107585921
 PM_LSU_LDF (LSU executed Floating Point load instruction) :       176458949541

 Utilization rate                                :              97.385 %
 Load and store operations                       :          219421.381 M
 MIPS                                            :              989.840
 Instructions per cycle                          :                0.596
 HW Float points instructions per Cycle          :                0.285
 Floating point instructions + FMAs              :          351753.928 M
 Float point instructions + FMA rate             :              549.087 Mflip/s
 FMA percentage                                  :               52.307 %
 Computation intensity                           :                1.603
```

<OpenMP 병렬코드(32 threads) hpmcount 결과>

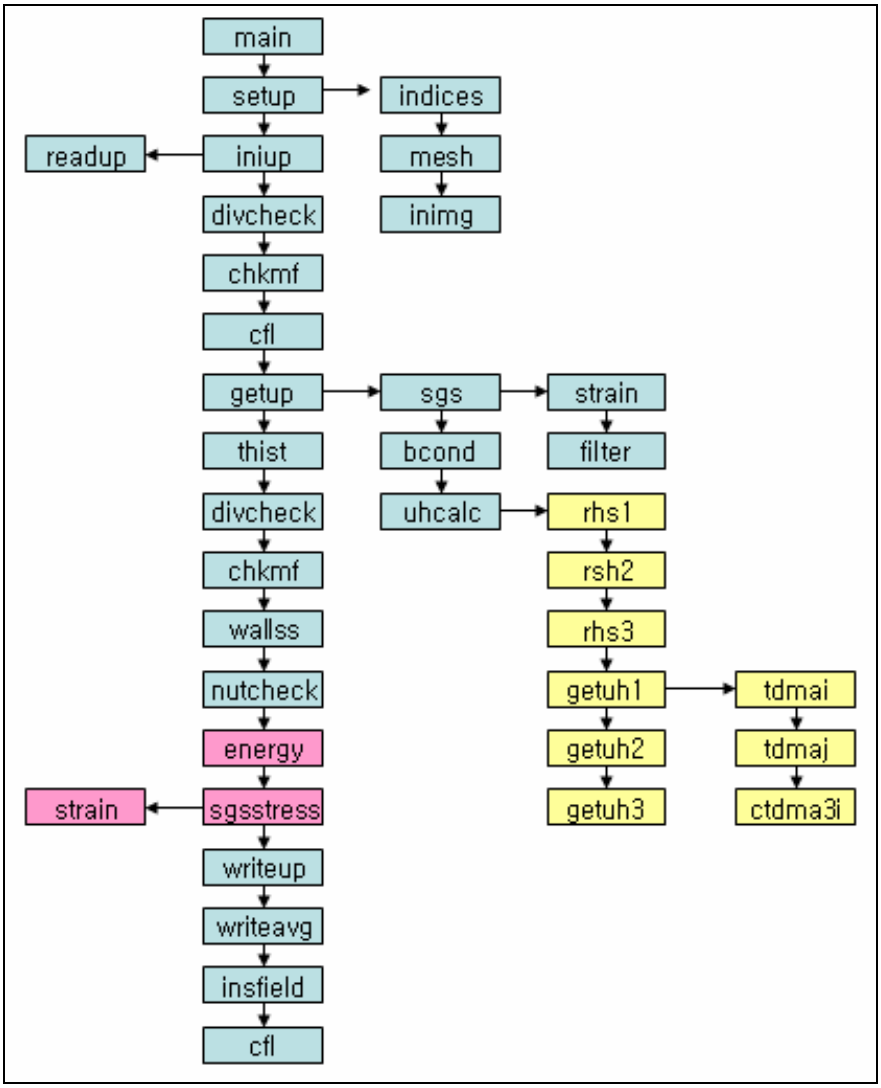
```
hpmcount (V 2.4.3) summary

 Total execution time (wall clock time): 1122.704714 seconds

 ########  Resource Usage Statistics  ########

 Total amount of time in user mode            : 35778.020000 seconds
 Total amount of time in system mode          : 41.680000 seconds
 Maximum resident set size                    : 1991524 Kbytes
 Average shared memory use in text segment    : 19340222 Kbytes*sec
 Average unshared memory use in data segment  : 2147483647 Kbytes*sec
 Number of page faults without I/O activity   : 501100
 Number of page faults with I/O activity      : 100
 Number of times process was swapped out      : 0
 Number of times file system performed INPUT  : 0
 Number of times file system performed OUTPUT : 0
```

```
Number of IPC messages sent                    : 0
Number of IPC messages received                : 0
Number of signals delivered                    : 0
Number of voluntary context switches           : 29238
Number of involuntary context switches         : 214747

#######   End of Resource Statistics   ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                    :       60463353445
 PM_FPU_FMA (FPU executed multiply-add instruction)             :      939077528764
 PM_FPU0_FIN (FPU0 produced a result)                           :     1523279932835
 PM_FPU1_FIN (FPU1 produced a result)                           :     1510868193283
 PM_CYC (Processor cycles)                                      :    60863505123061
 PM_FPU_STF (FPU executed store instruction)                    :      427864323806
 PM_INST_CMPL (Instructions completed)                          :    51822153973224
 PM_LSU_LDF (LSU executed Floating Point load instruction)      :     1856317505637

 Utilization rate                              :          3181.426 %
 Load and store operations                     :       2284181.829 M
 MIPS                                          :          46158.312
 Instructions per cycle                        :              0.851
 HW Float points instructions per Cycle        :              0.050
 Floating point instructions + FMAs            :       3545361.331 M
 Float point instructions + FMA rate           :          3157.875 Mflip/s
 FMA percentage                                :             52.975 %
 Computation intensity                         :              1.552
```

main.x에서는 단위시간당 부동소수 연산회수가 413.6Mflip/s(original)에서 549.0Mflip/s(최적화)로 증가하였고, 부동소수 연산회수 자체도 475415.385M(original)에서 351753.928M(최적화)로 감소해 코드 성능과 효율이 개선되었음을 확인할 수 있다.

4. inflow 코드 최적화/병렬화 비교 및 분석

inflow는 main.x에서 사용할 입력 데이터를 생성하는 코드로, main.x에 비해 적은 시간을 소비한다. 다음은 inflow 코드의 실행 순서를 보여 주는 흐름도 이다.



< 그림 III.2 Inflow Flowchart >

4.1 OpenMPcut.f

< 최적화/병렬화 코드 >

```
1                       subroutine OpenMPcut(ks,kstart,kend,nthds)
2                       integer ks(2,*)
4                       nn=kend-kstart+1
5                       nblk=nn/nthds
6                       ndelta=mod(nn,nthds)
7                       if(ndelta.gt.0) then
 8                          k=1
 9                          ks(1,1)=1
10                          ks(2,1)=ks(1,1)+nblk
11                       do k=2,ndelta
12                          ks(1,k)=ks(1,k-1)+nblk+1
13                          ks(2,k)=ks(1,k)+nblk
14                       enddo
15                          k=ndelta+1
16                          ks(1,k)=ks(1,k-1)+nblk+1
17                          ks(2,k)=ks(1,k)+nblk-1
18                       do k=ndelta+2,nthds
19                          ks(1,k)=ks(1,k-1)+nblk
20                          ks(2,k)=ks(1,k)+nblk-1
21                       enddo
22                       else
23                          ks(1,1)=1
24                          ks(2,1)=ks(1,1)+nblk-1
25                       do k=2,nthds
26                          ks(1,k)=ks(1,k-1)+nblk
27                          ks(2,k)=ks(1,k)+nblk-1
28                       enddo
29                       endif
30                       do k=1,nthds
31                          ks(1,k)=ks(1,k)+kstart-1
32                          ks(2,k)=ks(2,k)+kstart-1
33                       enddo
```

③ 병렬화를 위한 루틴이며, kstart에서 kend까지의 값을 주어진 개수(nthds)만큼의 스레드에 블록분할해 준다. nthds개의 스레드 중 p(0 ≤ p ≤ nthds-1) 스레드가 할당 받게되는 데이터 영역은 ks(1,p)부터 ks(2,p)까지가 된다.

kend = 100(kstart=1, kend=100), nthds = 2 일 때

ks(1,1) = 1 , ks(2,1) = 50

ks(1,2) = 51 , ks(2,2) = 100

kend = 100(kstart=1, kend=100), nthds = 3 일 때

ks(1, 1) = 1 , ks(2, 1) = 34 (34)

ks(1, 2) = 35 , ks(2, 2) = 67 (33)

ks(1, 3)= 68 , ks(2, 3) = 100 (33)

kend = 100(kstart=1, kend=100), nthds = 6 일 때

ks(1, 1) = 1 , ks(2, 1) = 17 (17)

ks(1. 2) = 18 , ks(2, 2) = 34 (17)

ks(1, 3)= 35 , ks(2, 3) = 51 (17)

ks(1, 4) = 52 , ks(2, 4) = 68 (17)

ks(1. 5) = 69 , ks(2, 5) = 84 (16)

ks(1, 6)= 85 , ks(2, 6) = 100 (16)

4.2 divcheck.f

< Original 코드 >

```
10                      DO 20 K=1,N3M
11                      KP=KPA(K)
12                      DO 20 J=1,N2M
13                      JP=J+1
14                      DO 20 I=1,N1M
15          10          IP=IPA(I)
16          111         DIV=(U(IP,J,K,1)-U(I,J,K,1))*DX1
17                     >    +(U(I,JP,K,2)-U(I,J,K,2))/DR(J)/RM(J)
18                     >    +(U(I,J,KP,3)-U(I,J,K,3))*DX3/RM(J)**2.0
19          59          DIVMAX=AMAX1(ABS(DIV),DIVMAX)
20                  20  CONTINUE
```

< 최적화/병렬화 코드>

```
16                  !$omp parallel do private(KP,JP,IP,DIV) reduction(max:DIVMAX)
17                      DO 20 K=1,N3M
18                         KP=KPA(K)
19                      DO 20 J=1,N2M
20                         JP=J+1
21                      DO 20 I=1,N1M
```

```
22          27                IP=IPA(I)
23          83              DIV=(U(IP,J,K,1)-U(I,J,K,1))*DX1
24                        >    +(U(I,JP,K,2)-U(I,J,K,2))/DR(J)/RM(J)
25                        >    +(U(I,J,KP,3)-U(I,J,K,3))*DX3/RM(J)**2.0
26          31              DIVMAX=AMAX1(ABS(DIV),DIVMAX)
27           5        20    CONTINUE
```

⑦ parallel do를 이용해 K루프를 병렬화 하였다.

⑧ 스레드마다 따로 값을 가져야 하는 KP, JP, IP, DIV 변수는 private 속성을 주었고, 병렬로 계산되는 DIV의 최대값 DIVMAX를 찾기 위해 reduction지시어와 max 연산을 이용하고 있다.

4.3 chkmf.f

< Original 코드 >

```
 8                  C       Calculate the mass flow rate in x1 direction
 9                          FLOW1=0.0
10                          DO 2 K=1,N3M
11                          DO 2 J=1,N2M
12          4               FLOW1=FLOW1
13                        >          +U(1,J,K,1)*DR(J)/DX3*RM(J)
14                       2 CONTINUE
15
16                  C       Calculate the mass flow rate in x3 direction
17                          FLOW3=0.0
18                          DO 3 J=1,N2M
19                          DO 3 I=1,N1M
20                          FLOW3=FLOW3
21                        >          +U(I,J,1,3)*DR(J)/DX1/RM(J)
22                       3 CONTINUE
```

< 최적화/병렬화 코드>

```
18                  !$omp parallel do reduction(+:FLOW1)
19                          DO 2 K=1,N3M
20                          DO 2 J=1,N2M
21          4               FLOW1=FLOW1
22                        >          +U(1,J,K,1)*DR(J)/DX3*RM(J)
23                       2 CONTINUE
24
```

```
25                      C       Calculate the mass flow rate in x3 direction
26                              FLOW3=0.0
27                      !$omp parallel do reduction(+:FLOW3)
28                              DO 3 J=1,N2M
29                              DO 3 I=1,N1M
30                              FLOW3=FLOW3
31                             >          +U(I,J,1,3)*DR(J)/DX1/RM(J)
32                           3 CONTINUE
```

① parallel do를 이용해 K루프에 대해서 병렬화 하였다.

② 병렬로 계산되는 FLOW1과 FLOW3의 최종 합을 얻기 위해 reduction 지시어를 사용하였다.

4.4 cfl.f

< Original 코드 >

```
15                              DO 10 K=1,N3M
16                              KP=KPA(K)
17              1               DO 10 J=1,N2M
18                              JP=J+1
19                              DO 10 I=1,N1M
20                              IP=IPA(I)
21            254               CFLL=ABS(U(I,J,K,1)+U(IP,J,K,1))*0.5*DX1
22                             >     +ABS(U(I,J,K,2)+U(I,JP,K,2))*0.5/DR(J)/RM(J)
23                             >     +ABS(U(I,J,K,3)+U(I,J,KP,3))*0.5*DX3/RM(J)**2.0
24              6              IF(CFLL.GT.CFLM) THEN
25                             CFLM=CFLL
26                             IMAX=I
27                             JMAX=J
28                             KMAX=K
29                             ENDIF
30             52        10    CONTINUE
```

< 최적화/병렬화 코드>

```
20                      !$omp parallel do private(ks,ke,KP,JP,IP,CFLL)
21                              do ii=0,nthds-1
22                                 ks=kse(1,ii)
23                                 ke=kse(2,ii)
24                              CFLM(ii)=0.0
25                              IMAX(ii)=0
```

```
26                          JMAX(ii)=0
27                          KMAX(ii)=0
29                          DO 10 K=ks,ke
30                              KP=KPA(K)
31                          DO 10 J=1,N2M
32                              JP=J+1
33                          DO 10 I=1,N1M
34            1                 IP=IPA(I)
35          262                 CFLL=ABS(U(I,J,K,1)+U(IP,J,K,1))*0.5*DX1
36                             >   +ABS(U(I,J,K,2)+U(I,JP,K,2))*0.5/DR(J)/RM(J)
37                             >   +ABS(U(I,J,K,3)+U(I,J,KP,3))*0.5*DX3/RM(J)**2.0
38            5                 IF(CFLL.GT.CFLM(ii)) THEN
39                               CFLM(ii)=CFLL
40                               IMAX(ii)=I
41                               JMAX(ii)=J
42                               KMAX(ii)=K
43                              ENDIF
44           41          10     CONTINUE
46                          ENDDO
47                              ii=0
48                              i1=ii
49                           CFLM1=CFLM(ii)
50                          do ii=1,nthds-1
51                          CFLL=CFLM(ii)
52                          IF(CFLL.GT.CFLM1) THEN
53                           CFLM1=CFLM(ii)
54                           i1=ii
55                          ENDIF
56                          ENDDO
57                           IMAX1=IMAX(i1)
58                           JMAX1=JMAX(i1)
59                           KMAX1=KMAX(i1)
```

① OpenMP 지시어를 사용하여 병렬화를 하였다.

② 병렬 영역내에서 CFLL의 최대값 CFLM을 구하는 것에 대해서는 앞서와 마찬가지로 reduction을 이용해 처리 가능하나 그때의 I, J, K 값 IMAX, JMAX, KMAX를 병렬로 찾는데 문제가 있다. 각 스레드마다 CFLL의 로컬 최대값을 구하고 그때의 IMAX, JMAX, KMAX를 스레드마다 따로 저장시켜 병렬 영역 밖으로 빼내기 위해 스레드 수만큼의 크기를 가지는 배열을 이용하고 있다.

③ 사용하는 스레드 개수(nthds)에 의존해 K 루프를 명시적으로 분할하였다. 명시적인 분할을 위해 호출해 사용하는 루틴이 openmpcut이다.

④ 각 스레드가 계산한 로컬 데이터 CFLM(n), IMAX(n), JMAX(n), KMAX(n)을 비교해 글로벌 최대값 데이터를 얻기 위해 병렬 영역 밖에서 다시 한 번 비교 과정을 거친다.

4.5 getup.f

< Original 코드 >

```
12                  C INITIALIZE THE INTERMEDIATE VELOCITY AND PRESSURE
13                        DO 10 NV=1,3
14                        DO 10 K=1,N3
15            3           DO 10 J=0,N2
16                        DO 10 I=1,N1
17          189       10  UH(I,J,K,NV)=0.0
18                        DO 20 K=1,N3
19            1           DO 20 J=1,N2
20                        DO 20 I=1,N1
21           59       20  DP(I,J,K)=0.0
22                        DO 30 K=1,N3
23            1           DO 30 J=0,N2
24                        DO 30 I=1,N1
25           62       30  SGSVIS(I,J,K)=0.0
```

< 최적화/병렬화 코드 >

```
15                  C INITIALIZE THE INTERMEDIATE VELOCITY AND PRESSURE
16                  !$omp parallel do
17                        DO K=1,N3
18                         DO J=0,N2
19                         DO I=1,N1
20          147               UH(I,J,K,1)=0.0
21           82               UH(I,J,K,2)=0.0
22           68               UH(I,J,K,3)=0.0
23            6               SGSVIS(I,J,K)=0.0
24                         ENDDO
25            1            ENDDO
26                         DO J=1,N2
27                         DO I=1,N1
```

```
28          74              DP(I,J,K)=0.0
29                        ENDDO
30                        ENDDO
31           1           ENDDO
```

① original 코드에서 3 개의 루프가 동일한 K 루프를 가지고 있다. 병렬화 효과를 극대화하기 위해 보다 큰 루프를 만들 필요가 있고 이를 위해 최적화/병렬화 코드에서는 세 개의 루프를 하나로 합쳤다. 이 과정에서 K 루프 밖에 NV 루프가 있는 첫 번째 루프에서는 NV 루프를 안으로 집어 넣었고 반복이 3회 밖에 되지 않으므로 루프를 풀어 버렸다(unrolling).

4.6 bcond.f

< Original 코드 >

```
 8                      DO 10 K=1,N3M
 9                      DO 10 I=1,N1M
10          1           UBC(I,K,1,1)=0.0
11                      UBC(I,K,2,1)=0.0   ! V=r*V_r       : Radial flux
12          3           UBC(I,K,3,1)=0.0   ! W=r*V_theta : Azimutal flux
13                      UBC(I,K,1,2)=0.0   ! PIPE WALL
14          2           UBC(I,K,2,2)=0.0
15          3           UBC(I,K,3,2)=0.5*ROTN*ALR
16                   10 CONTINUE
```

< 최적화/병렬화 코드>

```
13                   !$omp parallel do
14                      DO 10 K=1,N3M
15                      DO 10 I=1,N1M
16                      UBC(I,K,1,1)=0.0
17                      UBC(I,K,2,1)=0.0   ! V=r*V_r       : Radial flux
18          1           UBC(I,K,3,1)=0.0   ! W=r*V_theta : Azimutal flux
19                      UBC(I,K,1,2)=0.0   ! PIPE WALL
20          4           UBC(I,K,2,2)=0.0
21          6           UBC(I,K,3,2)=0.5*ROTN*ALR
22                   10 CONTINUE
```

② parallel Do를 이용해 K 루프를 병렬화 하였다.

4.7 rhs1.f~rhs3.f

< Original 코드 >

```
 12                              DO 10 K=1,N3M
 15                              DO 10 J=1,N2M
 20                              DO 10 I=1,N1M
 24              14              SGSI2=SGSVIS(I ,J,K)

 ---------------- 생     략 --------------
176              1         10    CONTINUE
```

< 최적화/병렬화 코드 >

```
23                      !$omp   parallel do private(KP,KM,JP,JM,IP,IM,FJUM,FJUP,
24                      !$omp& SGSI2,SGSI1,NUTI,SGSJP,SGSJC,SGSJM,SGSJ2,SGSJ1,
25                      !$omp& NUTJ,SGSKP,SGSKC,SGSKM,SGSK2,SGSK1,NUTK,
26                      !$omp& U2,U1,VIS11,V2,V1,VIS12,W2,W1,VIS13,VISCOS,
27                      !$omp& BC_DOWN,BC_UP,BC, PRESSG1,
28                      !$omp& APJ,ACJ,AMJ,API,ACI,AMI,APK,ACK,AMK,
29                      !$omp& RM11U_N,RM12V_N,RM13W_N)
30                              DO 10 K=1,N3M
33                              DO 10 J=1,N2M
38                              DO 10 I=1,N1M
42              32              SGSI2=SGSVIS(I ,J,K)

  ---------------- 생     략 --------------
194              1         10    CONTINUE
```

③ parallel Do를 이용해 K 루프를 병렬화 하였다.

④ rhs1.f~ rhs3.f 는 모두 동일하게 병렬화 되어있다.

4.8 getuh1.f

< Original 코드 >

```
20                              DO 2 K=1,N3M
21                              DO 20 J=1,N2M
26                              DO 20 I=1,N1M
```

```
------------- 생  략 ------------
60          251             AMJ(I,J)=FJUM*(
61                         >       -0.5*HM(J)*(NUTI+1./RE)+0.25*RM(JM)
62                         >      /RM(J)**2./H(J )*((SGSJ2-SGSJ1)/DR(J)-SGSJC)
63                         >       -0.25*RM(JM)*V1/RM(J)/R(J)/H(J)
64                         >       )*DT
65           45             R2(I,J)=RUH1(I,J,K)*DT
66                    20   CONTINUE
67                         CALL TDMAI(AMJ,ACJ,APJ,R2,R2,1,N2M,1,N1M)
68                         DO 21 J=1,N2M
69                         DO 21 I=1,N1M
70           62       21   UH(I,J,K,1)=R2(I,J)
71                    2    CONTINUE
```

< 최적화/병렬화 코드 >

```
30                  !$omp parallel do private(JP,JM,IP,IM,FJUM,FJUP,
31                  !$omp& SGSJP,SGSJC,SGSJM,SGSJ2,SGSJ1,NUTJ,
32                  !$omp& SGSI2,SGSI1,NUTI,
33                  !$omp& V2,V1,APJ,ACJ,AMJ)
34                        DO 2 K=1,N3M
35                        DO 20 J=1,N2M
40                        DO 20 I=1,N1M
------------- 생  략 ------------
74          235           AMJ(I,J)=FJUM*(
75                       >       -0.5*HM(J)*(NUTI+1./RE) +0.25*RM(JM)
76                       >       /RM(J)**2./H(J )*((SGSJ2-SGSJ1)/DR(J)-SGSJC)
77                       >       -0.25*RM(JM)*V1/RM(J)/R(J)/H(J)
78                       >       )*DT
79           50          UH(I,J,K,1)=RUH1(I,J,K)*DT
80                    20 CONTINUE
81                       CALL TDMAI1(AMJ,ACJ,APJ,UH(0,0,K,1),1,N2M,1,N1M)
82                    2  CONTINUE
```

① original 코드에서 각 K에 대해 RUH1(I,J,K)*DT → R2(I,J) → (call TDMAI) → R2(I,J) → UH(I,J,K,1)로 이어지는 데이터의 흐름을 RUH1(I,J,K)*DT → UH(I,J,K,1) → (call TDMAI1) → UH(I,J,K,1)으로 줄여 불필요한 데이터 할당과 메모리 낭비를 줄이고 있다.

② parallel do 지시어를 삽입하여 K루프를 병렬화 하였다.

③ 최적화 코드의 루틴 TDMAI1에서는 입력과 출력을 더미배열 X(0:M1,0:M2) 하나로 처리한다. 이와 달리 original의 서브루틴 TDMAI

는 입력과 출력을 위해 두 개의 더미배열 D(M1,M2), X(M1,M2)를 쓰고 있으며 그 외 모든 것은 동일하다. 아래에서 두 서브루틴을 비교해 볼 수 있다.

< Original 코드: TDMAI >

```
 1 c***************** TDMAI ***********************
 2       SUBROUTINE TDMAI(A,B,C,D,X,NS,NF,NIS,NIF)
 3       INCLUDE 'PARAM.H'
 4       INCLUDE 'COMMON.H'
 6       REAL A(M1,M2),B(M1,M2),C(M1,M2),D(M1,M2),X(M1,M2)
 7       REAL BET,GAM(M1,0:M2)
 9       J=NS
10       DO I=NIS,NIF
11       BET=1.0/B(I,J)
12       GAM(I,J)=C(I,J)*BET
13       X(I,J)=D(I,J)*BET
14       ENDDO
16       DO 10 J=NS+1,NF
17       DO I=NIS,NIF
18       BET=1.0/(B(I,J)-A(I,J)*GAM(I,J-1))
19       GAM(I,J)=C(I,J)*BET
20       X(I,J)=(D(I,J)-A(I,J)*X(I,J-1))*BET
21       ENDDO
22  10   CONTINUE
24       DO 20 J=NF-1,NS,-1
25       DO I=NIS,NIF
26       X(I,J)=X(I,J)-X(I,J+1)*GAM(I,J)
27       ENDDO
28  20   CONTINUE
30       RETURN
31       END
```

< 최적화/병렬화 코드: TDMAI1 >

```
32 c***************** TDMAI ***********************
33       SUBROUTINE TDMAI1(A,B,C,X,NS,NF,NIS,NIF)
34       INCLUDE 'PARAM.H'
35       COMMON/DIM/N1,N2,N3,N1M,N2M,N3M
36       REAL A(M1,M2),B(M1,M2),C(M1,M2)
37       REAL X(0:M1,0:M2)
38       REAL BET,GAM(M1,0:M2)
40       J=NS
```

```
41          DO I=NIS,NIF
42             BET=1.0/B(I,J)
43             GAM(I,J)=C(I,J)*BET
44             X(I,J)=X(I,J)*BET
45          ENDDO
47          DO J=NS+1,NF
48          DO I=NIS,NIF
49             BET=1.0/(B(I,J)-A(I,J)*GAM(I,J-1))
50             GAM(I,J)=C(I,J)*BET
51             X(I,J)=(X(I,J)-A(I,J)*X(I,J-1))*BET
52          ENDDO
53          ENDDO
55          DO J=NF-1,NS,-1
56          DO I=NIS,NIF
57             X(I,J)=X(I,J)-X(I,J+1)*GAM(I,J)
58          ENDDO
59          ENDDO
61          RETURN
62          END
```

< Original 코드: getuh1.f 계속>

```
 73                      DO 1 K=1,N3M
 74                      DO 10 J=1,N2M
 75                      DO 10 I=1,N1M
 79            10        SGSI2=SGSVIS(I ,J,K)
 80            29        SGSI1=SGSVIS(IM,J,K)
 81            15        NUTI=0.5*(SGSI1+SGSI2)
 83            19        U2=0.5*(U(IP,J,K,1)+U(I ,J,K,1))
 84             8        U1=0.5*(U(I ,J,K,1)+U(IM,J,K,1))
 86            42        API(I,J)=(
 87                     >       -0.5*DX1Q*(NUTI+1./RE)
 88                     >       -0.5*DX1Q*(SGSI2-SGSI1)
 89                     >       +DX1*U2*0.5
 90                     >        )*DT
 91            45        ACI(I,J)=1.0+(
 92                     >        DX1Q*(NUTI+1./RE)
 93                     >       +DX1*(U2*0.5-U1*0.5)
 94                     >        )*DT
 95            25        AMI(I,J)=(
 96                     >       -0.5*DX1Q*(NUTI+1./RE)
 97                     >       +0.5*DX1Q*(SGSI2-SGSI1)
 98                     >       -DX1*U1*0.5
 99                     >        )*DT
100             9        R1(I,J)=UH(I,J,K,1)
```

```
101                     10   CONTINUE
102                          CALL CTDMA1J(AMI,ACI,API,R1,R1,1,N1M,1,N2M)
103                          DO 11 J=1,N2M
104                          DO 11 I=1,N1M
105           36         11   UH(I,J,K,1)=R1(I,J)
106                      1    CONTINUE
```

< 최적화/병렬화 코드: getuh1.f 계속 >

```
84                    !$omp parallel do private(IP,IM,
85                    !$omp& SGSI2,SGSI1,NUTI,
86                    !$omp& U2,U1,API,ACI,AMI)
87                           DO 1 K=1,N3M
88                           DO 10 J=1,N2M
89                           DO 10 I=1,N1M
93             5             SGSI2=SGSVIS(I ,J,K)
94            44             SGSI1=SGSVIS(IM,J,K)
95                           NUTI=0.5*(SGSI1+SGSI2)
97            30             U2=0.5*(U(IP,J,K,1)+U(I ,J,K,1))
98             1             U1=0.5*(U(I ,J,K,1)+U(IM,J,K,1))
100            25             API(I,J)=(
101                          >        -0.5*DX1Q*(NUTI+1./RE)
102                          >        -0.5*DX1Q*(SGSI2-SGSI1)
103                          >        +DX1*U2*0.5
104                          >         )*DT
105            27             ACI(I,J)=1.0+(
106                          >         DX1Q*(NUTI+1./RE)
107                          >        +DX1*(U2*0.5-U1*0.5)
108                          >         )*DT
109            46             AMI(I,J)=(
110                          >        -0.5*DX1Q*(NUTI+1./RE)
111                          >        +0.5*DX1Q*(SGSI2-SGSI1)
112                          >        -DX1*U1*0.5
113                          >         )*DT
114             1         10   CONTINUE
115                         CALL CTDMA1J1(AMI,ACI,API,UH(0,0,K,1),1,N1M,1,N2M)
116                      1    CONTINUE
```

④ 각 K에 대해 UH(I,J,K,1) → R1(I,J) → (call TDMA1J) → R1(I,J) → UH(I,J,K,1)로 진행되는 흐름을 UH(I,J,K,1) → (call TDMA1J1) → UH(I,J,K,1)으로 단순화 시켜 불필요한 데이터 할당과 메모리 낭비를 줄이고 있다.

⑤ parallel do 지시어를 이용해 K 루프를 병렬화 하였다.

< Original 코드: getuh1.f 계속>

```
108                       DO 3 J=1,N2M
109                       DO 30 K=1,N3M
112                       DO 30 I=1,N1M
129           37           APK(I,K)=(
130                       >       -0.5*DX3Q/RM(J)**2.0*(NUTI+1./RE)
131                       >       -0.25*DX3Q/RM(J)**2.0*(SGSK2-SGSK1)
132                       >       +0.25*DX3/RM(J)**2.0*W2
133                       >       )*DT
134           73           ACK(I,K)=1.0+(
135                       >        DX3Q/RM(J)**2.0*(NUTI+1./RE)
136                       >       +0.25*DX3/RM(J)**2.0*(W2-W1)
137                       >       )*DT
138            6           AMK(I,K)=(
139                       >       -0.5*DX3Q/RM(J)**2.0*(NUTI+1./RE)
140                       >       +0.25*DX3Q/RM(J)**2.0*(SGSK2-SGSK1)
141                       >       -0.25*DX3/RM(J)**2.0*W1
142                       >       )*DT
143            9           R3(I,K)=UH(I,J,K,1)
144            1     30    CONTINUE
145                      CALL CTDMA3I(AMK,ACK,APK,R3,UH(0,0,0,1),J,N3M,1,N1M)
146                       3    CONTINUE
```

< 최적화/병렬화 코드: getuh1.f 계속>

```
118                      !$omp parallel do private(KP,KM,IP,IM,
119                      !$omp& SGSKP,SGSKC,SGSKM,SGSK2,SGSK1,NUTK,
120                      !$omp& SGSI2,SGSI1,NUTI,
121                      !$omp& W2,W1,APK,ACK,AMK)
122                       DO 3 J=1,N2M
123                       DO 30 K=1,N3M
126                       DO 30 I=1,N1M
143           28           APK(I,K)=(
144                       >       -0.5*DX3Q/RM(J)**2.0*(NUTI+1./RE)
145                       >       -0.25*DX3Q/RM(J)**2.0*(SGSK2-SGSK1)
146                       >       +0.25*DX3/RM(J)**2.0*W2
147                       >       )*DT
148           48           ACK(I,K)=1.0+(
149                       >        DX3Q/RM(J)**2.0*(NUTI+1./RE)
150                       >       +0.25*DX3/RM(J)**2.0*(W2-W1)
151                       >       )*DT
```

```
152             5           AMK(I,K)=(
153                        >        -0.5*DX3Q/RM(J)**2.0*(NUTI+1./RE)
154                        >        +0.25*DX3Q/RM(J)**2.0*(SGSK2-SGSK1)
155                        >        -0.25*DX3/RM(J)**2.0*W1
156                        >        )*DT
157                    30  CONTINUE
158                     CALL CTDMA3I1(AMK,ACK,APK,UH(0,0,0,1),J,N3M,1,N1M)
159                    3   CONTINUE
```

⑥ UH → R3 → (call CTDMA3I) → UH의 데이터 이동을 UH → (call CTDMA3I1) → UH로 간단히 하였다.

4.9 getduh3.f

getduh3.f에 대한 최적화/병렬화 과정은 전체적으로 getuh1.f의 최적화/병렬화 과정과 동일하게 진행되었다. 아래 부분은 getduh1.f와 다른 부분이라 따로 정리하였다.

< Original 코드 >

```
288                 C       INTERMEDIATE VELOCITY UPDATE
289                         DO 300 K=1,N3M
290                         DO 300 J=1,N2M
291                         DO 300 I=1,N1M
292         128             UH(I,J,K,1)=U(I,J,K,1)+UH(I,J,K,1)
293                 300     UH(I,J,K,3)=U(I,J,K,3)+UH(I,J,K,3)
294
295                         DO 310 K=1,N3M
296                         DO 310 J=2,N2M
297                         DO 310 I=1,N1M
298          64     310     UH(I,J,K,2)=U(I,J,K,2)+UH(I,J,K,2)
299
```

< 최적화/병렬화 코드 >

```
311                 C       INTERMEDIATE VELOCITY UPDATE
312                 !$omp parallel do
313                         DO K=1,N3M
314                          DO J=1,N2M
315                          DO I=1,N1M
```

```
316          119                UH(I,J,K,1)=U(I,J,K,1)+UH(I,J,K,1)
317                             UH(I,J,K,3)=U(I,J,K,3)+UH(I,J,K,3)
318                           ENDDO
319            1              ENDDO
320
321                           DO J=2,N2M
322                           DO I=1,N1M
323           62                UH(I,J,K,2)=U(I,J,K,2)+UH(I,J,K,2)
324                           ENDDO
325                           ENDDO
326                          ENDDO
```

① K 인덱스가 동일한 두 개의 루프를 합치고 병렬화 하였다.

4.10 rhsdp.f

< Original 코드 >

```
10                            DO 10 K=1,N3M
11                            KP=KPA(K)
12                            KM=KMA(K)
13                            DO 10 J=1,N2M
14                            JP=J+1
15                            JM=J-1
16                            FJUM=FJMU(J)
17                            FJUP=FJPA(J)
19                            DO 10 I=1,N1M
20           39               IP=IPA(I)
21                            IM=IMA(I)
23           51               DIVUH=(UH(IP,J,K,1)-UH(I,J,K,1))*DX1+(FJUP
24                           > *UH(I,JP,K,2)-FJUM*UH(I,J,K,2))/DR(J)/RM(J)
25                           >     +(UH(I,J,KP,3)-UH(I,J,K,3))*DX3/RM(J)**2.0
27           85               CBC=(1.-FJUM)*UBC(I,K,2,1)/DR(J)/RM(J)
28                           >          -(1.-FJUP)*UBC(I,K,2,2)/DR(J)/RM(J)
30           27               RDP(I,J,K)=(DIVUH-CBC)/DT
31                      10    CONTINUE
```

< 최적화/병렬화 코드 >

```
19                      !$omp parallel do private(KP,KM,JP,JM,IP,IM,FJUM,FJUP,
20                      !$omp& DIVUH,CBC)
21                            DO 10 K=1,N3M
22                               KP=KPA(K)
```

```
23                                KM=KMA(K)
24                               DO 10 J=1,N2M
25                                JP=J+1
26                                JM=J-1
27                                FJUM=FJMU(J)
28                                FJUP=FJPA(J)
30                               DO 10 I=1,N1M
31          25                    IP=IPA(I)
32           1                   IM=IMA(I)
34          43                   DIVUH=(UH(IP,J,K,1)-UH(I,J,K,1))*DX1+(FJUP
35                              > *UH(I,JP,K,2)-FJUM*UH(I,J,K,2))/DR(J)/RM(J)
36                              >     +(UH(I,J,KP,3)-UH(I,J,K,3))*DX3/RM(J)**2.0
38          98                   CBC=(1.-FJUM)*UBC(I,K,2,1)/DR(J)/RM(J)
39                              >          -(1.-FJUP)*UBC(I,K,2,2)/DR(J)/RM(J)
41          24                   RDP(I,J,K)=(DIVUH-CBC)/DT
42                         10    CONTINUE
```

① parallel do 지시어를 삽입해 병렬화 하였다.

4.11 getdp.f

real 데이터 RDP(I,J,K)를 J에 대해 반복하면서 2차원 FFT 수행하고 그 결과로 나오는 complex 데이터 coef(I,K)의 real 부분을 FRDP(I,J,2K-1)에 imaginary 부분을 FRDP(I,J,2K)에 할당하는 과정이다.

< Original 코드 >

```
45                        C---   FOR IMSL(Kelvin)
46                               CALL DFFTCI(N1M,WFF1)
47                               CALL DFFTCI(N3M,WFF2)
49                               DO 10 J=1,N2M
50                               DO 1 K=1,N3M
51                               DO 1 I=1,N1M
52          60                   XR(I,K)=CMPLX(RDP(I,J,K),0.0)
53                             1 CONTINUE
55                               CALL DF2T2D(N1M,N3M,XR,N1M,COEF,N1M,
                                * WFF1,WFF2,CWK,CPY)
62                               RATIO=1.0/DBLE(N1M*N3M) ! DEC ONLY USE
63                        C      REDUCE FOURIER COMPONENT N3M->N3MH
64                               DO 11 K=1,N3MH
65                               KR=2*K-1
```

```
66                          KI=2*K
67                          DO 11 I=1,N1M
70          34              FRDP(I,J,KR)=DBLE(COEF(I,K))          !NWT
72          42              FRDP(I,J,KI)=IMAG(COEF(I,K))          !NWT
73                      11 CONTINUE
74                      10 CONTINUE
```

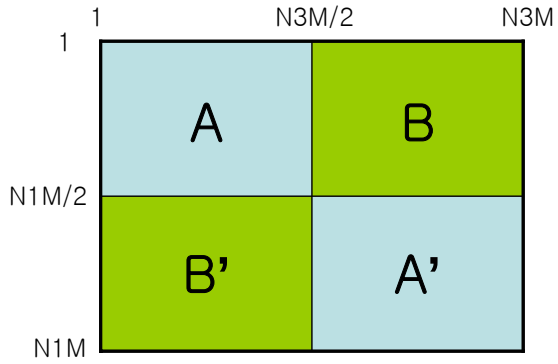
< 최적화/병렬화 코드 >

```
38                          isign=1
39                          scale=1.d0
40                          call drcft2(1,XR,M1M+2,XR,M1M/2+1,M1M,M3M,isign,
41                         & scale,aux1,naux1,aux2,naux2)
42                  !$omp   parallel do private(XR,aux2)
43                          DO 10 J=1,N2M
44                            DO K=1,N3M
45                            DO I=1,N1M
46          46                  XR(I,K)=RDP(I,J,K)
47                            ENDDO
48                            ENDDO
50                          call drcft2(0,XR,M1M+2,XR,M1M/2+1,M1M,M3M,isign,
51                         & scale,aux1,naux1,aux2,naux2)
53                            DO K=1,N3M
54                            DO I=1,N1MH
55          22                  FRDP(1,I,J,K)=XR(2*I-1,K)
56          32                  FRDP(2,I,J,K)=XR(2*I   ,K)
57                            ENDDO
58                            ENDDO
59                      10 CONTINUE
```

① IMSL에는 2차원 real to complex FFT 루틴이 없어 real 데이터 RDP를 real 부분으로 가지고 imaginary 부분이 0인 complex 데이터 XR을 만든 후 complex to complex 2D FFT(DF2T2D)를 수행하고 있다.

② 2차원 real to complex FFT의 경우 다음과 같은 대칭성을 가진다. 그림은 real to complex FFT의 결과로 나오는 N1M x N3M 크기의 2차원 complex 배열을 나타낸 것으로 영역 A와 A', 영역 B와 B'이 동일한 값을 가지게 됨을 보여 주고 있다. IMSL의 DF2T2D 루틴은 이 모든 complex 데이터를 계산하지만 ESSL의 drcft2 루틴은 모든

complex 데이터를 계산하지 않고 대칭성을 고려해 A와 B 영역만을 계산해 결과로 내어 놓는다.



< 그림 III.3 2차원 real to complex FFT의 대칭성 >

③ IMSL루틴 DF2T2D는 complex to complex 이므로 이 모든 FFT 계산을 수행한다. Original 코드를 보면 FFT 계산의 결과로 나오는 크기 N1M x N3M의 complex 데이터 COEF를 받아 그것의 1/2(K = 1, N3M/2; 위 그림에서 영역 A, B'에 해당)을 FRDP에 할당해 사용하고 있다.

④ 최적화 코드에서는 IBM 시스템에 최적화된 라이브러리인 ESSL의 2차원 real to complex FFT 루틴(drcft2)를 사용하였다. Drcft2 루틴은 대칭성을 고려해 전체 2차원 데이터의 1/2만을 계산한다(그림에서 A와 B영역의 데이터를 결과로 내놓는다). 이로 인해 보다 효율적이고 빠른 real to complex 계산이 가능하다.

⑤ Original 코드에서는 RDP(I,J,K)의 FFT coefficients를 real 부분은 FRDP(I,J,2K-1)에 imaginary 부분은 FRDP(I,J,2K)에 번갈아 가며 저장하고 있다(여기서 K가 1부터 N3M/2 까지 반복된다). 이렇게 저장된 데이터는 다음 단계에서 real 부분 따로, imaginary 부분 따로 접근, 계산되므로 불연속 데이터 접근(stride가 2)이 발생하게 된다. 이 부분에서 발생하는 메모리 불연속성을 없애주기 위해 최적화 코드에서는 FRDP를 4차원으로 정의하였고 FFT coefficients의 real 부분을 FRDP(1,I,J,K)에 imaginary 부분을

FRDP(2,I,J,K)에 각각 저장하였다(여기서는 I가 1부터 N1M/2까지 반복된다).

⑥ Parallel do 지시어를 이용해 J 루프를 병렬화하였다.

< Original 코드: getdp.f 계속>

```
 78                    C---- REAL PART CALCULATION
 79                          DO 110 K=1,N3MH
 80                          KR=2*K-1
 81                          DO 110 I=1,N1M
 82          1               DO 120 J=1,N2M
 83          2               CMJ(J)=PMJ(J)
 84         51               CCJ(J)=PCJ(J)-AK3(K)/RM(J)**2.0-AK1(I)
 85         75               CPJ(J)=PPJ(J)
 86         46               CFJ(J)=FRDP(I,J,KR)
 87                          IF(I.EQ.1.AND.K.EQ.1) THEN
 88                          CMJ(1)=0.0
 89                          CCJ(1)=1.0
 90                          CPJ(1)=0.0
 91                          CFJ(1)=0.0
 92                          ENDIF
 93                      120 CONTINUE
 94                          CALL TDMAP(CMJ,CCJ,CPJ,CFJ,N2M,CFJ)
 95          1               DO 130 J=1,N2M
 96         21               FDP(I,J,KR)=CFJ(J)
 97                      130 CONTINUE
 98                      110 CONTINUE
100                    C---- IMAG PART CALCULATION
101                          DO 140 K=1,N3MH
102                          KI=2*K
103                          DO 140 I=1,N1M
104          2               DO 150 J=1,N2M
105          4               CMJ(J)=PMJ(J)
106         93               CCJ(J)=PCJ(J)-AK3(K)/RM(J)**2.0-AK1(I)
107          6               CPJ(J)=PPJ(J)
108         82               CFJ(J)=FRDP(I,J,KI)
110                          IF(((I.EQ.1).AND.(K.EQ.1)).OR.((I.EQ.1)
                            >.AND.(K.EQ.N3MH)).OR.((I.EQ.N1MH).AND.(K.EQ.1))
111                         >.OR.((I.EQ.N1MH).AND.(K.EQ.N3MH))) THEN
112                          CMJ(J)=0.0
113                          CCJ(J)=1.0
114                          CPJ(J)=0.0
115                          CFJ(J)=0.0
```

```
116                         ENDIF
117                   150 CONTINUE
118                         CALL TDMAP(CMJ,CCJ,CPJ,CFJ,N2M,CFJ)
119                         DO 160 J=1,N2M
120               12        FDP(I,J,KI)=CFJ(J)
121                   160 CONTINUE
122                   140 CONTINUE
```

< 최적화/병렬화 코드: getdp.f 계속 >

```
 63                   C---- REAL PART CALCULATION
 65                   !$omp   parallel do private(KK,I,CCJ,CPJ,CFJ,CCJ2)
 66                         DO K=1,N3MH
 67                         IF(K.EQ.1) THEN
 68                            I=1
 69                            CCJ(1)=1.0
 70                            CPJ(1)=0.0
 71                            CFJ(1)=0.0
 72                            FRDP(1,I,1,K)=0.0
 73                          DO J=2,N2M
 74                            CMJ(J)=PMJ(J)
 75                            CCJ(J)=PCJ(J)-AK3(K)/RM(J)**2.0-AK1(I)
 76                            CPJ(J)=PPJ(J)
 77                          ENDDO
 78                          CALL TDMAP(CMJ,CCJ,CPJ,CFJ,FRDP(1,I,1,K),
                          >    N1MH,N2M)
 80                          DO J=1,N2M
 81                          DO I=2,N1MH
 82                            CCJ2(I,J)=PCJ(J)-AK3(K)/RM(J)**2.0-AK1(I)
 83                          ENDDO
 84                          ENDDO
 85                          CALL TDMAP20(PMJ,CCJ2(2,1),PPJ,FRDP(1,2,1,K),
                          >   N1MH,N2M,N1MH-1)
 87                        ELSEIF(K.GT.1.AND. K.LT. N3MH) THEN
 88                            KK=N3M-K+2
 89               1          DO J=1,N2M
 90                          DO I=1,N1MH
 91               3            CCJ2(I,J)=PCJ(J)-AK3(K)/RM(J)**2.0-AK1(I)
 92                          ENDDO
 93                          ENDDO
 94                          CALL TDMAP2(PMJ,CCJ2,PPJ,FRDP(1,1,1,K),
 95                         & FRDP(1,1,1,KK),N2M,N1MH)
 97                         ELSEIF(K.EQ.N3MH) THEN
 98                         DO J=1,N2M
 99                         DO I=1,N1MH
```

```
100                         CCJ2(I,J)=PCJ(J)-AK3(K)/RM(J)**2.0-AK1(I)
101                       ENDDO
102                       ENDDO
103                       CALL TDMAP20(PMJ,CCJ2,PPJ,FRDP(1,1,1,K),
                        >  N1MH,N2M,N1MH)
105                       ENDIF
106                       ENDDO
```

⑦ Original 코드에서는 real 부분 계산과 imaginary 부분에 대한 계산이 분리돼 각각 K = 1, N3MH를 반복하고 있다. 최적화 코드에서는 real 부분과 imaginary 부분을 합쳐 하나의 K루프로 만들고 이를 병렬화하고 있다.

⑧ Original 코드에서는 K 루프가 반복되며, real 부분에서 FRDP(I,J,2K-1)의 데이터, imaginary 부분에서는 FRDP(I,J,2K)의 데이터에 접근하므로 각각 불연속 접근이 이뤄지고 있다. 최적화 코드에서는 real 부분을 FRDP(1,I,J,K)에 imaginary 부분을 FRDP(2,I,J,K)에 저장해 real 부분과 imaginary 부분을 한꺼번에 처리함으로써 데이터 접근이 연속적으로 이뤄지도록 하고 있다.

⑨ Original 코드에서는 각 K, I에 대해 J 루프를 반복시켜 1차원 배열 CMJ, CCJ, CPJ를 구성하고 이에 대해 서브루틴 TDMAP를 호출해 1차원 배열에 대한 계산을 수행한다. 최적화 코드에서는 1차원 배열 PMJ, PPJ를 불필요하게 다른 1차원 배열 CMJ, CPJ로 할당하는 부분을 없애고, 2차원 배열 CCJ2를 정의해 메모리 접근과 사용량에 있어 효율성을 높이고 있다. 이 과정에서 IF 구문을 J 루프 밖으로 빼내 코드의 흐름을 좋게 하고 있다. 그리고, 1차원 배열을 처리하는 original 코드의 서브루틴 TDMAP 대신 2차원 배열을 처리하고 FRDP에 대한 real 부분과 imaginary 부분을 한꺼번에 처리하는 서브루틴 TDMAP2와 TDMAP20를 새롭게 정의해 사용하였다.

< TDMAP - Original 코드 >

```
2                       SUBROUTINE TDMAP(A,B,C,D,N,X)
6                       REAL GAM(M2),A(M2),B(M2),C(M2),D(M2),X(M2)
```

```
 8                      BET=1.0/B(1)
 9                      X(1)=D(1)*BET
10          4           DO 11 J=2,N
11        327           GAM(J)=C(J-1)*BET
12         13           BET=1.0/(B(J)-A(J)*GAM(J))
13         64           X(J)=(D(J)-A(J)*X(J-1))*BET
14                    11 CONTINUE
15                      DO 12 J=N-1,1,-1
16         53           X(J)=X(J)-GAM(J+1)*X(J+1)
17                    12 CONTINUE
```

< TDMAP – 최적화/병렬화 코드 >

```
20                      SUBROUTINE TDMAP(A,B,C,D,X,LDX,N)
22                      REAL GAM(M2),A(M2),B(M2),C(M2),D(M2),
                     >  X(2,LDX,M2)
24                         J=1
25                         BET=1.0/B(J)
26                         D(J)=X(1,1,J)*BET
27                      DO J=2,N
28                         GAM(J)=C(J-1)*BET
29                         BET=1.0/(B(J)-A(J)*GAM(J))
30                         D(J)=(X(1,1,J)-A(J)*D(J-1))*BET
31                      ENDDO
32                         J=N
33                         X(1,1,J)=D(J)
34                      DO J=N-1,1,-1
35                         X(1,1,J)=D(J)-GAM(J+1)*X(1,1,J+1)
36                      ENDDO

42                      SUBROUTINE TDMAP2(A,B,C,X,Y,N,N1)
44                      REAL GAM(M2),A(M2),B(N1,M2),C(M2),X(2,N1,M2),
                     >  Y(2,N1,M2)
46        126           DO I=1,N1
48                         J=1
49                         BET=1.0/B(I,J)
50         17              X(1,I,J)=X(1,I,J)*BET
51          6              X(2,I,J)=X(2,I,J)*BET
52          6              Y(1,I,J)=Y(1,I,J)*BET
53                         Y(2,I,J)=Y(2,I,J)*BET
54                      DO J=2,N
55         18              GAM(J)=C(J-1)*BET
56                         BET=1.0/(B(I,J)-A(J)*GAM(J))
57          1              X(1,I,J)=(X(1,I,J)-A(J)*X(1,I,J-1))*BET
58          1              X(2,I,J)=(X(2,I,J)-A(J)*X(2,I,J-1))*BET
```

```
59                          Y(1,I,J)=(Y(1,I,J)-A(J)*Y(1,I,J-1))*BET
60          2               Y(2,I,J)=(Y(2,I,J)-A(J)*Y(2,I,J-1))*BET
61                        ENDDO
62                        DO J=N-1,1,-1
63         17               X(1,I,J)=X(1,I,J)-GAM(J+1)*X(1,I,J+1)
64          1               X(2,I,J)=X(2,I,J)-GAM(J+1)*X(2,I,J+1)
65         23               Y(1,I,J)=Y(1,I,J)-GAM(J+1)*Y(1,I,J+1)
66         12               Y(2,I,J)=Y(2,I,J)-GAM(J+1)*Y(2,I,J+1)
67                        ENDDO
69                        ENDDO

74                        SUBROUTINE TDMAP20(A,B,C,X,LDX,N,N1)
76                        REAL GAM(M2),A(M2),B(LDX,M2),C(M2),
                         > X(2,LDX,M2)
78                        DO I=1,N1
80                          J=1
81                          BET=1.0/B(I,J)
82                          X(1,I,J)=X(1,I,J)*BET
83          1               X(2,I,J)=X(2,I,J)*BET
84                        DO J=2,N
85                          GAM(J)=C(J-1)*BET
86                          BET=1.0/(B(I,J)-A(J)*GAM(J))
87          1               X(1,I,J)=(X(1,I,J)-A(J)*X(1,I,J-1))*BET
88          1               X(2,I,J)=(X(2,I,J)-A(J)*X(2,I,J-1))*BET
89                        ENDDO
90                        DO J=N-1,1,-1
91                          X(1,I,J)=X(1,I,J)-GAM(J+1)*X(1,I,J+1)
92                          X(2,I,J)=X(2,I,J)-GAM(J+1)*X(2,I,J+1)
93                        ENDDO
95                        ENDDO
```

⑩ original 코드에서 1차원 데이터를 받아 처리하는 TDMAP와 달리 최적화 코드의 TDMAP는 더미변수 X가 3차원으로 정의돼 각 K에 대한 FRDP(1,1,1,K)를 직접 받아 처리한다.

⑪ 최적화 코드의 TDMAP2는 각 K에 대해 3차원 데이터 FRDP(1,1,1,1:N3MH)와 FRDP(1,1,1,N3MH+1:N3M)을 받아 처리한다. Original 코드에서는 없는 부분인 N3MH+1부터 N3M까지 데이터가 최적화 코드에 있는 것은 앞서 FFT를 수행한 후 original 코드에서는 K가 1부터 N3MH까지의 데이터를 사용하도록 한 반면,

최적화 코드의 ESSL 루틴에서는 1부터 N3M까지의 데이터를 모두 받고 대신 I를 1부터 N1MH까지만 사용하도록 데이터를 저장하였기 때문이다. 아울러 real과 imaginary 부분이 FRDP(1,…), FRDP(2,…)로 각각 전달되어 한꺼번에 처리되도록 하고 있다.

⑫ TDMAP20는 TDMAP2와 동일하며 단지 FRDP(1,1,1,1:N3MH)만 처리되도록 하는 점이 다를 뿐이다.

< Original 코드: getdp.f 계속 >

```
124                 C       DO THE INVERSE FFT.
125                         DO 20 J=1,N2M
126                         DO 21 K=1,N3MH
127                         KR=2*K-1
128                         KI=2*K
129                         DO 21 I=1,N1M
130           41            COEF(I,K)=CMPLX(FDP(I,J,KR),FDP(I,J,KI))
131                      21 CONTINUE
133                         DO 22 K=N3MH+1,N3M
134                         KK=N3M-K+2
135                         DO 22 I=1,N1M
136            7            II=N1M-I+2
137            9            IF(I.EQ.1) II=1
139           15            COEF(I,K)=CONJG(COEF(II,KK))
140                      22 CONTINUE
142                         CALL DF2T2B(N1M,N3M,XR,N1M,COEF,N1M,WFF1,
                    >       WFF2,CWK,CPY)
148                         DO 23 K=1,N3M
149                         DO 23 I=1,N1M
150          138            DP(I,J,K)=DBLE(XR(I,K))*RATIO
151            9         23 CONTINUE
152                      20 CONTINUE
```

< 최적화/병렬화 코드: getdp.f 계속 >

```
107                 C       DO THE INVERSE FFT.
108                         isign=-1
109                         scale=1.0/FLOAT(N1M*N3M)
110                         call dcrft2(1,XR,M1M/2+1,XR,M1M+2,M1M,M3M,isign,
111                        & scale,aux1,naux1,aux2,naux2)
112                 !$omp    parallel do private(XR,aux2)
113                         DO 20 J=1,N2M
114                          DO K=1,N3M
```

```
115                         DO I=1,N1MH
116          30               XR(2*I-1,K)=FDP(1,I,J,K)
117           5               XR(2*I   ,K)=FDP(2,I,J,K)
118                         ENDDO
119                         ENDDO
120                  c====>
121                       call dcrft2(0,XR,M1M/2+1,XR,M1M+2,M1M,M3M,isign,
122                      & scale,aux1,naux1,aux2,naux2)
123                  c====>
124                         DO K=1,N3M
125                         DO I=1,N1M
126          67               DP(I,J,K)=XR(I,K)
127                         ENDDO
128           2             ENDDO
129                     20 CONTINUE
```

⑬ Backward Fourier Transform이 수행되는 부분으로 앞서의 FFT를 수행한 부분과 동일한 최적화 병렬화 과정이 진행되었다.

⑭ Original 코드에서 IMSL 루틴 DF2T2B를 호출해 Complex to complex 변환을 시도하기 위해 1부터 N3MH까지 정의된 complex 2차원 데이터 COEF를 정의하고, N3MH+1부터 N3M까지는 이의 complex conjugate를 이용해 전체 2차원 complex data COEF를 정의하였다. Complex to complex 변환의 결과로 나온 complex 데이터 XR의 real 부분만을 취해 real 데이터 DP를 최종적으로 얻고 있다. 최적화 코드에서는 이미 1부터 N3M까지 정의된 데이터를 이용해 2차원 데이터 XR을 할당하고 complex to real 변환을 수행하는 dcrft2를 호출해 real 데이터 XR을 얻는다.

⑮ parallel do 지시어를 이용해 바깥쪽 루프 J에 대해 병렬화를 수행하였다.

4.12 energy.f

< Original 코드 >

```
22                          DO 10 K=1,N3M
```

```
 25                      DO 10 J=1,N2M
 30                      DO 10 I=1,N1M
 34          3           U1KP=0.5*(U(I,J,KP,1)+U(IP,J,KP,1))
 43         15           U1JP=0.5*(U(I,JP,K,1)+U(IP,JP,K,1))
 52         15           U3JP=0.5*(U(I,JP,K,3)+U(I,JP,KP,3))
 60                      DV3DX2=(U32-U31)/RM(J)/DR(J)
 67        161           VOR(I,J,K,1)=DV3DX2-DV2DX3
 70                   10 CONTINUE
 75                      DO 20 J=1,N2M
107          1          U1U2U3=0.0
108                     DO 30 K=1,N3M
110                     DO 30 I=1,N1M
112        103          V1=(U(I,J,K,1)+U(IP,J,K,1))*0.5
146                  30 CONTINUE
147                     VM(J,1)=V1M*NXZ
178          1       20 CONTINUE
181                     DO 40 J=1,N2M
190                     DO 50 K=1,N3M
192          1          DO 50 I=1,N1M
194         66          U_JP=(U(I,JP,K,1)+U(IP,JP,K,1))*0.5
222                  50 CONTINUE
223          1          DUDR(J)=DU_DR*NXZ
227                  40 CONTINUE
233                     DO 60 J=1,N2M
234                     P_DUDX=0.0
248                     DO 70 K=1,N3M
251                     DO 70 I=1,N1M
256          9          U_KP=(U(I,J,KP,1)+U(IP,J,KP,1))*0.5
293          2       70 CONTINUE
294                     PDUDX(J)=P_DUDX*NXZ
308                  60 CONTINUE
312                     DO 80 J=1,N2M
317                     P_DUDR=0.0
326                     DO 90 K=1,N3M
328                     DO 90 I=1,N1M
331         81          U_JP=(U(I,JP,K,1)+U(IP,JP,K,1))*0.5
368          1       90 CONTINUE
369                     PDUDR(J)=P_DUDR*NXZ
378                  80 CONTINUE
381                     DO 100 J=1,N2M
387                     DO 110 K=1,N3M
390                     DO 110 I=1,N1M
393         16          U_KP=(U(I,J,KP,1)+U(IP,J,KP,1))*0.5
413          1      110 CONTINUE
414                     DUDT2(J)=DU_DT2*NXZ
```

```
418                     100 CONTINUE
```

< 최적화/병렬화 코드 >

```
 48                !$omp parallel do private(KP,KM,JP,JM,IP,IM,FJUM,FJUP,
 49                !$omp& U1KP,U1KM,U2KP,U2KM,U2IP,U2IM,U3IP,U3IM,
 50                !$omp& U1JP,U1JC,U1JM,U12,U11, DUW_DT2,
 51                !$omp& U3JP,U3JC,U3JM,U32,U31, DVW_DT2,
 52                !$omp& DV3DX2,DV2DX3,DV3DX1,DV1DX3,DV2DX1,DV1DX2,
 53                !$omp& V1,V2,V3, PMMMS,PUMS, DW_DT2,
 54                !$omp& U_JP,U_JC,U_JM,W_JP,W_JC,W_JM,P_JP,P_JC,P_JM,
 55                !$omp& U1,U2,W1,W2,P1,P2, VOR1MS, PVMS,
 56                !$omp& U_KP,U_KC,U_KM,V_IP,V_IC,V_IM,V_KP,V_KC,V_KM,
 57                !$omp& W_IP,W_IC,W_IM,U_C,V,W,P2X,P1X,P2Z,P1Z,
 58                !$omp& V1M,V2M,V3M,PMM,VM1M,VM2M,VM3M,
 59                !$omp& U11MS,U22MS,U33MS,U12MS,U13MS,U23MS,
 60                !$omp& VOR2MS,VOR3MS,U1SK,U2SK,U3SK,U1FL,
 61                !$omp& U11U2,U2U33,U1U22,U1U33,U22U3,U1U2U3,
 62                !$omp& DU_DR,DV_DR,DW_DR,DP_DR, DUV_DX2,
 63                !$omp& P_DUDX,P_DWDT,P_DVDX,DU_DX2,DV_DX2,
 64                !$omp& W_DPDX,U_DPDT,V_DPDT,DUW_DX2,DVW_DX2,
 65                !$omp& P_DUDR,P_DVDR,DU_DR2,DV_DR2,DW_DR2,
 66                !$omp& DU_DT2,DV_DT2,DW_DX2,DUV_DT2,DUV_DR2,
                   !$omp& DUW_DR2,DVW_DR2, U2FL,U3FL, W_DPDR,)
 67                      DO J=1,N2M
 72                      DO 10 K=1,N3M
 75                      DO 10 I=1,N1M
 79           175          U1KP=0.5*(U(I,J,KP,1)+U(IP,J,KP,1))
115             3       10 CONTINUE
122                      V1M=0.0
153                      DO 30 K=1,N3M
155                      DO 30 I=1,N1M
157            89        V1=(U(I,J,K,1)+U(IP,J,K,1))*0.5
191             2       30 CONTINUE
192                      VM(J,1)=V1M*NXZ
223                   20 CONTINUE
231                      DU_DR=0.0
235             2        DO 50 K=1,N3M
237                      DO 50 I=1,N1M
239            14        U_JP=(U(I,JP,K,1)+U(IP,JP,K,1))*0.5
267                   50 CONTINUE
268                      DUDR(J)=DU_DR*NXZ
272                   40 CONTINUE
279                      P_DUDX=0.0
293                      DO 70 K=1,N3M
```

```
296                         DO 70 I=1,N1M
299                         U1=U(I ,J,K,1)
338          4           70 CONTINUE
339                         PDUDX(J)=P_DUDX*NXZ
353                      60 CONTINUE
362                         P_DUDR=0.0
371                         DO 90 K=1,N3M
373                         DO 90 I=1,N1M
376          94             U_JP=(U(I,JP,K,1)+U(IP,JP,K,1))*0.5
413                      90 CONTINUE
414                         PDUDR(J)=P_DUDR*NXZ
423                      80 CONTINUE
427                         DU_DT2=0.0
432                         DO 110 K=1,N3M
435                         DO 110 I=1,N1M
438          27             U_KP=(U(I,J,KP,1)+U(IP,J,KP,1))*0.5
458                     110 CONTINUE
459                         DUDT2(J)=DU_DT2*NXZ
463                     100 CONTINUE
464                         ENDDO
```

① 공통적으로 반복되는 여러 개의 J루프를 하나로 합치고, 합쳐진 루프를 parallel do 지시어를 이용해 병렬화 하였다.

5. inflow summary

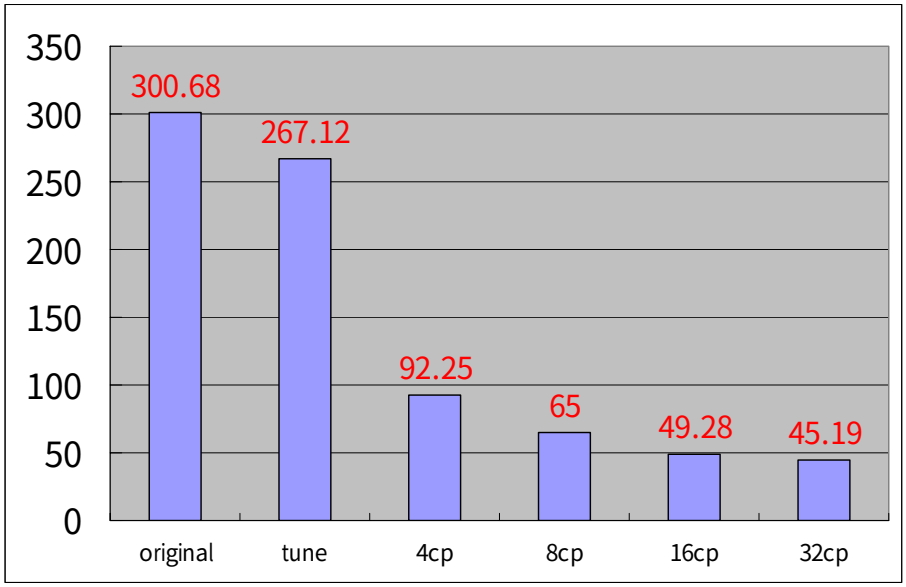
<표 III. 1 inflow first step >

		Parallel Speed up
Original (Serial)	300.68s	
Tuned 1cp	267.12s	1.12
4 cp	92.25s	3.26
8 cp	65.00s	4.62
16 cp	49.28s	6.10
32 cp	45.19s	6.65

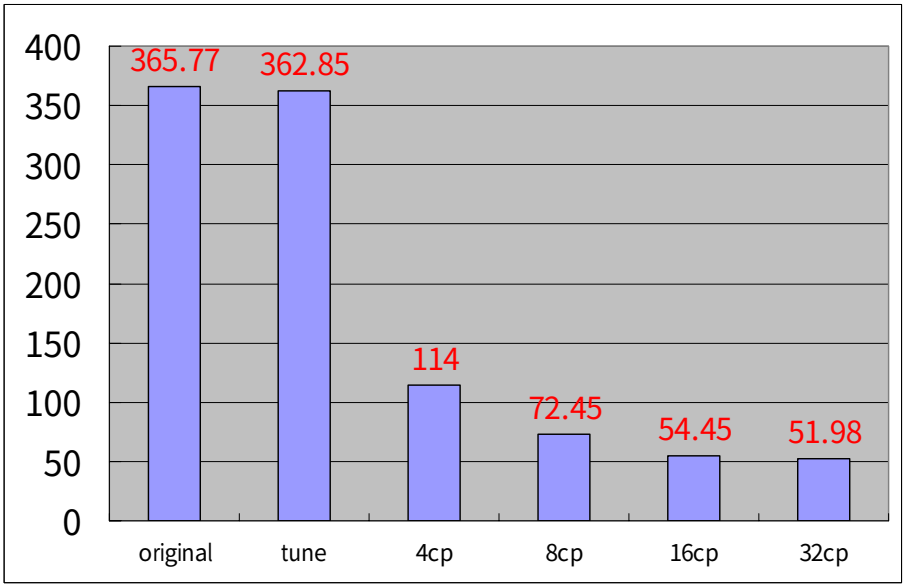
< 표 III.2 inflow second step >

		Parallel Speed up
Original (Serial)	365.77s	
Tuned 1cp	362.85s	1.01
4 cp	114.00s	3.2
8 cp	72.45s	5.04
16 cp	54.45s	6.71
32 cp	51.98s	7.03

Inflow 코드는 실행속도가 메인 코드에 비해 그리 큰 시간을 차지하지 않는 코드이고, 코드의 최적화가 이미 한 번 진행된 상태이다. 따라서 코드에서 진행된 최적화를 통해 얻어진 성능이득은 그다지 크지 않으며 주로 병렬화를 통해서 성능이득을 얻고 있다. 루프의 fusion과 불필요한 데이터 할당을 없애는 등의 간단한 최적화를 수행하였고 특히 real to complex, complex to real 만으로 충분한 FFT 변환을 IMSL의 complex to complex FFT 루틴을 사용함으로써 발생하는 original 코드의 비효율성을 ESSL 루틴을 사용함으로써 주요한 성능향상 효과를 얻고 있다.

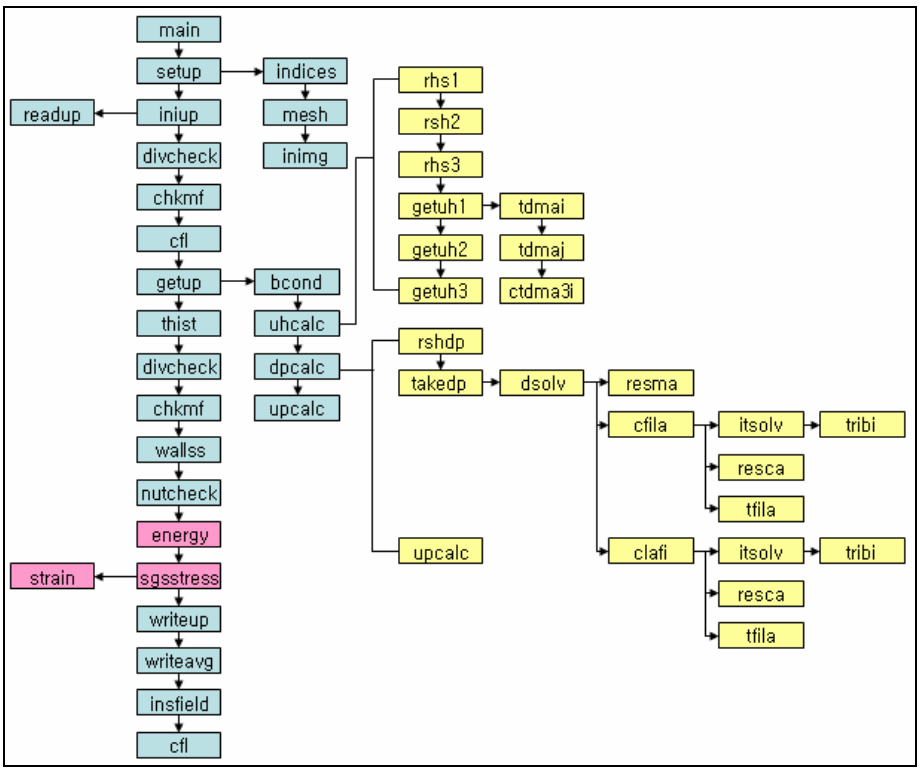


< 그림 III.4 first step 에서 최적화 및 병렬화 성능 비교 >



< 그림 III.5 second step에서 최적화 및 병렬화 성능 비교 >

6. main.x 코드 최적화/병렬화 비교 및 분석



<그림 III.6 main.x Flowchart >

위 그림은 프로그램의 주 실행부분인 main.x의 subroutine을 Flowchart로 나타낸 것이다. 전체 흐름 중 주된 계산이 이루어지는 곳은 energy 서브루틴 부분이며, 이와 함께 takedp 서브루틴들도 전체 코드에서 많은 시간을 차지하고 있다. 최적화를 통해 비교적 많은 성능향상을 얻은 곳은 takedp 부분으로, original 코드에서 사용하는 IMSL FFT 루틴 대신 ESSL의 FFT루틴을 사용하고 있다. 두 번 실행되는 동안, 첫 번째에서는 energy 부분을 계산하지 않고, 두 번째 실행 시 붉은색으로 표시된 energy 루틴을 계산하게 된다.

6.1 OpenMPcut.f

< 최적화/병렬화 코드 >

```
 4                          nn=kend-kstart+1
 5                          nblk=nn/nthds
 6                          ndelta=mod(nn,nthds)
 7                          if(ndelta.gt.0) then
 8                              k=1
 9                              ks(1,1)=1
10                              ks(2,1)=ks(1,1)+nblk
11                          do k=2,ndelta
12                              ks(1,k)=ks(1,k-1)+nblk+1
13                              ks(2,k)=ks(1,k)+nblk
14                          enddo
15                              k=ndelta+1
16                              ks(1,k)=ks(1,k-1)+nblk+1
17                              ks(2,k)=ks(1,k)+nblk-1
18                          do k=ndelta+2,nthds
19                              ks(1,k)=ks(1,k-1)+nblk
20                              ks(2,k)=ks(1,k)+nblk-1
21                          enddo
22                          else
23                              ks(1,1)=1
24                              ks(2,1)=ks(1,1)+nblk-1
25                          do k=2,nthds
26                              ks(1,k)=ks(1,k-1)+nblk
27                              ks(2,k)=ks(1,k)+nblk-1
28                          enddo
29                          endif
30                          do k=1,nthds
31                              ks(1,k)=ks(1,k)+kstart-1
32                              ks(2,k)=ks(2,k)+kstart-1
33                          enddo
```

① 병렬화를 위한 루틴이며, kstart에서 kend까지의 값을 주어진 개수(nthds)만큼의 스레드에 블록분할해 준다. nthds개의 스레드 중 p(0 ≤ p ≤ nthds-1) 스레드가 할당 받게 되는 데이터 영역은 ks(1,p)부터 ks(2,p)까지가 된다.

kend = 100(kstart=1, kend=100), nthds = 2 일 때

ks(1,1) = 1 , ks(2,1) = 50

ks(1,2) = 51 , ks(2,2) = 100

kend = 100(kstart=1, kend=100), nthds = 3 일 때

ks(1, 1) = 1 , ks(2, 1) = 34 (34)

ks(1, 2) = 35 , ks(2, 2) = 67 (33)

ks(1, 3)= 68 , ks(2, 3) = 100 (33)

kend = 100(kstart=1, kend=100), nthds = 6 일 때

ks(1, 1) = 1 , ks(2, 1) = 17 (17)

ks(1. 2) = 18 , ks(2, 2) = 34 (17)

ks(1, 3)= 35 , ks(2, 3) = 51 (17)

ks(1, 4) = 52 , ks(2, 4) = 68 (17)

ks(1. 5) = 69 , ks(2, 5) = 84 (16)

ks(1, 6)= 85 , ks(2, 6) = 100 (16)

6.2 cfl.f

<Original 코드>

```
9                       REAL U(0:M1,0:M2,0:M3,3)
10                       CFLM=0.0
11                       IMAX=0
12                       JMAX=0
13                       KMAX=0
14
15                       DO 10 K=1,N3M
16                       KP=KPA(K)
17                       DO 10 J=1,N2M
18                       II=IIW(J)
19                       NN=NNW(J)
20                       JP=J+1
21                       DO 10 I=II,NN
22                       IP=I+1
23
24            469        CFLL= ABS(U(I,J,K,1)+U(IP,J,K,1))*0.5/DX(I)
25                      >    +ABS(U(I,J,K,2)+U(I,JP,K,2))*0.5/DR(J)/RM(J)
26                      >    +ABS(U(I,J,K,3)+U(I,J,KP,3))*0.5*DX3/RM(J)**2.0
27                       IF(CFLL.GT.CFLM) THEN
28                       CFLM=CFLL
29                       IMAX=I
30                       JMAX=J
```

```
31                          KMAX=K
32                          ENDIF
33          75      10      CONTINUE
```

<최적화/병렬화 코드>

```
14                  !$omp parallel do private(ks,ke,KP,JP,IP,II,NN,CFLL)
15                  do jj=0,nthds-1
16                     ks=kse(1,jj)
17                     ke=kse(2,jj)
18                  CFLM(jj)=0.0
19                  IMAX(jj)=0
20                  JMAX(jj)=0
21                  KMAX(jj)=0
22
23                  DO 10 K=ks,ke
24                     KP=KPA(K)
25                  DO 10 J=1,N2M
26                     II=IIW(J)
27                     NN=NNW(J)
28                     JP=J+1
29                  DO 10 I=II,NN
30                     IP=I+1
31
32          456        CFLL= ABS(U(I,J,K,1)+U(IP,J,K,1))*0.5/DX(I)
33                    >     +ABS(U(I,J,K,2)+U(I,JP,K,2))*0.5/DR(J)/RM(J)
34                    >     +ABS(U(I,J,K,3)+U(I,J,KP,3))*0.5*DX3/RM(J)**2.0
35                     IF(CFLL.GT.CFLM(jj)) THEN
36                      CFLM(jj)=CFLL
37                      IMAX(jj)=I
38                      JMAX(jj)=J
39                      KMAX(jj)=K
40                     ENDIF
41          77      10  CONTINUE
42                     ENDDO
43
44                        jj=0
45                        i1=jj
46                      CFLM1=CFLM(jj)
47                     do jj=1,nthds-1
48                     CFLL=CFLM(jj)
49                     IF(CFLL.GT.CFLM1) THEN
50                      CFLM1=CFLM(jj)
51                      i1=jj
52                     ENDIF
```

```
53                          ENDDO
54                           IMAX1=IMAX(i1)
55                           JMAX1=JMAX(i1)
56                           KMAX1=KMAX(i1)
```

⑦ OpenMP 지시어를 사용하여 병렬화를 하였다.

⑧ 병렬 영역내에서 CFLL의 최대값 CFLM을 구할 때 I, J, K 값 IMAX, JMAX, KMAX를 병렬로 찾는데 문제가 있다. 각 스레드 마다 CFLL의 로컬 최대값을 구하고 그때의 IMAX, JMAX, KMAX를 스레드 마다 따로 저장시켜 병렬 영역 밖으로 빼내기 위해 스레드 수만큼의 크기를 가지는 배열을 이용하고 있다.

⑨ 사용하는 스레드 개수(nthds)에 의존해 K 루프를 명시적으로 분할 하였다. 명시적인 분할을 위해 호출해 사용하는 루틴이 openmpcut이다.

⑩ 각 스레드가 계산한 로컬 데이터 CFLM(n), IMAX(n), JMAX(n), KMAX(n)을 비교해 글로벌 최대값 데이터를 얻기 위해 병렬 영역 밖에서 다시 한 번 비교 과정을 거친다.

6.3 bcond.f

<Original 코드>

```
12                      C INITIALIZE THE INTERMEDIATE VELOCITY AND PRESSURE
13                            DO 10 NV=1,3
14                            DO 10 K=1,N3
15                            DO 10 J=0,N2
16                            DO 10 I=0,N1
17          283         10     UH(I,J,K,NV)=0.0
18                            DO 20 K=1,N3
19                            DO 20 J=1,N2
20                            DO 20 I=1,N1
21           92         20     DP(I,J,K)=0.0
22                            DO 30 K=1,N3
23                            DO 30 J=0,N2
24                            DO 30 I=0,N1
25           96         30     SGSVIS(I,J,K)=0.0
```

<최적화/병렬화 코드>

```
13                      C INITIALIZE THE INTERMEDIATE VELOCITY AND PRESSURE
14                      !$omp parallel do
15                            DO K=1,N3
16                             DO J=0,N2
17                             DO I=0,N1
18              20                UH(I,J,K,1)=0.0
19              11                UH(I,J,K,2)=0.0
20             169                UH(I,J,K,3)=0.0
21             202                SGSVIS(I,J,K)=0.0
22                             ENDDO
23                             ENDDO
24                             DO J=1,N2
25                             DO I=1,N1
26              98                DP(I,J,K)=0.0
27                             ENDDO
28                             ENDDO
29                            ENDDO
```

① parallel do를 이용해 K 루프를 병렬화 하였다.

② 인덱스 K에 대해서 Fusion을 하였다.

6.4 uhcalc.f

<Original 코드>

```
23                            DO 400 K=1,N3M
24                            DO 400 J=1,N2M
25                            DO 400 I=IIU(J),NNU(J)
26              78        400  UH(I,J,K,1)=U(I,J,K,1)+UH(I,J,K,1)
27
28                            DO 410 K=1,N3M
29                            DO 410 J=2,N2M
30                            DO 410 I=IIV(J),NNV(J)
31              82        410  UH(I,J,K,2)=U(I,J,K,2)+UH(I,J,K,2)
32
33                            DO 420 K=1,N3M
34                            DO 420 J=1,N2M
35                            DO 420 I=IIW(J),NNW(J)
36              90        420  UH(I,J,K,3)=U(I,J,K,3)+UH(I,J,K,3)
```

<최적화/병렬화 코드>

```
24                      !$omp parallel do private(J)
25                            DO K=1,N3M
26                                J=1
27                             DO I=IIU(J),NNU(J)
28                                UH(I,J,K,1)=U(I,J,K,1)+UH(I,J,K,1)
29                             ENDDO
30                             DO I=IIW(J),NNW(J)
31           2                    UH(I,J,K,3)=U(I,J,K,3)+UH(I,J,K,3)
32                             ENDDO
34                             DO J=2,N2M
35                             DO I=IIU(J),NNU(J)
36          74                    UH(I,J,K,1)=U(I,J,K,1)+UH(I,J,K,1)
37                             ENDDO
38                             DO I=IIV(J),NNV(J)
39          65                    UH(I,J,K,2)=U(I,J,K,2)+UH(I,J,K,2)
40                             ENDDO
41                             DO I=IIW(J),NNW(J)
42         113                    UH(I,J,K,3)=U(I,J,K,3)+UH(I,J,K,3)
43                             ENDDO
44                             ENDDO
46                            ENDDO
```

① 인덱스 K에 대해서 Fusion을 하였다. 이 과정에서 J 인덱스가 일치하지 않기에 J가 1일 때 UH(I,J,K,1)과 UH(I,J,K,3)의 계산을 수행한 후, J 루프를 2부터 실행 하고 있다.

② parallel do를 이용해 K 루프를 병렬화 하였다.

6.5 rhs1.f

<Original 코드>

```
 12                    DO 10 K=1,N3M
 15                    DO 10 J=1,N2M
 20        5           DO 10 I=II,NN
 31       29           SGSI2=SGSVIS(I ,J,K)
 32       35           SGSI1=SGSVIS(IM,J,K)
 33       33           NUTI=0.5*(SGSI1+SGSI2) ! M03-008
 34       88           SGSJP=0.5*(DX(IM)*SGSVIS(I,JP,K)
                      >     +DX(I)*SGSVIS(IM,JP,K))/G(I)
249           36       10    CONTINUE
```

```
254                         DO 110 K=1,N3M
255                         SGSI2=SGSVIS(IB2   ,JB1-1,K)
258                         SGSJP=0.5*(DX(IB2-1)*SGSVIS(IB2   ,JB1   ,K)
259                        >        +DX(IB2   )*SGSVIS(IB2-1,JB1   ,K))/G(IB2)
294                   110   CONTINUE
296                         DO 120 K=1,N3M
297                         SGSI2=SGSVIS(IB3   ,JB1-1,K)
300                         SGSJP=0.5*(DX(IB3-1)*SGSVIS(IB3   ,JB1   ,K)
301                        >        +DX(IB3   )*SGSVIS(IB3-1,JB1   ,K))/G(IB3)
335                   120   CONTINUE
337                         DO 130 K=1,N3M
338                         SGSI2=SGSVIS(IB2   ,JB2,K)
341                         SGSJP=0.5*(DX(IB2-1)*SGSVIS(IB2   ,JB2+1,K)
342                        >        +DX(IB2   )*SGSVIS(IB2-1,JB2+1,K))/G(IB2)
376                   130   CONTINUE
378                         DO 140 K=1,N3M
379                         SGSI2=SGSVIS(IB3   ,JB2,K)
382                         SGSJP=0.5*(DX(IB3-1)*SGSVIS(IB3   ,JB2+1,K)
383                        >        +DX(IB3   )*SGSVIS(IB3-1,JB2+1,K))/G(IB3)
417                   140   CONTINUE
```

<최적화/병렬화 코드>

```
 12                         !$omp   parallel do private(KP,KM,JP,JM,IP,IM,
 13                         !$omp& II,NN,FIUP1,FIUM1,FJUP1,FJUM1,JJ,MM, SGSJ1,
 14                         !$omp& SGSI2,SGSI1,NUTI,SGSJP,SGSJC,SGSJM,SGSJ2,
 15                         !$omp& NUTJ,SGSKP,SGSKC,SGSKM,SGSK2,SGSK1,NUTK,
 16                         !$omp& U2,U1,VIS11,V2,V1,VIS12,W2,W1,VIS13,VISCOS,
 17                         !$omp& BC2,BC1,BC_DOWN,BC_UP,BC_IN,BC_OUT,BC,
 18                         !$omp& APJ,ACJ,AMJ,API,ACI,AMI,APK,ACK,AMK, PRESSG1,
 19                         !$omp& RM11U_N,RM12V_N,RM13W_N, BC_DOWN_RIGHT,
 20                         !$omp& BC_UP_WRONG,BC_UP_RIGHT,BC_DOWN_WRONG)
 21                               DO 30 K=1,N3M
 24                               DO 10 J=1,N2M
 29                    3          DO 10 I=II,NN
 40                               SGSI2=SGSVIS(I ,J,K)
 43                  147          SGSJP=0.5*(DX(IM)*SGSVIS(I,JP,K)
                                 >       +DX(I)*SGSVIS(IM,JP,K))*GR(I)
257                   2      10   CONTINUE
261                              SGSI2=SGSVIS(IB2   ,JB1-1,K)
264                              SGSJP=0.5*(DX(IB2-1)*SGSVIS(IB2   ,JB1   ,K)
265                                       >                +DX(IB2    )*SGSVIS(IB2-
1,JB1   ,K))*GR(IB2)
417                         30    CONTINUE
```

① 인덱스 K 에 대해서 fusion을 하였다.

② parallel do를 이용해 K 루프를 병렬화 하였다.

6.6 getuh1.f

<Original 코드>

```
 15                        DO 2 K=1,N3M
 16            4           DO 20 J=1,N2M
 19                        DO 20 I=2,N1M
 28                        SGSI2=SGSVIS(I ,J,K)
 31          109           SGSJP=0.5*(DX(IM)*SGSVIS(I,JP,K)
                          >     +DX(I)*SGSVIS(IM,JP,K))/G(I)
 61           89           R2(I,J)=RUH1(I,J,K)*DT
 62                    20  CONTINUE
 63                        CALL TDMAI(AMJ,ACJ,APJ,R2,R2,1,N2M,2,N1M,1,0,0)
 64                        DO 21 J=1,N2M
 65                        DO 21 I=2,N1M
 66           71       21  UH(I,J,K,1)=R2(I,J)
 67                    2   CONTINUE
 69                        DO 1 K=1,N3M
 70                        DO 10 J=1,N2M
 71                        DO 10 I=2,N1M
 77           47           SGSI2=SGSVIS(I ,J,K)
 81           30           U2=0.5*(U(IP,J,K,1)+U(I ,J,K,1))
 97           40           R2(I,J)=UH(I,J,K,1)
 98                    10  CONTINUE
 99                        CALL TDMAJ(AMJ,ACJ,APJ,R2,R2,2,N1M,1,N2M,1,0,0)
100                        DO 11 J=1,N2M
101                        DO 11 I=2,N1M
102           41       11  UH(I,J,K,1)=R2(I,J)
103                    1   CONTINUE
```

<최적화/병렬화 코드>

```
 15                    !$omp parallel do private(JP,JM,IP,IM,
 16                    !$omp& II,NN,FIUP1,FIUM1,FJUP1,FJUM1,JJ,MM,
 17                    !$omp& SGSI2,SGSI1,NUTI,
 18                    !$omp& SGSJP,SGSJC,SGSJM,SGSJ2,SGSJ1,NUTJ,
 19                    !$omp& V2,V1,APJ,ACJ,AMJ,
 20                    !$omp& U2,U1)
```

```
 21                      DO 1 K=1,N3M
 22                      DO 20 J=1,N2M
 28           15         DO 20 I=II,NN
 37                      SGSI2=SGSVIS(I ,J,K)
 40           21         SGSJP=0.5*(DX(IM)*SGSVIS(I,JP,K)
                        >   +DX(I)*SGSVIS(IM,JP,K))*GR(I)
 70           50         UH(I,J,K,1)=RUH1(I,J,K)*DT
 71            2     20  CONTINUE
 72                      CALL TDMAI(AMJ,ACJ,APJ,UH(0,0,K,1),1,N2M,2,
                        >   N1M,1,0,0)
 73                    2 CONTINUE
 75                      DO 10 J=1,N2M
 79            7         DO 10 I=II,NN
 85            1         SGSI2=SGSVIS(I ,J,K)
105            9     10  CONTINUE
106                      CALL TDMAJ(AMJ,ACJ,APJ,UH(0,0,K,1),
                        >   IIU,NNU,1,N2M)
107                    1 CONTINUE
```

① parallel do를 이용해 K 루프를 병렬화 하였다.

② 인덱스 K 에 대해서 fusion을 하였다.

③ 각 K에 대해 UH(I,J,K,1) → R2(I,J) → (call TDMAJ) → R2(I,J) → UH(I,J,K,1)로 진행되는 흐름을 UH(I,J,K,1) → (call TDMAJ) → UH(I,J,K,1)으로 단순화 시켜 불필요한 데이터 할당과 메모리 낭비를 줄이고 있다.

<Original 코드: getuh1.f 계속>

```
105                      DO 3 J=1,N2M
108                      DO 30 K=1,N3M
111                      DO 30 I=II,NN
115           11         SGSKP=0.5*(DX(IM)*SGSVIS(I,J,KP)
                        >    +DX(I)*SGSVIS(IM,J,KP))/G(I)
118            7         SGSK2=0.5*(SGSKC+SGSKP)
120                      NUTK=0.5*(SGSK1+SGSK2)
125           39         W2=0.5*(DX(IM)*U(I,J,KP,3)
                        >      +DX(I)*U(IM,J,KP,3))/G(I)
127           47         APK(I,K)=(
128                      >          -0.5*DX3Q/RM(J)**2.0*(NUTI+1./RE)
129                      >          -0.25*DX3Q/RM(J)**2.0*(SGSK2-SGSK1)
```

```
130                     >          +0.25*DX3/RM(J)**2.0*W2
131                     >          )*DT
141           27          R3(I,K)=UH(I,J,K,1)
142                   30  CONTINUE
143                       IF (N3M.NE.1) THEN
144                       CALL CTDMA3I(AMK,ACK,APK,R3,UH,1,J,N3M,II,NN)
145                       ENDIF
146                   3   CONTINUE
```

<최적화/병렬화 코드: getuh1.f 계속 >

```
109                   !$omp parallel do private(KP,KM,IP,IM,II,NN,
110                   !$omp& SGSKP,SGSKC,SGSKM,SGSK2,SGSK1,NUTK,
111                   !$omp& SGSI2,SGSI1,NUTI,
112                   !$omp& W2,W1,APK,ACK,AMK)
113                       DO 3 J=1,N2M
116                       DO 30 K=1,N3M
119                       DO 30 I=II,NN
123           37          SGSKP=0.5*(DX(IM)*SGSVIS(I,J,KP)
                          >      +DX(I)*SGSVIS(IM,J,KP))*GR(I)
126           10          SGSK2=0.5*(SGSKC+SGSKP)
128                       NUTK=0.5*(SGSK1+SGSK2)
133           49          W2=0.5*(DX(IM)*U(I,J,KP,3)
                         >    +DX(I)*U(IM,J,KP,3))*GR(I)
135           28          APK(I,K)=(
136                       >          -0.5*DX3Q/RM(J)**2.0*(NUTI+1./RE)
137                       >          -0.25*DX3Q/RM(J)**2.0*(SGSK2-SGSK1)
138                       >          +0.25*DX3/RM(J)**2.0*W2
139                       >          )*DT
149                   30  CONTINUE
150                       IF (N3M.NE.1) THEN
151                       CALL CTDMA3I(AMK,ACK,APK,UH(0,0,0,1)
                         >                  ,J,N3M,II,NN)
152                       ENDIF
153                   3   CONTINUE
```

④ parallel do를 이용해 J루프를 병렬화 하였다.

6.7 nutcheck.f

<Original 코드>

```
14                        DO 10 K=1,N3M
```

```
15                          DO 10 J=0,N2
16                          DO 10 I=1,N1
17          14              SGSVISL=SGSVIS(I,J,K)
18          30              IF(SGSVISL.LT.SGSVISM) THEN
19                          SGSVISM=SGSVISL
20                          IMIN=I
21                          JMIN=J
22                          KMIN=K
23                          ENDIF
24          30      10      CONTINUE
```

<최적화/병렬화 코드>

```
13                  !$omp parallel do private(ks,ke,SGSVISL)
14                  do ii=0,nthds-1
15                      ks=kse(1,ii)
16                      ke=kse(2,ii)
17                  SGSVISM(ii)=1.0E+10
18                  IMIN(ii)=0
19                  JMIN(ii)=0
20                  KMIN(ii)=0
21
22                  DO 10 K=ks,ke
23                  DO 10 J=0,N2
24                  DO 10 I=1,N1
25          62      SGSVISL=SGSVIS(I,J,K)
26           4      IF(SGSVISL.LT.SGSVISM(ii)) THEN
27                  SGSVISM(ii)=SGSVISL
28                  IMIN(ii)=I
29                  JMIN(ii)=J
30           2      KMIN(ii)=K
31                  ENDIF
32           5  10  CONTINUE
33                  ENDDO
35                      ii=0
36                      i1=ii
37                  SGSVISM1=SGSVISM(ii)
38                  do ii=1,nthds-1
39                  SGSVISL=SGSVISM(ii)
40                  IF(SGSVISL.LT.SGSVISM1) THEN
41                  SGSVISM1=SGSVISL
42                   i1=ii
43                  ENDIF
44                  ENDDO
45                  IMIN1=IMIN(i1)
```

```
46                              JMIN1=JMIN(i1)
47                              KMIN1=KMIN(i1)
```

① OpenMP 지시어를 사용하여 병렬화를 하였다.

② 병렬영역 내에서 SGSVISL의 최소값 SGSVISM을 구하고 있다. 병렬영역 내에서 각 스레드마다 로컬 최소값을 구하고 그때의 IMIN, JMIN, KMIN값을 저장하기 위해서 배열을 이용하고 있다. 각 스레드에서 계산된 로컬 최소값을 비교해 글로벌 최소값을 구하기 위해서 병렬영역 밖에서 다시 한 번 비교 과정을 거치고 있다.

6.8 takedp.f

<Original 코드>

```
241                     C---   FOR IMSL
242                            CALL DFFTRI(N3M,WFFTR)
244                            DO 110 J=1,N2M
245                            II=IIW(J)
246                            NN=NNW(J)
247                            DO 110 I=II,NN
249              6             DO K=1,N3M
250           1184                 RFIN(K)=RDP(I,J,K)
251              1             END DO
253              2             CALL DF2TRF(N3M,RFIN,RFEX,WFFTR)
255              1                FRDP_R(I,J,0)=RFEX(1)
256                               FRDP_I(I,J,0)=0.0
257             11             DO M=1,N3M/2-1
258            267                FRDP_R(I,J,M)=RFEX(2*M)
259            379                FRDP_I(I,J,M)=RFEX(2*M+1)
260                           ENDDO
261                              FRDP_R(I,J,N3M/2)=RFEX(N3M)
263                     110    CONTINUE
```

<최적화/병렬화 코드>

```
 78                     C---   FOR IMSL
 80                            isign=1
 81                            scale=1.d0
 82                     !$omp   parallel do private(II,NN,M,yy,aux1,aux2,mm)
 83                            DO 110 J=1,N2M
```

```
 84                      II=IIW(J)
 85                      NN=NNW(J)
 86                      MM=NN-II+1
 87                      DO K=1,N3M
 88                      DO I=II,NN
 89          514             yy(K,I)=RDP(I,J,K)
 90                      ENDDO
 91            1         ENDDO
 93            3         call drcft(1,yy,N3M+2,yy,(N3M+2)/2,N3M,mm,isign,
 94                     & scale,aux1,naux1,aux2,naux2)
 95                      call drcft(0,yy(1,II),N3M+2,yy(1,II),(N3M+2)/2,
 96                     & N3M,mm,isign,scale,aux1,naux1,aux2,naux2)
 98            3         DO I=II,NN
 99                      DO M=0,N3M/2-1
100           74            FRDP_R(I,J,M)=yy(2*M+1,I)
101          364            FRDP_I(I,J,M)=yy(2*M+2,I)
102                     enddo
103                         M=N3M/2
104           13            FRDP_R(I,J,M)=yy(2*M+1,I)
105            3        ENDDO
107                 110 CONTINUE
```

① Original 코드에서는 3차원 배열 RDP(I,J,K)를 K에 대해 읽어 들여 RFIN(K)에 넣고 있는데 이 과정에서 많은 캐시실패가 발생하고 있다. 이렇게 읽어 들인 RFIN(K)를 IMSL 루틴 DF2TRF를 이용해 1차원 FFT를 수행하고 그 결과로 나오는 RFEX를 real 부분(FRDP_R)과 imaginary 부분(FRDP_I)으로 나눠 할당하고 있다. 이러한 일련의 과정이 J, I 루프 내에서 반복되고 있고 따라서 IMSL루틴 DF2TRF 역시 각 반복마다 호출돼 FFT를 수행하고 있다.

② 최적화 코드에서는 RDP(I,J,K)를 읽어 들이는 과정에서 I, K 루프를 돌려 RDP(I,J,K)의 값을 2차원 배열 yy(K,I)에 넣고 있다. 이 과정에서 original 코드와 비교하여 캐시실패를 많이 줄일 수 있다. 1차원배열을 모아 놓은 2차원 배열 yy는 ESSL 루틴 drcft를 호출해 한꺼번에 FFT를 수행한다. drcft는 1차원 real to complex FFT를 수행하는 ESSL 루틴으로 여기서는 2차원 배열 yy의 각 열에 대해 FFT를 수행하고 그 결과를 yy의 각 열에 저장한다. 이런 처리를 하게 됨으로써 최적화 코드의 FFT 루틴 호출의 회수는 original 코드

와 비교해 대폭 줄어들게 되고, 또 한 번의 호출에 더 많은 계산을 수행하게 되므로 코드 효율성을 높일 수 있다.

③ parallel do를 이용해 J 루프를 병렬화 하였다.

<Original 코드: takedp.f 계속>

```
266                     DO 70 K=1,N3M/2+1
267          1          DO 71 J=1,N2M
268                     II=IIW(J)
269          2          NN=NNW(J)
270          2          DO 71 I=II,NN
272        130          IJ=IGJG(I,J,1)
273        189          PHM(IJ)=FDP_R(I,J,K-1)*(1.0-REAL(IPH0))
274        200          REST(IJ)=FRDP_R(I,J,K-1)
275         89     71   CONTINUE
276                     CALL DSOLV(PHM,REST,K)
278                     DO 72 J=1,N2M
279                     II=IIW(J)
280                     NN=NNW(J)
281                     DO 72 I=II,NN
283         66          IJ=IGJG(I,J,1)
284        151          FDP_R(I,J,K-1)=PHM(IJ)
285         39     72   CONTINUE
286                70   CONTINUE
```

<최적화/병렬화 코드 : takedp.f 계속 >

```
110                !$omp  parallel do private(II,NN,IJ,PHM,REST)
111                     DO 70 K=1,N3M/2+1
112          1          DO 71 J=1,N2M
113                     II=IIW(J)
114                     NN=NNW(J)
115          1          DO 71 I=II,NN
117                     IJ=IGJG(I,J,1)
118                     PHM(IJ)=FDP_R(I,J,K-1)*(1.0-REAL(IPH0))
119        637          REST(IJ)=FRDP_R(I,J,K-1)
120          4     71   CONTINUE
121                     CALL DSOLV(PHM,REST,K)
123                     DO 72 J=1,N2M
124                     II=IIW(J)
125                     NN=NNW(J)
126                     DO 72 I=II,NN
128                     IJ=IGJG(I,J,1)
```

```
129          166          FDP_R(I,J,K-1)=PHM(IJ)
130           14     72    CONTINUE
131                  70    CONTINUE
```

④ parallel do를 이용해 K 루프를 병렬화 하였다.

<Original 코드: takedp.f 계속 >

```
288                       DO 80 K=2,N3M/2
289                       DO 81 J=1,N2M
290                       II=IIW(J)
291                       NN=NNW(J)
292           3           DO 81 I=II,NN
294         134           IJ=IGJG(I,J,1)
295         182           PHM(IJ)=FDP_I(I,J,K-1)*(1.0-REAL(IPH0))
296         206           REST(IJ)=FRDP_I(I,J,K-1)
297          64     81    CONTINUE
298                       CALL DSOLV(PHM,REST,K)
299                       DO 82 J=1,N2M
300                       II=IIW(J)
301                       NN=NNW(J)
302                       DO 82 I=II,NN
304          51           IJ=IGJG(I,J,1)
305         132           FDP_I(I,J,K-1)=PHM(IJ)
306          27     82    CONTINUE
307                  80    CONTINUE
```

<최적화/병렬화 코드 : takedp.f 계속 >

```
133                       !$omp   parallel do private(II,NN,IJ,PHM,REST)
134                       DO 80 K=2,N3M/2
135           1           DO 81 J=1,N2M
136                       II=IIW(J)
137                       NN=NNW(J)
138           2           DO 81 I=II,NN
140                       IJ=IGJG(I,J,1)
141                       PHM(IJ)=FDP_I(I,J,K-1)*(1.0-REAL(IPH0))
142         604           REST(IJ)=FRDP_I(I,J,K-1)
143                  81    CONTINUE
144                       CALL DSOLV(PHM,REST,K)
145                       DO 82 J=1,N2M
146                       II=IIW(J)
147                       NN=NNW(J)
148           2           DO 82 I=II,NN
```

```
150                         IJ=IGJG(I,J,1)
151          147            FDP_I(I,J,K-1)=PHM(IJ)
152           15     82     CONTINUE
153                   80     CONTINUE
```

⑤ parallel do를 이용해 K 루프를 병렬화 하였다.

<Original 코드 : takedp.f 계속 >

```
334                   C---   FOR IMSL
335                          DO 200 J=1,N2M
336                          II=IIW(J)
337                          NN=NNW(J)
338                          DO 200 I=II,NN
340           17                RFIN(1)=FDP_R(I,J,0)
341                          DO M=1,N3M/2-1
342          635                RFIN(2*M)=FDP_R(I,J,M)
343          694                RFIN(2*M+1)=FDP_I(I,J,M)
344                          ENDDO
345                             RFIN(N3M)=FDP_R(I,J,N3M/2)
347            8             CALL DF2TRB(N3M,RFIN,RFEX,WFFTR)
349            4             SCALE = 1.0/REAL(N3M)
351           23             DO 23 K=1,N3M
352          779      23     DP(I,J,K)=RFEX(K)*SCALE
354           11     200     CONTINUE
```

<최적화/병렬화 코드 : takedp.f 계속 >

```
180                   C---   FOR IMSL
181                          isign=-1
182                          SCALE = 1.0/REAL(N3M)
183                   !$omp   parallel do private(II,NN,M,yy,aux1,aux2,mm)
184            1             DO 200 J=1,N2M
185                          II=IIW(J)
186                          NN=NNW(J)
187                          MM=NN-II+1
188                             M=0
189                          DO I=II,NN
190                             yy(2*M+1,I)=FDP_R(I,J,0)
191                             yy(2*M+2,I)=0.
192                          ENDDO
193                          DO M=1,N3M/2-1
194                          DO I=II,NN
195                             yy(2*M+1,I)=FDP_R(I,J,M)
```

```
196                                 yy(2*M+2,I)=FDP_I(I,J,M)
197                               ENDDO
198                               ENDDO
199                                 M=N3M/2
200                               DO I=II,NN
201                                 yy(2*M+1,I)=FDP_R(I,J,M)
202                                 yy(2*M+2,I)=0.
203                               ENDDO
205                               call dcrft(1,yy,(N3M+2)/2,yy,N3M+2,N3M,mm,
206                              & isign,scale,aux1,naux1,aux2,naux2)
207                               call dcrft(0,yy(1,II),(N3M+2)/2,yy(1,II),N3M+2,
208                              & N3M,mm,isign,scale,aux1,naux1,aux2,naux2)
210                               DO I=II,NN
211                                 DO K=1,N3M
212                                   DP(I,J,K)=yy(K,I)
213                                 ENDDO
214                               ENDDO
216                       200     CONTINUE
```

⑥ Backward Fourier Transform이 수행되는 부분으로 앞서 Forward FFT를 수행한 부분과 동일한 최적화 병렬화 과정이 진행되었다. Original코드에서는 I루프 내에서 M루프를 돌려 FDP_R(I,J,M), FDP_I(I,J,M)을 1차원 배열 RFIN에 할당하고 있고 이 과정에서 많은 캐시실패가 발생하게 된다. 최적화 코드에서는 M루프 내에서 I 루프를 돌려 FDP_R(I,J,M), FDP_I(I,J,M)을 2차원 배열 yy에 저장하고 있고 이를 통해 많은 캐시실패를 줄이고 있다. 아울러 2차원 배열 yy를 ESSL 루틴 dcrft를 이용해 계산하면서 original 코드와 비교해 서브루틴 호출을 줄이고 한 번에 더 많은 계산을 수행하게 함으로써 코드 효율성을 높이고 있다.

⑦ parallel do를 이용해 J 루프를 병렬화 하였다.

6.9 resma.f

<Original 코드>

```
421                       SUBROUTINE RESMA(NL,PHM,REML1,REML2,K,REST)
423                       INCLUDE 'PARAM.H'
424                       INCLUDE 'COMMON.H'
```

```
426                          REAL PHM(NIJ),REST(NIJ)
428                          IGJG(I,J,L)=I+(J-1)*(N1G(L)+1)+N12G(L-1)
430                          REML1=0.
431                          REML2=0.
432                          DO 10 JC=1,N2M
433            2             IC0=IIW(JC)
434           30             IC1=NNW(JC)
435            9             II=(IC0-1)*2**FLOAT(-NL+1)+1
436           94             NN=(IC1)*2**FLOAT(-NL+1)
437           16             DO 20 IC=II,NN
439         2398              IJ=IGJG(IC,JC,NL)
440        22414              D2PH=+(CO(1,IJ)
                             >          *PHM(IGJG(IPLV(IC,JC,NL),JC,NL))+
441                          >          CO(2,IJ)*PHM(IGJG(IMLV(IC,JC,NL),JC,NL))+
442                          >          CO(3,IJ)*PHM(IGJG(IC,JPLV(IC,JC,NL),NL))+
443                          >          CO(4,IJ)*PHM(IGJG(IC,JMLV(IC,JC,NL),NL))+
444                          >         (CO(5,IJ)-AK3(K)/RM(JC)**2.)*PHM(IJ))
445         1160              RS=(-D2PH+REST(IJ))
457         2100              REML1=AMAX1(ABS(RS),REML1)
458         2772              REML2=RS**2*CO(6,IJ)+REML2
459         1224       20     CONTINUE
460           16       10     CONTINUE
463            1              REML2=SQRT(REML2/(N1G(NL)*N2M))
468                    337    FORMAT(4(I5,1X),1(E12.5,1X))
469                           RETURN
470            1              END
```

<최적화/병렬화 코드>

```
 7                           SUBROUTINE RESMA(NL,PHM,REML1,REML2,K,REST,
 8                          & CO1,LDA,IPLV1,IMLV1,JPLV1,JMLV1)
10                           INCLUDE 'PARAM.H'
11                           INCLUDE 'COMMON.H'
13                           REAL CO1(6,LDA,*),PHM(LDA,*),REST(LDA,*)
14                           INTEGER IPLV1(M1,M2),IMLV1(M1,M2)
15                           INTEGER JPLV1(M1,M2),JMLV1(M1,M2)
17                           REML1=0.
18                           REML2=0.
19                           DO 10 JC=1,N2M
20                           IC0=IIW(JC)
21                           IC1=NNW(JC)
22                           II=ISHFT(IC0-1,-NL+1)+1
23                           NN=ISHFT(IC1,-NL+1)
24             33            DO 20 IC=II,NN
25                           D2PH=+(CO1(1,IC,JC)*PHM(IPLV1(IC,JC),JC)+
```

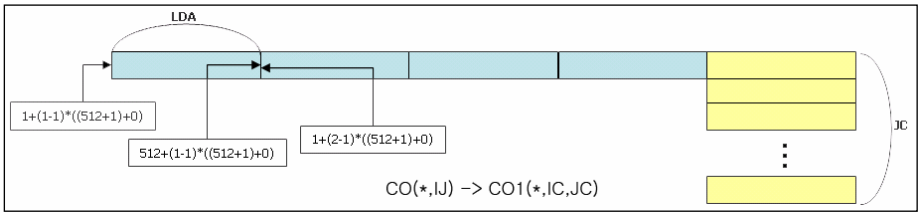
```
26                      >        CO1(2,IC,JC)*PHM(IMLV1(IC,JC),JC)+
27                      >        CO1(3,IC,JC)*PHM(IC,JPLV1(IC,JC))+
28                      >        CO1(4,IC,JC)*PHM(IC,JMLV1(IC,JC))+
29                      >       (CO1(5,IC,JC)-AK3(K)/RM(JC)**2.)*PHM(IC,JC))
30        14814          RS=(-D2PH+REST(IC,JC))
31          411          REML1=AMAX1(ABS(RS),REML1)
32          797          REML2=RS**2*CO1(6,IC,JC)+REML2
33                  20   CONTINUE
34           14     10   CONTINUE
36                      REML2=SQRT(REML2/(N1G(NL)*N2M))
37                  337  FORMAT(4(I5,1X),1(E12.5,1X))
38                      RETURN
```

① Original 코드에서의 (IC0-1)*2**FLOAT(-NL+1) 계산을 최적화 코드에서 비트 이동을 지시하는 포트란 함수인 ISHFT를 이용해 수행하였다.

- A*2^x 계산은 ISHFT(A,x)와 그 결과가 같다.
- 예) ISHFT(3,2)는 3의 2진 표현 11을 두 자리 만큼 이동하도록 해서 그 값이 2진 표현으로 1100이 되도록 한다. 이것은 10진수로 12가 돼 3*2^2과 그 결과가 동일하다.

② Original 코드에서 인덱스 IC, JC의 함수인 IGJG를 이용해 계산되는 인덱스 IJ를 통해 배열 CO, PHM, REST 등에 접근하고 있다. 최적화 코드에서는 IGJG에 의해 계산되는 인덱스를 사용하지 않고 IC, JC를 직접 배열의 인덱스로 사용하고 있다. 이는 함수 IGJG를 거치는 과정이 생략되기에 훨씬 효율적이다. 이를 위해 2차원 배열 CO를 3차원 배열 CO1으로 변경하였고 PHM과 REST도 각각 2차원 배열로 변경해 사용하고 있다.

③ 아래 그림은 2차원 배열의 CO(5, IJ)가 3차원 배열 CO1(5,IC,JC)로 변경되는 과정을 나타내고 있다. 여기서 앞의 5는 상수의 형태이므로 뒤의 CO배열의 IJ를 CO1배열의 IC , JC로 1차원에서 2차원으로 변경되는 그림으로 간략화하여 나타내었다.



< 그림 III.7 CO(5,IJ) 배열의 CO1(5,IC,JC)로 변경방법 >

6.10 cfila.f

이곳에서 호출해 사용하는 함수 ITSOLV, RESCA, TFILA의 변화된 내용에 대해서는 다음 부분에 따로 정리하였다.

<Original 코드>

```
494                        IGJG(I,J,L)=I+(J-1)*(N1G(L)+1)+N12G(L-1)
496                        DO 20 NL=MLI,MMLEV
498                        CALL ITSOLV(NL,PHM,REST,K)
499          2             CALL RESCA(NL,PHM,RES,K,REST)
501          1             DO 19 JC=1,N2M
502                        IC0=IIW(JC)
503         14             IC1=NNW(JC)
504        134             II=(IC0-1)*2**FLOAT(-NL)+1
505        107             NN=(IC1)*2**FLOAT(-NL)
506         47             DO 19 IC=II,NN
509        877             IJ=IGJG(IC,JC,NL+1)
510       1154             PHM(IJ)=0.
511         22      19     CONTINUE
512                        IF(NL.LT.MMLEV) THEN
513                 c---   get interpolated residual REST at level NL+1
514                 c      from the residual RES in finer mesh
515                        CALL TFILA(RES,NL,REST)
516                        ENDIF
517          3      20     CONTINUE
```

<최적화/병렬화 코드>

```
24                         IGJG(I,J,L)=I+(J-1)*(N1G(L)+1)+N12G(L-1)
26                         DO 20 NL=MLI,MMLEV
28           3             CALL ITSOLV(NL,PHM,REST,K,
29                        & CO(1,1+N12G(NL-1)),N1G(NL)+1,
```

```
30                          & PHM(1+N12G(NL-1)),REST(1+N12G(NL-1)),
31                          & IPLV(1,1,NL),IMLV(1,1,NL))
32
33          4                CALL RESCA(NL,PHM,RES,K,REST,
34                          & CO(1,1+N12G(NL-1)),N1G(NL)+1,
35                          & PHM(1+N12G(NL-1)),RES(1+N12G(NL-1)),
36                          & REST(1+N12G(NL-1)),IPLV(1,1,NL),IMLV(1,1,NL),
37                          & JPLV(1,1,NL),JMLV(1,1,NL))
39          3                DO 19 JC=1,N2M
40                           IC0=IIW(JC)
41                           IC1=NNW(JC)
42                           II=ISHFT(IC0-1,-NL)+1
43                           NN=ISHFT(IC1,-NL)
44         54                DO 19 IC=II,NN
46                           IJ=IC+(JC-1)*(N1G(NL+1)+1)+N12G(NL)
47       1333                PHM(IJ)=0.
48          8        19      CONTINUE
49                           IF(NL.LT.MMLEV) THEN
50                   c---    get interpolated residual REST at level NL+1
51                   c        from the residual RES in finer mesh
52                            CALL TFILA(RES,NL,REST,
53                           & CO(1,1+N12G(NL-1)),N1G(NL)+1,
54                           & CO(1,1+N12G(NL  )),N1G(NL+1)+1,
55                           & RES(1+N12G(NL-1)),REST(1+N12G(NL  )))
56                           ENDIF
57                   20      CONTINUE
```

① (IC0-1)*2**FLOAT(-NL+1) 계산을 포트란 함수인 ISHFT로 변경하였다.

② 함수 IGJG를 인라이닝 시켰다.

6.11 itsolv.f

<Original 코드>

```
544                          DO 10 JC=1,N2M
545          6               IC0=IIW(JC)
546         17               IC1=NNW(JC)
547        311               II=(IC0-1)*2**FLOAT(-NL+1)+1
548         64               NN=(IC1)*2**FLOAT(-NL+1)
549        109               DO 10 IC=II,NN,2
551       9467               IJ=IGJG(IC,JC,NL)
```

```
552          2362              APJ(JC,IC)=CO(3,IJ)
553          4892              ACJ(JC,IC)=CO(5,IJ)-AK3(K)/RM(JC)**2.
554           966              AMJ(JC,IC)=CO(4,IJ)
555         21644              FJJ(JC,IC)
556                           >    =(-(CO(1,IJ)*PHIT(IGJG(IPLV(IC,JC,NL),JC,NL))
557                           >        +CO(2,IJ)*PHIT(IGJG(IMLV(IC,JC,NL),JC,NL)))
558                           >        +RES(IJ))
559          1726        10    CONTINUE
568             4              CALL TRIBI(AMJ,ACJ,APJ,FJJ,2,N2M,FJJ,1,N1G(NL),NL)
571            12              DO 40 JC=1,N2M
572             4              IC0=IIW(JC)
573            68              IC1=NNW(JC)
574           104              II=(IC0-1)*2**FLOAT(-NL+1)+1
575            99              NN=(IC1)*2**FLOAT(-NL+1)
576           157              DO 40 IC=II,NN,2
578          4471              IJ=IGJG(IC,JC,NL)
579          1071              PHIT(IJ)=FJJ(JC,IC)
580            62        40    CONTINUE
```

<최적화/병렬화 코드>

```
 26             7              DO 10 JC=1,N2M
 27                            IC0=IIW(JC)
 28                            IC1=NNW(JC)
 29                            II=ISHFT(IC0-1,-NL+1)+1
 30                            NN=ISHFT(IC1,-NL+1)
 31                            NN1=(NN-II)/2+1
 32                            ICS=(II+1)/2
 35                                IC=II
 36           247              DO IC2=ICS,ICS+NN1-1
 37          1679              APJ(IC2,JC)=CO1(3,IC,JC)
 38          2015              ACJ(IC2,JC)=CO1(5,IC,JC)-AK3(K)/RM(JC)**2.
 39           329              AMJ(IC2,JC)=CO1(4,IC,JC)
 40         15035              FJJ(IC2,JC)
 41                           >    =(-(CO1(1,IC,JC)*PHIT1(IPLV1(IC,JC),JC)
 42                           >        +CO1(2,IC,JC)*PHIT1(IMLV1(IC,JC),JC))
 43                           >        +RES1(IC,JC))
 44                            IC=IC+2
 45           332              ENDDO
 46            43        10    CONTINUE
 48             4              CALL TRIBI(AMJ,ACJ,APJ,FJJ,2,N2M,FJJ,1,N1G(NL),NL)
 50             3              DO 40 JC=1,N2M
 51                            IC0=IIW(JC)
 52                            IC1=NNW(JC)
 53                            II=ISHFT(IC0-1,-NL+1)+1
```

```
54                              NN=ISHFT(IC1,-NL+1)
55                              NN1=(NN-II)/2+1
56                              ICS=(II+1)/2
59                                  IC=II
60          86                  DO IC2=ICS,ICS+NN1-1
61        3388                  PHIT1(IC,JC)=FJJ(IC2,JC)
62                              IC=IC+2
63           1                  ENDDO
64          24          40      CONTINUE
```

① Original 코드에서의 곱셈 A*2**FLOAT(X)를 ISHFT(A,X)로 대체하였다.

② IC, JC의 함수 IGJG에 의해 계산되는 인덱스 IJ를 사용하지 않고 IC, JC를 배열 CO1, PHIT1, RES1에 직접 사용하였다.

<Original 코드 : itsolv.f 계속>

```
591           2                 DO 110 JC=1,N2M
592           6                 IC0=IIW(JC)
593          18                 IC1=NNW(JC)
594         177                 II=(IC0-1)*2**FLOAT(-NL+1)+1
595         185                 NN=(IC1)*2**FLOAT(-NL+1)
596         198                 DO 110 IC=II+1,NN,2
598        5026                 IJ=IGJG(IC,JC,NL)
599        2434                 APJ(JC,IC)=CO(3,IJ)
600        1983                 ACJ(JC,IC)=CO(5,IJ)-AK3(K)/RM(JC)**2.
601         897                 AMJ(JC,IC)=CO(4,IJ)
602       27753                 FJJ(JC,IC)
603                            >   =(-(CO(1,IJ)*PHIT(IGJG(IPLV(IC,JC,NL),JC,NL))
604                            >       +CO(2,IJ)*PHIT(IGJG(IMLV(IC,JC,NL),JC,NL)))
605                            >       +RES(IJ))
606         397         110     CONTINUE
608                             CALL TRIBI(AMJ,ACJ,APJ,FJJ,2,N2M,FJJ,2,N1G(NL),NL)
611           5                 DO 140 JC=1,N2M
612          17                 IC0=IIW(JC)
613                             IC1=NNW(JC)
614         137                 II=(IC0-1)*2**FLOAT(-NL+1)+1
615         109                 NN=(IC1)*2**FLOAT(-NL+1)
616         133                 DO 140 IC=II+1,NN,2
618        4404                 IJ=IGJG(IC,JC,NL)
619         700                 PHIT(IJ)=FJJ(JC,IC)
620         589         140     CONTINUE
```

<최적화/병렬화 코드: itsolv.f 계속>

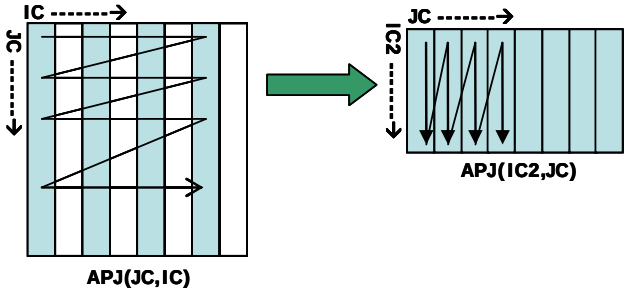
```
 66                                DO 110 JC=1,N2M
 67                                IC0=IIW(JC)
 68                                IC1=NNW(JC)
 69                                II=ISHFT(IC0-1,-NL+1)+1
 70                                NN=ISHFT(IC1,-NL+1)
 71                                NN1=(NN-II-1)/2+1
 72                                ICS=(II+1)/2
 75                                   IC=II+1
 76          167                   DO IC2=ICS,ICS+NN1-1
 77         1509                   APJ(IC2,JC)=CO1(3,IC,JC)
 78         2405                   ACJ(IC2,JC)=CO1(5,IC,JC)-AK3(K)/RM(JC)**2.
 79          291                   AMJ(IC2,JC)=CO1(4,IC,JC)
 80        14412                   FJJ(IC2,JC)
 81                               >   =(-(CO1(1,IC,JC)*PHIT1(IPLV1(IC,JC),JC)
 82                               >       +CO1(2,IC,JC)*PHIT1(IMLV1(IC,JC),JC))
 83                               >       +RES1(IC,JC))
 84                                IC=IC+2
 85          200                   ENDDO
 86           31          110     CONTINUE
 88            7                  CALL TRIBI(AMJ,ACJ,APJ,FJJ,2,N2M,FJJ,2,N1G(NL),NL)
 90            1                  DO 140 JC=1,N2M
 91                                IC0=IIW(JC)
 92                                IC1=NNW(JC)
 93                                II=ISHFT(IC0-1,-NL+1)+1
 94                                NN=ISHFT(IC1,-NL+1)
 95                                NN1=(NN-II-1)/2+1
 96                                ICS=(II+1)/2
 99                                   IC=II+1
100          104                   DO IC2=ICS,ICS+NN1-1
101         3377                   PHIT1(IC,JC)=FJJ(IC2,JC)
102                                IC=IC+2
103            5                   ENDDO
104            6          140     CONTINUE
```

③ Original 코드에서의 곱셈 A*2**FLOAT(X)를 ISHFT(A,X)로 대체하였다.

④ IC, JC의 함수 IGJG에 의해 계산되는 인덱스 IJ를 사용하지 않고 IC, JC를 배열 CO1, PHIT1, RES1에 직접 사용하였다.

⑤ Original 코드에서 루프 반복이 JC, IC 순으로 진행되는 반면 내부

에서 접근되는 배열들은 APJ(JC,IC), ACJ(JC,IC), AMJ(JC,IC), FJJ(JC,IC)로 구성돼 메모리 접근이 불연속적이다. 또한 IC 루프는 2씩 건너뛰며 반복되도록 구성돼 메모리 접근 불연속성은 더욱 그러하다. 최적화 코드에서는 아래 그림과 같이 데이터 접근이 이루어지도록 해서 메모리 접근 연속성을 높이고 있다.



< 그림 III.8 최적화 과정에서의 데이터 저장 변환 >

6.12 tribi.f

<Original 코드>

```
648          5          DO 5 I=MI,NI,2
649        352          BET(I)=B(1,I)
650       1103          U(1,I)=D(1,I)/BET(I)
651                  5  CONTINUE
653                     JS=JB2
655          6          DO 11 J=MJ,NJ
656                     IC0=IIV(J)
657         51          IC1=NNV(J)
659         40          IF(J.EQ.JS+1) THEN
660                     II=(IB1-1)*2**FLOAT(-NL+1)+1
661                     NN=(IB2-1)*2**FLOAT(-NL+1)
662          2          DO 6 I=II+MI-1,NN,2
663         18          BET(I)=B(JS,I)
664         84          U(JS,I)=D(JS,I)/BET(I)
665                  6  CONTINUE
666                     II=(IB3-1)*2**FLOAT(-NL+1)+1
667                     NN=(IB4-1)*2**FLOAT(-NL+1)
668                     DO 7 I=II+MI-1,NN,2
669         40          BET(I)=B(JS,I)
```

```
670          85          U(JS,I)=D(JS,I)/BET(I)
671                   7  CONTINUE
672                      ENDIF
674         270          II=(IC0-1)*2**FLOAT(-NL+1)+1
675         278          NN=(IC1)*2**FLOAT(-NL+1)
676         151          DO 21 I=II+MI-1,NN,2
677       27233          GAM(J,I)=C(J-1,I)/BET(I)
678         516          BET(I)=B(J,I)-A(J,I)*GAM(J,I)
679       58123          U(J,I)=(D(J,I)-A(J,I)*U(J-1,I))/BET(I)
680                  21  CONTINUE
681         251      11  CONTINUE
682           5          DO 12 J=NJ-1,MJ-1,-1
683          13          IC0=IIW(J+1) ! CAUTION NOT J !
684         163          IC1=NNW(J+1)
685         148          II=(IC0-1)*2**FLOAT(-NL+1)+1
686         202          NN=(IC1)*2**FLOAT(-NL+1)
687         226          DO 22 I=II+MI-1,NN,2
688       20157          U(J,I)=U(J,I)-GAM(J+1,I)*U(J+1,I)
689                  22  CONTINUE
690          61      12  CONTINUE
```

<최적화/병렬화 코드>

```
13                        J=1
14                     IC0=IIV(J)
15                     IC1=NNV(J)
16                     II=ISHFT(IC0-1,-NL+1)+1
17                     NN=ISHFT(IC1,-NL+1)
18                     IIS=(II+MI-1+1)/2
19                     MM=(NN-(II+MI-1))/2+1
22          21         DO I=IIS,IIS+MM-1
23                        BET=1./B(I,J)
24          55            GAM(I,J)=C(I,J)*BET
25          62            U(I,J)=D(I,J)*BET
26                     ENDDO
28                     JS=JB2
30           6         DO J=MJ,NJ
32           6         IF(J.EQ.JS+1) THEN
33                     II=ISHFT(IB1-1,-NL+1)+1
34                     NN=ISHFT(IB2-1,-NL+1)
35           1         DO I1=II+MI-1,NN,2
36                        I=(I1+1)/2
37                        BET=1./B(I,JS)
38          34            GAM(I,JS)=C(I,JS)*BET
39           2            U(I,JS)=D(I,JS)*BET
```

```
40                      ENDDO
41                      II=ISHFT(IB3-1,-NL+1)+1
42                      NN=ISHFT(IB4-1,-NL+1)
43                      DO I1=II+MI-1,NN,2
44                          I=(I1+1)/2
45                          BET=1./B(I,JS)
46          23              GAM(I,JS)=C(I,JS)*BET
47           3              U(I,JS)=D(I,JS)*BET
48           1          ENDDO
49                      ENDIF
51                      IC0=IIV(J)
52                      IC1=NNV(J)
53                      II=ISHFT(IC0-1,-NL+1)+1
54                      NN=ISHFT(IC1,-NL+1)
55                      IIS=(II+MI-1+1)/2
56                      MM=(NN-(II+MI-1))/2+1
59         115          DO I=IIS,IIS+MM-1
60        5771              BET=1./(B(I,J)-A(I,J)*GAM(I,J-1))
61        6173              GAM(I,J)=C(I,J)*BET
62       11589              U(I,J)=(D(I,J)-A(I,J)*U(I,J-1))*BET
63          49          ENDDO
64          40          ENDDO
65
66                      DO J=NJ-1,MJ-1,-1
67                      IC0=IIW(J+1) ! CAUTION NOT J !
68                      IC1=NNW(J+1)
69                      II=ISHFT(IC0-1,-NL+1)+1
70                      NN=ISHFT(IC1,-NL+1)
71                      IIS=(II+MI-1+1)/2
72                      MM=(NN-(II+MI-1))/2+1
75         176          DO I=IIS,IIS+MM-1
76        3663              U(I,J)=U(I,J)-GAM(I,J)*U(I,J+1)
77        1006          ENDDO
78          39          ENDDO
```

① tribi는 itsolv.f에서 호출되므로 itsolv에서 진행되었던 배열변환의 영향을 그대로 받는다. itsolv에서 변화된 AMJ, ACJ, APJ, FJJ 등이 tribi의 더미배열 A, B, C, D에 해당되고, 그에 따라 데이터 접근이 이뤄지도록 코드가 변경되었다. 즉, Original 코드에서 J, I 루프 반복 내에서 A(J,I), B(J,I), C(J,I), D(J,I), U(J,I)와 같이 접근되는 것을 A(I,J), B(I,J), C(I,J), D(I,J), U(I,J)로 접근되도록 코드를

변경하였다.

② Original 코드에서의 곱셈 A*2**FLOAT(X)를 ISHFT(A,X)로 대체하였다.

③ 임시 변수 BET는 배열을 이용하지 않고 스칼라 변수를 쓰도록 해서 불필요한 메모리 접근이 발생하는 것을 피하고 있고, 나눗셈 대신 역수의 곱셈을 사용하였다.

④ 마지막 I 루프 계산에서 2씩 건너뛰며 접근하는 것을 최적화 코드에서는 연속 접근이 이뤄지도록 코드를 수정하였다.

6.13 resca.f

<Original 코드>

```
729                      IGJG(I,J,L)=I+(J-1)*(N1G(L)+1)+N12G(L-1)
731                      DO 10 JC=1,N2M
732          6           IC0=IIW(JC)
733          2           IC1=NNW(JC)
734        144           II=(IC0-1)*2**FLOAT(-NL+1)+1
735        132           NN=(IC1)*2**FLOAT(-NL+1)
736         37           DO 20 IC=II,NN
738       8770           IJ=IGJG(IC,JC,NL)
739      35763           D2PHIJ=+(CO(1,IJ)
                        >            *PH(IGJG(IPLV(IC,JC,NL),JC,NL))+
740                     >            CO(2,IJ)*PH(IGJG(IMLV(IC,JC,NL),JC,NL))+
741                     >            CO(3,IJ)*PH(IGJG(IC,JPLV(IC,JC,NL),NL))+
742                     >            CO(4,IJ)*PH(IGJG(IC,JMLV(IC,JC,NL),NL))+
743                     >           (CO(5,IJ)-AK3(K)/RM(JC)**2.)*PH(IJ))
744        162           RS(IJ)=(-D2PHIJ+RST(IJ))
745         21      20    CONTINUE
746         40      10    CONTINUE
```

<최적화/병렬화 코드>

```
16                       DO 10 JC=1,N2M
17                       IC0=IIW(JC)
18                       IC1=NNW(JC)
19                       II=ISHFT(IC0-1,-NL+1)+1
20                       NN=ISHFT(IC1,-NL+1)
21         119           DO 20 IC=II,NN
22                       D2PHIJ=+(CO1(1,IC,JC)*PH1(IPLV1(IC,JC),JC)+
```

```
23                          >            CO1(2,IC,JC)*PH1(IMLV1(IC,JC),JC)+
24                          >            CO1(3,IC,JC)*PH1(IC,JPLV1(IC,JC))+
25                          >            CO1(4,IC,JC)*PH1(IC,JMLV1(IC,JC))+
26                          >           (CO1(5,IC,JC)-AK3(K)/RM(JC)**2.)
                            >            *PH1(IC,JC))
27          20710             RS1(IC,JC)=(-D2PHIJ+RST1(IC,JC))
28                    20      CONTINUE
29            32      10      CONTINUE
```

① Original 코드에서의 곱셈 A*2**FLOAT(X)를 ISHFT(A,X)로 대체하였다.

② IC, JC의 함수 IGJG에 의해 계산되는 인덱스 IJ를 사용하지 않고 IC, JC를 배열 CO1, PH1, RST1에 직접 사용하였다.

6.14 tfila.f

<Original 코드>

```
764                         IGJG(I,J,L)=I+(J-1)*(N1G(L)+1)+N12G(L-1)
766                         DO 10 JCL=1,N2M
767                         JCF=JCL
768           14            DO 10 ICL=1,N1G(NL+1)
770           610           ICF=ICL*2-1
771           3544          IJL=IGJG(ICL,JCL,NL+1)
772           245           IJF=IGJG(ICF,JCF,NL)
773                         IPJF=IJF+1
774           7404          REST(IJL)=(RES(IJF)*CO(6,IJF)+RES(IPJF)
                           >           *CO(6,IPJF))
775                        >            /CO(6,IJL)
776           507         10 CONTINUE
```

<최적화/병렬화 코드>

```
17                          DO 10 JCL=1,N2M
18                          JCF=JCL
19                          DO 10 ICL=1,N1G(NL+1)
21                          ICF=ICL*2-1
27          4435            REST2(ICL,JCL)=(RES1(ICF,JCF)*CO1(6,ICF,JCF)
28                         >             +RES1(ICF+1,JCF)*CO1(6,ICF+1,JCF))
29                          >           /CO2(6,ICL,JCL)
30            21          10 CONTINUE
```

① ICF, JCF의 함수 IGJG에 의해 계산되는 인덱스 IJF 또는 IJL를 사용하지 않고 ICF, JCF를 배열 CO1, REST2, RES1, CO2 등에 직접 사용하였다.

7. main.x Summary

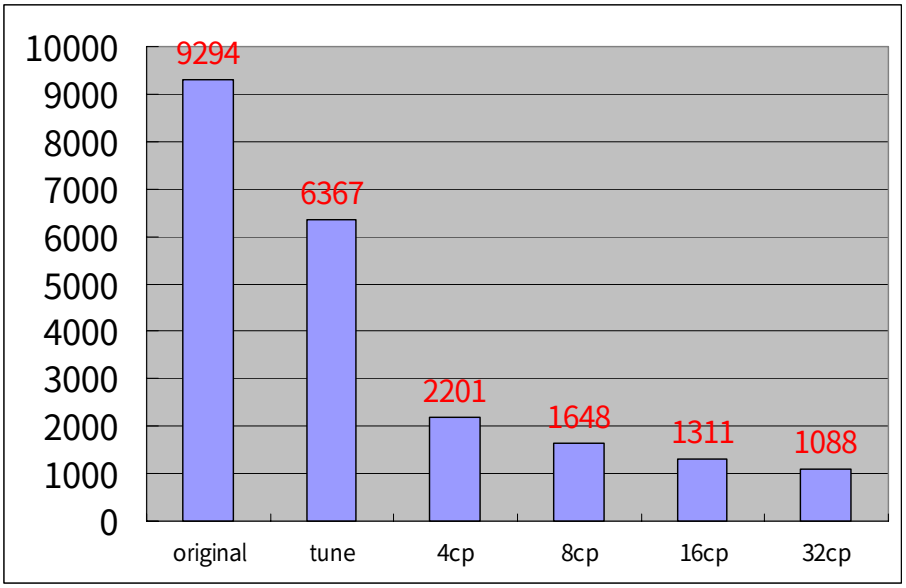
< 표III.3 main.x first step >

		Parallel Speed up
Original (Serial)	7416s	
Tuned 1cp	3989s	1.86
4 cp	1287s	5.76
8 cp	964s	7.69
16 cp	736s	10.00
32 cp	649s	11.42

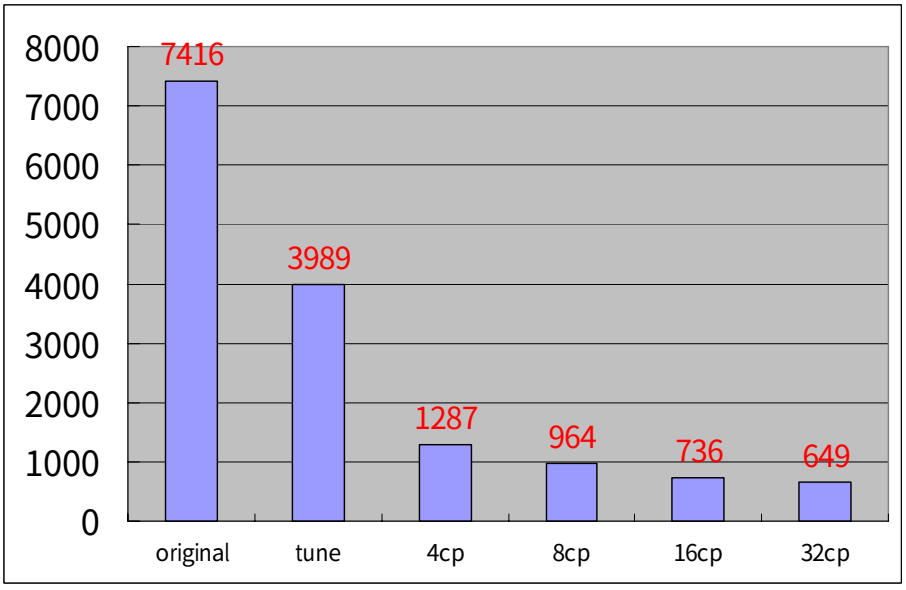
< 표 III.4 main.x second step >

		Parallel Speed up
Original (Serial)	9294.91s	
Tuned 1cp	6367s	1.46
4 cp	2201s	4.22
8 cp	1648s	5.63
16 cp	1311s	7.08
32 cp	1088s	8.54

Inflow와 비교해 조금 더 낫기는 하지만 main.x 역시도 최적화를 통해 얻게 된 성능향상 효과가 그렇게 크지는 않다. 이 부분 역시 inflow에서와 마찬가지로 주로 병렬화를 통해서 성능이득을 얻고 있다. 불필요한 인덱스 계산을 위한 함수 호출을 없애고 메모리 접근에 대한 연속성을 높여 캐시 실패를 줄이도록 코드를 최적화 하였다. Inflow에서와 마찬가지로 IMSL 루틴에 의한 FFT 계산을 IBM 시스템에 최적화된 ESSL의 FFT 루틴을 사용함으로써 주요한 성능향상 효과를 얻고 있다.



< 그림 III. 9 main.x first step에서 최적화 및 병렬화 성능 비교 >



< 그림 III.10 main. x second step 에서 최적화 및 병렬화 성능 비교 >

CHAPTER IV. 조선대 김재수 교수 코드

1. Code information

이 코드는 아음속 및 초음속에서 유동의 압력장 및 소음 방사를 계산하는 CFD 코드로 평판에서의 유동현상과 소음의 특성을 알아보기 위한 코드이다. 지배 방정식으로는 Navier-Stokes Equation을 사용하였고 이 방정식을 풀기 위해서 고차 집적유한 차분법을 사용하였다. 고차 집적유한 차분법을 풀기 위해서 최적화된 4차-order를 pentadiagonal을 사용하였고 Runge-Kutta 시간 적분법을 이용했는데 4차-order를 갖는 Runge-Kutta를 사용다. 그리고 일반화된 특성치 경계 조건을 사용하였고 인공 감쇄항을 이용하여 코드의 안정화를 시켰다.

2. Makefile

< Original 코드의 Makefile >

```
.SUFFIXES: .f .o
F90 = xlf

SRC = a_bound_sub_far.f a_datain.f a_diss_epsil4.f a_initcotom.f ₩
      a_outputtest.f a_outputtest_turb.f ₩
      acoustics.f acoustics_sub_060131.f bound_charact_20060530.f ₩
      grid.f jacobi_3d.f source_convec_20060707.f

OBJ = $(SRC:f=o)

EXE      = acous.x
LIB      = -lmassv -lmass
INFO     = -pg -g
LDINFO   = -pg
F90FLAG = -q64 -O3 -qtune=pwr4 -qarch=pwr4 -qcache=auto $(INFO)
LDFLAG   = -q64 $(LIB) $(LDINFO)

$(EXE): $(OBJ)
      $(F90) -o $@ $(LDFLAG) $(OBJ)
.f.o: a_vcom_cav_rk.inc a_vcom_param.f
      $(F90) -c $(F90FLAG) $<
clean:
      rm -f $(OBJ) $(EXE) *.lst
```

< Serial 최적화 코드의 Makefile >

```
.SUFFIXES: .f .o
F90 = xlf

SRC = a_bound_sub_far.f a_datain.f a_diss_epsil4.f a_initcotom.f ₩
      a_outputtest.f a_outputtest_turb.f ₩
      acoustics.f acoustics_sub_060131.f grid.f jacobi_3d.f ₩
      deri_xi_total.f deri_et_total.f deri_zt_total.f turbkom.f ₩
      deri_xi_total_5.f deri_et_total_5.f deri_zt_total_5.f₩
      deri_xi_total_4.f deri_et_total_4.f deri_zt_total_4.f₩
      non_reflect_wall.f p_and_pinv_charact.f deriv_normal_surface.f ₩
      condition_surface.f charc_convec_term.f non_reflect_far_wall.f ₩
      non_reflect_far.f a_flux_bound.f ₩
      matrix_source.f a_bound_total.f bound_freestream.f bound_extrap.f ₩
```

```
        bound_symmetric.f

OBJ = $(SRC:f=o)

EXE        = acous.x
LIB        = -lmassv -lmass
INFO       = -pg -g
LDINFO     = -pg
F90FLAG = -q64 -O3 -qtune=pwr4 -qarch=pwr4 -qcache=auto $(INFO)
LDFLAG     = -q64 $(LIB) $(LDINFO)

$(EXE): $(OBJ)
        $(F90) -o $@ $(LDFLAG) $(OBJ)
.f.o: a_vcom_cav_rk.inc a_vcom_param.inc
        $(F90) -c $(F90FLAG) $<
clean:
        rm -f $(OBJ) $(EXE)
```

< MPI 코드의 Makefile >

```
.SUFFIXES: .f .o
F90 = mpxlf

SRC = a_bound_sub_far.f a_datain.f a_diss_epsil4.f a_initcotom.f ₩
        a_outputtest.f a_outputtest_turb.f ₩
        acoustics.f acoustics_sub_060131.f bound_charact_20060530.f ₩
        grid.f jacobi_3d.f source_convec_20060707.f ₩
        deri_xi_total.f deri_et_total.f deri_zt_total.f turbkom.f ₩
        deri_xi_total_5.f deri_et_total_5.f deri_zt_total_5.f ₩
        deri_xi_total_4.f deri_et_total_4.f deri_zt_total_4.f ₩
        a_flux.f a_vsflux.f comm_subrt.f a_disspation.f ₩
        a_flux_bound.f a_bound_total.f outputtest.f

OBJ = $(SRC:f=o)

EXE        = acous.x
LIB        = -lmassv -lmass -L/applic/mpitrace/lib -lmpitrace
INFO       = -pg -g
LDINFO     = -pg
F90FLAG = -q64 -O3 -qtune=pwr4 -qarch=pwr4 -qcache=auto $(INFO)
LDFLAG     = -q64 -O3 $(LIB) $(LDINFO)

$(EXE): $(OBJ)
          $(F90) -o $@ $(LDFLAG) $(OBJ)
.f.o: a_vcom_cav_rk.inc a_vcom_param.inc mpi_variables.inc
          $(F90) -c $(F90FLAG) $<
clean:
```

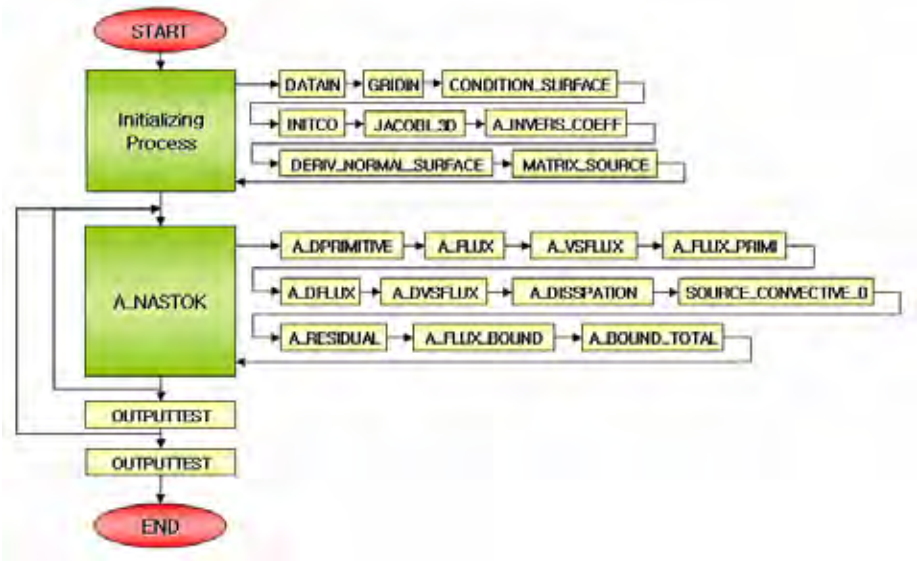
```
	rm -f $(OBJ) $(EXE)
```

Original 코드, Serial 최적화된 코드 및 MPI 코드의 Makefile에서 동일한 컴파일 및 링크 옵션을 사용하고 있으며, 프로파일링을 위해 -pg -g 및 -pg 옵션이 컴파일 및 링크할 때 각각 추가되어 있다.

MPI 코드에서는 통신량을 확인하기 위해 mpitrace 라이브러리를 링크할 때 추가하였다.

3. Flow Chart

아래의 Flow Chart 그림과 같이 Initializing process를 거쳐 A_NASTOCK 루틴에서 main routine들이 계산되어 진다. A_NASTOK 루틴부분에서 4번의 iteration을 수행하고 outputtest에서 그 결과를 한번 write 한 후 다시 전체 main iteration이 수행되는 형태를 가진다.



< 그림 IV.1 Flowchart >

4. Profiling 결과

<Original 코드의 profiling 결과>

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
37.9	2473.54	2473.54	11200	220.85	220.85	.deri_xi_total [4]
28.2	4315.64	1842.10	11200	164.47	164.47	.deri_zt_total [7]
21.4	5715.69	1400.05	11200	125.00	125.00	.deri_et_total [9]
6.4	6134.05	418.36	800	522.95	522.95	.a_disspation [10]
2.3	6282.97	148.92	800	186.15	186.15	.a_vsflux [11]
0.9	6339.61	56.64	1	56640.00	6486630.00	._main [1]
0.5	6373.63	34.02				._pow [12]
0.4	6402.74	29.11	800	36.39	36.39	.a_flux [13]
0.4	6431.20	28.46	800	35.58	2587.22	.a_dflux [5]
0.4	6459.29	28.09	800	35.11	2076.43	.a_dvsflux [8]
0.3	6476.50	17.21	800	21.51	21.51	.a_flux_primi [14]
0.2	6488.47	11.97	800	14.96	14.96	.a_residual [16]
0.2	6498.64	10.17	800	12.71	8034.13	.a_nastok [3]
0.1	6503.40	4.76	800	5.95	5.95	.source_convective_0 [17]
0.1	6507.10	3.70	9711200	0.00	0.00	.charc_convec_term [19]
0.1	6510.62	3.52	9711200	0.00	0.00	.p_and_pinv_charact [20]
0.0	6513.19	2.57	800	3.21	3.21	.a_flux_bound [24]
0.0	6515.48	2.29				_tanh [26]
0.0	6516.93	1.45	4194692	0.00	0.00	.cvtloop [29]
0.0	6518.37	1.45				.__mcount [30]
0.0	6519.70	1.33	2879200	0.00	0.00	.non_reflect_far_wall [21]
0.0	6520.65	0.95	3952800	0.00	0.00	.non_reflect_wall [18]
0.0	6521.59	0.94	1279719	0.00	0.00	.deri_zt [31]
0.0	6522.51	0.92	800	1.15	14.96	.a_bound_total [15]
0.0	6523.35	0.84				.__mcount [32]
0.0	6524.16	0.81	12290400	0.00	0.00	.bound_extrap [33]
0.0	6524.90	0.74	1261419	0.00	0.00	.deri_xi [34]
0.0	6525.47	0.57	2879200	0.00	0.00	.non_reflect_far [23]
0.0	6526.02	0.55	4448691	0.00	0.00	.__cvt_r [27]
0.0	6526.57	0.55	1279719	0.00	0.00	.deri_et [35]

<Serial 최적화 코드의 profiling 결과>

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
20.4	551.84	551.84	800	689.80	689.80	.deri_zt_total_5 [3]
15.9	981.86	430.02	3200	134.38	134.38	.deri_zt_total [4]
15.2	1391.86	410.00	800	512.50	512.50	.deri_zt_total_4 [5]
7.5	1596.09	204.23	800	255.29	255.29	.deri_xi_total_5 [6]

7.5	1799.23	203.14	3200	63.48	63.48	.deri_xi_total [7]
6.0	1962.59	163.36	800	204.20	204.20	.deri_xi_total_4 [8]
5.6	2113.87	151.28	800	189.10	189.10	.a_disspation [9]
5.4	2261.16	147.29	800	184.11	184.11	.a_vsflux [10]
4.4	2380.26	119.10	800	148.88	148.88	.deri_et_total_5 [11]
3.8	2483.51	103.25	3200	32.27	32.27	.deri_et_total [12]
3.5	2579.23	95.72	800	119.65	119.65	.deri_et_total_4 [13]
1.3	2615.05	35.82				._pow [14]
1.3	2649.11	34.06	1	34060.00	2659803.33	.main [1]
1.1	2677.58	28.47	800	35.59	35.59	.a_flux [15]
0.1	2681.50	3.92	9711200	0.00	0.00	.charc_convec_term [18]
0.1	2685.05	3.55	9711200	0.00	0.00	.p_and_pinv_charact [20]
0.1	2687.46	2.41				.tanh [23]
0.1	2689.44	1.98	800	2.48	2.48	.a_flux_bound [27]
0.1	2691.21	1.77	2879200	0.00	0.00	.non_reflect_far_wall [17]
0.1	2692.60	1.39				.__mcount [28]
0.0	2693.84	1.24	4194497	0.00	0.00	.cvtloop [29]
0.0	2694.94	1.10	2879200	0.00	0.00	.non_reflect_far [21]
0.0	2695.91	0.97	800	1.21	16.36	.a_bound_total [16]
0.0	2696.85	0.94	1279719	0.00	0.00	.deri_zt [30]
0.0	2697.69	0.84	12290400	0.00	0.00	.bound_extrap [31]
0.0	2698.53	0.84	4448691	0.00	0.00	.__cvt_r [25]
0.0	2699.28	0.75	1261419	0.00	0.00	.deri_xi [32]
0.0	2699.97	0.69	3952800	0.00	0.00	.non_reflect_wall [19]
0.0	2700.62	0.65				.__mcount [33]
0.0	2701.20	0.58	1279719	0.00	0.00	.deri_et [34]

Original 코드와 Serial 최적화 코드의 프로파일링 결과에서 각각 대응되는 서브루틴들을 비교해보면 수행시간이 많이 줄어든 것을 확인해 볼 수 있다. Original 코드에서 하나의 루틴이 Serial 최적화 코드에서는 몇 개의 루틴으로 나누어 지는 경우 각각을 합쳐서 비교해 볼 수 있다. 상대적으로 hot spot 부분에 대한 성능향상이 뚜렷하게 나타남을 확인해 볼 수 있다.

<MPI 코드의 profiling 결과>

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
10.8	15.07	15.07	800	18.84	18.84	.deri_zt_total_5 [3]
9.3	27.96	12.89	800	16.11	16.11	.jk45_bcast [4]
9.1	40.59	12.63				._barrier_onnode [5]
8.5	52.46	11.87	3200	3.71	3.71	.deri_zt_total [6]
8.4	64.13	11.67				._tag_waiting [7]
8.2	75.58	11.45	800	14.31	14.31	.deri_zt_total_4 [8]
4.5	81.82	6.24	800	7.80	7.86	.a_disspation [9]
4.2	87.64	5.82	800	7.28	7.28	.deri_xi_total_5 [10]

4.1	93.28	5.64	3200	1.76	1.76	.deri_xi_total [11]
3.4	98.04	4.76	800	5.95	5.95	.deri_xi_total_4 [12]
3.1	102.42	4.38	800	5.47	5.47	.a_vsflux [13]
3.0	106.55	4.13	800	5.16	5.16	.deri_et_total_5 [14]
2.6	110.15	3.60	800	4.50	4.50	.bcast_cross [15]
2.6	113.71	3.56	3200	1.11	1.11	.deri_et_total [16]
2.3	116.89	3.18	800	3.98	3.98	.deri_et_total_4 [17]
2.2	119.97	3.08				.REG_3stream_store [18]
1.7	122.35	2.38				._cntr_waiting [19]
1.0	123.77	1.42	1	1420.00	101010.00	.main [1]
1.0	125.14	1.37				pow [21]
0.9	126.43	1.29	1	1290.00	1290.00	.jacobi_3d [22]
0.8	127.48	1.05				.REG_fmemcpy [23]
0.7	128.41	0.93	1279719	0.00	0.00	.deri_zt [24]
0.6	129.20	0.79	1	790.00	790.00	.initco [25]
0.6	129.98	0.78	800	0.97	0.97	.a_flux [26]
0.6	130.75	0.77				.icopy [27]
0.5	131.47	0.72				._lapi_shm_dispatcher [28]
0.5	132.15	0.68	1261419	0.00	0.00	.deri_xi [29]
0.5	132.82	0.67	800	0.84	0.84	.mpi_exchange [30]
0.4	133.41	0.59	1279719	0.00	0.00	.deri_et [32]
0.3	133.82	0.41				.icopy [33]
0.3	134.19	0.37	800	0.46	0.46	.a_flux_bound [34]

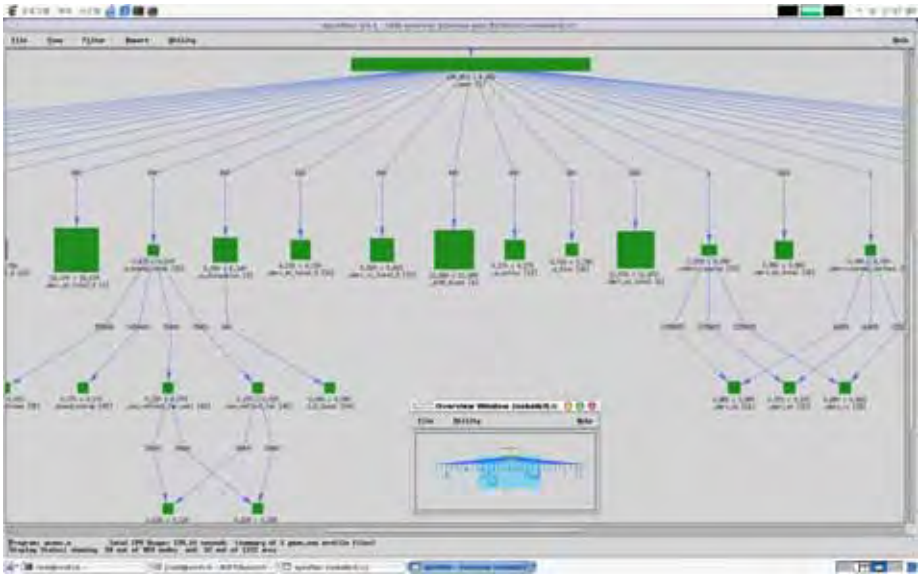
MPI task를 32개로 설정한 후 수행한 결과이며, 0번 task에 대한 프로파일링 정보이다.

MPI 병렬화를 위해 Serial 코드에서는 없는 루틴들이 몇몇 추가되었으며, 전체적으로 수행시간이 많이 줄어든 것을 확인해 볼 수 있다.

다음 그림은 xprofiler 결과를 보여준다.

< 그림 IV. 2 Original 코드의 xprofiler 결과 >

< 그림 IV.3 Serial 최적화 코드의 xprofiler 결과 >



< 그림 IV.4 MPI 코드의 xprofiler 결과 >

MPI task를 32개로 설정한 후 수행했을 때 0번 rank의 프로파일링 결과이다.

5. hpmcount 결과

컴파일된 실행파일을 hpmcount와 같이 수행할 때 나오는 코드 수행에 관한 정보는 다음과 같다.

<Original 코드의 profiling 결과>

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 6530.646774 seconds

########  Resource Usage Statistics  ########

Total amount of time in user mode            : 6529.630000 seconds
Total amount of time in system mode          : 0.620000 seconds
Maximum resident set size                    : 362972 Kbytes
Average shared memory use in text segment    : 8358656 Kbytes*sec
Average unshared memory use in data segment  : 2147483647 Kbytes*sec

#######  End of Resource Statistics  ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                    :        3290015422
 PM_FPU_FMA (FPU executed multiply-add instruction)             :     3396743329068
 PM_FPU0_FIN (FPU0 produced a result)                           :     4351276262321
 PM_FPU1_FIN (FPU1 produced a result)                           :     4470149367603
 PM_CYC (Processor cycles)                                      :    11114169629676
 PM_FPU_STF (FPU executed store instruction)                    :      610602717637
 PM_INST_CMPL (Instructions completed)                          :    14610403329944
 PM_LSU_LDF (LSU executed Floating Point load instruction)      :     4772979505964

 Utilization rate                          :          99.874 %
 Load and store operations                 :       5383582.224 M
 MIPS                                      :          2237.206
 Instructions per cycle                    :             1.315
 HW Float points instructions per Cycle    :             0.794
 Floating point instructions + FMAs        :     11607566.241 M
 Float point instructions + FMA rate       :       1777.399 Mflip/s
 FMA percentage                            :            58.526 %
 Computation intensity                     :             2.156
```

<Serial 최적화 코드의 profiling 결과>

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 2706.002647 seconds

########  Resource Usage Statistics  ########

Total amount of time in user mode            : 2704.660000 seconds
Total amount of time in system mode          : 0.690000 seconds
Maximum resident set size                    : 408048 Kbytes
Average shared memory use in text segment    : 4144458 Kbytes*sec
Average unshared memory use in data segment  : 1063705806 Kbytes*sec

#######  End of Resource Statistics  ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                    :        2917281462
 PM_FPU_FMA (FPU executed multiply-add instruction)             :     1670430773763
 PM_FPU0_FIN (FPU0 produced a result)                           :     2123059997345
 PM_FPU1_FIN (FPU1 produced a result)                           :     1877225308566
 PM_CYC (Processor cycles)                                      :     4601704916834
 PM_FPU_STF (FPU executed store instruction)                    :     1013761945982
 PM_INST_CMPL (Instructions completed)                          :     7778800569857
 PM_LSU_LDF (LSU executed Floating Point load instruction) :     3173304870256

 Utilization rate                      :           99.798 %
 Load and store operations             :      4187066.816 M
 MIPS                                  :           2874.646
 Instructions per cycle                :           1.690
 HW Float points instructions per Cycle :          0.869
 Floating point instructions + FMAs    :      4656954.134 M
 Float point instructions + FMA rate   :      1720.972 Mflip/s
 FMA percentage                        :           71.739 %
 Computation intensity                 :           1.112
```

Original 코드와 Serial 최적화 코드를 비교해 보면 Serial 최적화 코드에서 부동소수점 연산속도(FLOPS)는 조금 낮지만, 부동소수점 연산회수가 절반 이하가 되면서 전체적으로 수행시간도 2배이상 빨라졌음을 확인할 수 있다.

<MPI 코드의 profiling 결과>

```
hpmcount (V 2.4.3) summary
```

```
Total execution time (wall clock time): 154.303459 seconds

########  Resource Usage Statistics  ########

Total amount of time in user mode                 : 136.160000 seconds
Total amount of time in system mode               : 5.580000 seconds
Maximum resident set size                         : 208804 Kbytes
Average shared memory use in text segment         : 46510 Kbytes*sec
Average unshared memory use in data segment       : 28531716 Kbytes*sec

#######  End of Resource Statistics  ########

 PM_FPU_FDIV (FPU executed FDIV instruction)                    :        128953266
 PM_FPU_FMA (FPU executed multiply-add instruction)             :      49084941888
 PM_FPU0_FIN (FPU0 produced a result)                           :      65261214934
 PM_FPU1_FIN (FPU1 produced a result)                           :      57253171492
 PM_CYC (Processor cycles)                                      :     231011282758
 PM_FPU_STF (FPU executed store instruction)                    :      33139817428
 PM_INST_CMPL (Instructions completed)                          :     322627932432
 PM_LSU_LDF (LSU executed Floating Point load instruction)      :      97117064050

 Utilization rate                                :          87.859 %
 Load and store operations                       :      130256.881 M
 MIPS                                            :        2090.866
 Instructions per cycle                          :           1.397
 HW Float points instructions per Cycle          :           0.530
 Floating point instructions + FMAs              :      138459.511 M
 Float point instructions + FMA rate             :      897.320 Mflip/s
 FMA percentage                                  :          70.902 %
 Computation intensity                           :           1.063
```

MPI task를 32개로 설정한 후 수행했을 때 0번 rank의 프로파일링 결과이다. Serial 최적화 코드 대비 부동소수점 연산속도(FLOPS)가 많이 줄어 들었지만 영역분할 후 해당부분만 연산을 하기 때문에 부동소수점 연산회수가 영역분할 수로 나눈만큼 줄어들었음을 확인할 수 있고 그에 따라 수행시간도 빨라졌음을 확인할 수 있다. 전체적으로 부동소수점 연산속도(FLOPS)가 줄어든 이유는 상당한 크기의 통신량이 큰 이유가 될 것이다.

6. 최적화 코드 비교 및 분석

다음 subroutine들에 대해서 최적화를 진행하였다.

< 표 IV.1 서브루틴 별 최적화 tick수 비교 및 speed-up >

	Subroutine 명	Original 코드의 tick 수	최적화 코드의 tick 수	Speed-up
1	acoustics	5664	3406	1.66
2	jacobi_3d	6	26	0.23
3	a_nastok	1017	-	-
4	deri_xi_total	247354	57073	4.33
5	deri_zt_total	184210	139186	1.32
6	deri_et_total	140005	31807	4.40
7	a_vsflux	14892	14729	1.01
8	a_dflux	2846	-	-
9	a_dvsflux	2809	-	-
10	a_disspation	41836	15128	2.77
total	-	640639	261355	2.45

여기서 tick 수는 해당 subroutine의 대략적인 수행시간을 나타내는 것으로 1 tick당 0.01초를 나타내며 speed-up은 해당 코드에 대해 최적화 작업으로 빨라진 비율을 나타 낸다. 최적화 과정에서 수정된 subroutine들에 대한 결과를 비교한 것인데, 상대적으로 수행시간이 긴 hot spot 부분에서 많은 시간이 단축되어 전체적으로 대략 2.45배 정도의 성능이 향상되었음을 확인할 수 있다.

각 subroutine에 대해서 하나씩 분석해 보면 다음과 같다.

6.1 acoustics 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
 74                          DO 100 i=1,i2
 75                          DO 100 j=1,j2
 76                          DO 100 k=1,k2
 77          597              ur(i,j,k)=w(2,i,j,k)/w(1,i,j,k)
 78         1619              vr(i,j,k)=w(3,i,j,k)/w(1,i,j,k)
 79          924              wr(i,j,k)=w(4,i,j,k)/w(1,i,j,k)
 80         1868              p(i,j,k)=(w(5,i,j,k)-0.5*w(1,i,j,k)*(ur(i,j,k)**2
 81                          *    +vr(i,j,k)**2+wr(i,j,k)**2))*(gamm)
 82          556              t(i,j,k)=p(i,j,k)/w(1,i,j,k)
 83           13         100 continue
```

<Serial 최적화 코드>

```
163                          DO 100 k=1,k2
164                          DO 100 j=1,j2
165                          DO 100 i=1,i2
166           78              wrec=1.0/w(1,i,j,k)
167          164              ur(i,j,k)=w(2,i,j,k)*wrec
168          153              vr(i,j,k)=w(3,i,j,k)*wrec
169          219              wr(i,j,k)=w(4,i,j,k)*wrec
170          154              p(i,j,k)=(w(5,i,j,k)-0.5*w(1,i,j,k)*(ur(i,j,k)**2
171                          *    +vr(i,j,k)**2+wr(i,j,k)**2))*(gamm)
172           53              t(i,j,k)=p(i,j,k)*wrec
173            2         100 continue
```

⑤ Original 코드에서 비연속적인 메모리 access를 피하기 위해서 inner loop와 outer loop의 수행순서를 바꾸어 연속적인 메모리 access를 하도록 수정하였다.

⑥ 계산 load가 높은 나누기 연산을 피하기 위해 1.0/w(1,i,j,k)을 wrec로 저장한 후 이를 곱해주도록 하였다.

6.2 Jacobi_3d 루틴

Main 계산에서 반복되는 계산부분을 미리 수행

<Serial 최적화 코드>

```
 93                          do k=1,k2
 94                          do j=1,j2
 95                          do i=1,i2
102                       ai_arr(i,j,k)=sqrt(xix(i,j,k)**2+xiy(i,j,k)**2+xiz(i,j,k)**2)
103      1                aj_arr(i,j,k)=sqrt(etx(i,j,k)**2+ety(i,j,k)**2+etz(i,j,k)**2)
104      2                ak_arr(i,j,k)=sqrt(ztx(i,j,k)**2+zty(i,j,k)**2+ztz(i,j,k)**2)
105                          enddo
106                          enddo
107                          enddo

110                          do k=1,k2
111                          do j=1,j2
112                          do i=1,i2
114                            six=xix(i,j,k)
115                            siy=xiy(i,j,k)
116                            siz=xiz(i,j,k)
117                            sjx=etx(i,j,k)
118                            sjy=ety(i,j,k)
119              1             sjz=etz(i,j,k)
120                            skx=ztx(i,j,k)
121                            sky=zty(i,j,k)
122                            skz=ztz(i,j,k)
124              1             abcdef(1 ,i,j,k) = 4./3.*six**2 +siy**2 +siz**2
125                            abcdef(2 ,i,j,k) = six**2 +4./3.*siy**2 +siz**2
126                            abcdef(3 ,i,j,k) = six**2 +siy**2 +4./3.*siz**2
127                            abcdef(4 ,i,j,k) = six**2 +siy**2 +siz**2
128                            abcdef(5 ,i,j,k) = six*siy/3.
129                            abcdef(6 ,i,j,k) = siy*siz/3.
130                            abcdef(7 ,i,j,k) = six*siz/3.

170                            abcdef(42,i,j,k) = 4./3.*sjx*skx+sjy*sky+sjz*skz
171                            abcdef(43,i,j,k) = sjx*skx+4./3.*sjy*sky+sjz*skz
172                            abcdef(44,i,j,k) = sjx*skx+sjy*sky+4./3.*sjz*skz
173              1             abcdef(45,i,j,k) = sjx*skx+sjy*sky+sjz*skz
174                            abcdef(46,i,j,k) = sjx*sky-2./3.*sjy*skx
175                            abcdef(47,i,j,k) = sjx*skz-2./3.*sjz*skx
176                            abcdef(48,i,j,k) = sjy*skx-2./3.*sjx*sky
177                            abcdef(49,i,j,k) = sjy*skz-2./3.*sjz*sky
178                            abcdef(50,i,j,k) = sjz*skx-2./3.*sjx*skz
179                            abcdef(51,i,j,k) = sjz*sky-2./3.*sjy*skz
180
181                          enddo
```

```
182                         enddo
183                         enddo
```

① Main iteration 과정에서 반복되는 계산 부분을 미리 수행하여 ai_arr, aj_arr, ak_arr, abcdef array에 저장하는 과정을 추가하였다.

6.3 a_nastok 루틴

반복계산 삭제

<Original 코드>

```
167                         DO 60 K=1,K2
168                         do 60 j=1,j2
169                         do 60 i=1,i2
170                         do ir=1,5
171     811           W(ir,I,J,K)   =wn(ir,i,j,k)+rungefactor/vol(i,j,k)*DW(ir,I,J,K)
172                         enddo
173                      60 CONTINUE

176                         RTRMS       = 0.
177                         RTMAX       = 0.
178                         NSUP        = 0
179                         DO 90 K=2,KL
180                         DO 90 J=2,JL
181                         DO 90 I=2,IL
182                         do ir=1,5
183              81         RT          = DW(ir,i,J,K)
184             125         RTRMS(ir)       = RTRMS(ir)+RT**2
185                         enddo
186                      90 CONTINUE
```

<Serial 최적화 코드>

```
133                         DO 60 K=1,K2
134                         Do 60 j=1,j2
135                         Do 60 i=1,i2
136               3          runge_volres=rungefactor*volres(i,j,k)
137                          do ir=1,5
138             634          W(ir,I,J,K)   =wn(ir,i,j,k)+runge_volres*DW(ir,I,J,K)
139                          enddo
```

```
140                      60 CONTINUE
141
142                         RTRMS = 0.
143
144                         DO 90 K=2,KL
145                         DO 90 J=2,JL
146                         DO 90 I=2,IL
147                         do ir=1,5
148         219           RTRMS(ir)       = RTRMS(ir)+DW(ir,I,J,K)*DW(ir,I,J,K)
149                         enddo
150                      90 CONTINUE
```

① 반복계산을 피하기 위해 runge_volres에 rungefactor*volres(I,j,k) 값을 저장한 후 이를 그다음 계산에 사용하였다.

6.4 a_dprimitive 루틴

A. deri_xi_total, deri_et_total, deri_zt_total 루틴을 call

<Original 코드>

```
515                         call deri_xi_total(ur,dur_xi,i2,j2,k2)
516                         call deri_xi_total(vr,dvr_xi,i2,j2,k2)
517                         call deri_xi_total(wr,dwr_xi,i2,j2,k2)
518                         call deri_xi_total(p,dp_xi,i2,j2,k2)
519                         call deri_xi_total(t,dt_xi,i2,j2,k2)
520
521
522                         call deri_et_total(ur,dur_et,i2,j2,k2)
523                         call deri_et_total(vr,dvr_et,i2,j2,k2)
524                         call deri_et_total(wr,dwr_et,i2,j2,k2)
525                         call deri_et_total(p,dp_et,i2,j2,k2)
526                         call deri_et_total(t,dt_et,i2,j2,k2)
527
528
529           1             call deri_zt_total(ur,dur_zt,i2,j2,k2)
530                         call deri_zt_total(vr,dvr_zt,i2,j2,k2)
531                         call deri_zt_total(wr,dwr_zt,i2,j2,k2)
532                         call deri_zt_total(p,dp_zt,i2,j2,k2)
533                         call deri_zt_total(t,dt_zt,i2,j2,k2)
```

<Serial 최적화 코드>

```
74                          call deri_xi_total(ur,dur_xi)
75                          call deri_xi_total(vr,dvr_xi)
76                          call deri_xi_total(wr,dwr_xi)
77                          call deri_xi_total(t,dt_xi)
78
79                          call deri_et_total(ur,dur_et)
80                          call deri_et_total(vr,dvr_et)
81                          call deri_et_total(wr,dwr_et)
82                          call deri_et_total(t,dt_et)
83
84                          call deri_zt_total(ur,dur_zt)
85                          call deri_zt_total(vr,dvr_zt)
86                          call deri_zt_total(wr,dwr_zt)
87                          call deri_zt_total(t,dt_zt)
```

① deri_xi_total, deri_et_total, deri_zt_total 함수에 대해서 p array를 이용하여 계산하는 과정은 불필요하기 때문에 Serial 최적화 코드에서는 삭제하였으며, 동시에 main 루틴인 acoustics 루틴에 inline시켰다.

6.5 deri_xi_total 루틴

A. do-loop 분리 및 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
230                         il=i2-1
231                         do 101 k=1,k2
232           66            do 101 j=1,j2
233           37            do 101 i=1,i2
234                         is=1
235            9            duu(i,j,k)=a_inv_xi(i,is)*
236                   * (a_a12*(uu(2,j,k)-uu(1,j,k))+a_a13*(uu(3,j,k)-uu(1,j,k))
237                   * +a_a14*(uu(4,j,k)-uu(1,j,k)))

250         3461            do is=4,il-2
251       228660              duu(i,j,k)=duu(i,j,k)+a_inv_xi(i,is)
252                          * *(a_c*(uu(is+3,j,k)-uu(is-3,j,k))
253                          *   +a_b*(uu(is+2,j,k)-uu(is-2,j,k))
254                          *   +a_a*(uu(is+1,j,k)-uu(is-1,j,k)))
255                         enddo
```

```
263                             is=il
264          4323                 duu(i,j,k)=duu(i,j,k)-a_inv_xi(i,is)*
265                 * (a_a21*(uu(i2,j,k)-uu(il,j,k))+a_a23*(uu(il-1,j,k)-uu(il,j,k))
266                * +a_a24*(uu(il-2,j,k)-uu(il,j,k))+a_a25*(uu(il-3,j,k)-uu(il,j,k)))
267                             is=i2
268          2104                 duu(i,j,k)=duu(i,j,k)-a_inv_xi(i,is)*
269                            * (a_a12*(uu(il,j,k)-uu(i2,j,k))+a_a13*(uu(il-1,j,k)-
270                            * uu(i2,j,k))+a_a14*(uu(il-2,j,k)-uu(i2,j,k)))
271           341        101    continue
272                             return
```

<Serial 최적화 코드>

```
 7                              il=i2-1
 8                              do k=1,k2
 9                              do j=1,j2
10            12                duu_is(1)=(a_a12*(uu(2,j,k)-uu(1,j,k))+a_a13
11                             & *(uu(3,j,k)-uu(1,j,k))+a_a14*(uu(4,j,k)-uu(1,j,k)))

25                             duu_is(6)=(a_a12*(uu(il,j,k)-uu(i2,j,k))
26                             & +a_a13*(uu(il-1,j,k)-uu(i2,j,k))+a_a14
                                 *(uu(il-2,j,k)-uu(i2,j,k)))
27
28                             do i=1,i2
29            597              duu(i,j,k)=a_inv_xi(i,1)*duu_is(1)
30                             &   +a_inv_xi(i,2)*duu_is(2)+a_inv_xi(i,3)*duu_is(3)
31                             & -a_inv_xi(i,i2-2)*duu_is(4)-a_inv_xi(i,i2-1)
                                 *duu_is(5)-a_inv_xi(i,i2)*duu_is(6)
32                             enddo
33
34              5              do is=4,i2-3
35            768              uu_temp=a_c*(uu(is+3,j,k)-uu(is-3,j,k))
36                             &   +a_b*(uu(is+2,j,k)-uu(is-2,j,k))
37                             &   +a_a*(uu(is+1,j,k)-uu(is-1,j,k))
38                             do i=1,i2
39          18699                duu(i,j,k)=duu(i,j,k)+a_inv_xi(i,is)*uu_temp
40                             enddo
41            146              enddo
42
43              4              enddo
44              1              enddo
```

① Original 코드에서 I loop와 상관없는 반복 계산 과정을 I loop 바깥으로 이동하면서 duu_is array 및 uu_temp에 저장한 후 이

를 load하여 사용함으로써 불필요한 계산 과정을 삭제하였다.

② 또한 is loop와 I loop의 순서를 바꿈으로 해서 cache miss rate를 줄였다.

6.6 deri_zt_total 루틴

A. do-loop 분리 및 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
326                          kl=k2-1
327            1             do 101 k=1,k2
328            1             do 101 j=1,j2
329           34             do 101 i=1,i2
331                          is=1
333         5884               duu(i,j,k)=a_inv_zt(k,is)*
334                         * (a_a12*(uu(i,j,2)-uu(i,j,1)) +a_a13*(uu(i,j,3)
335                         * -uu(i,j,1))+a_a14*(uu(i,j,4)-uu(i,j,1)))

347          336             do is=4,kl-2
348       157198               duu(i,j,k)=duu(i,j,k)+a_inv_zt(k,is)
349                         * *(a_c*(uu(i,j,is+3)-uu(i,j,is-3))
350                         *   +a_b*(uu(i,j,is+2)-uu(i,j,is-2))
351                         *   +a_a*(uu(i,j,is+1)-uu(i,j,is-1)))
352                          enddo

365                          is=k2
366          6709               duu(i,j,k)=duu(i,j,k)-a_inv_zt(k,is)*
367                         * (a_a12*(uu(i,j,kl)-uu(i,j,k2))+a_a13*(uu(i,j,kl-1)
368                         * -uu(i,j,k2))+a_a14*(uu(i,j,kl-2)-uu(i,j,k2)))
369           814       101    continue
```

<Serial 최적화 코드>

```
  7                          kl=k2-1
  9                          do k=1,k2
 10            5             do j=1,j2
 11                          do i=1,i2
 13                          is=1
 15          688               duu(i,j,k)=a_inv_zt(k,is)*
 16                         & (a_a12*(uu(i,j,2)-uu(i,j,1)) +a_a13*(uu(i,j,3)
 17                         & -uu(i,j,1))+a_a14*(uu(i,j,4)-uu(i,j,1)))
```

```
40                          is=k2
41          1544              duu(i,j,k)=duu(i,j,k)-a_inv_zt(k,is)*
42                           & (a_a12*(uu(i,j,kl)-uu(i,j,k2))+a_a13*(uu(i,j,kl-1)
43                           & -uu(i,j,k2))+a_a14*(uu(i,j,kl-2)-uu(i,j,k2)))
44
45                          enddo
46            6             enddo
47                          enddo
49                          do k=1,k2
50                          do is=4,kl-2
51           55              do j=1,j2
52                          do i=1,i2
53        37260               duu(i,j,k)=duu(i,j,k)+a_inv_zt(k,is)
54                           & *(a_c*(uu(i,j,is+3)-uu(i,j,is-3))
55                           &   +a_b*(uu(i,j,is+2)-uu(i,j,is-2))
56                           &   +a_a*(uu(i,j,is+1)-uu(i,j,is-1)))
57                          enddo
58          235             enddo
59            1             enddo
60                          enddo
```

① Original 코드에서 is loop가 수행될 때 발생되는 cache miss를 피하기 위해 전체 loop를 분리한 후 do-loop 순서를 수정하여 cache miss를 최대한 피하도록 하였다.

6.7 deri_et_total 루틴

A. do-loop 분리 및 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
278                         jl=j2-1
279                         do 101 k=1,k2
280                         do 101 j=1,j2
281           29             do 101 i=1,i2
282                         is=1
283         4415             duu(i,j,k)=a_inv_et(j,is)*
284                         * ( a_a12*(uu(i,2,k)-uu(i,1,k))+a_a13*(uu(i,3,k)
285                         * -uu(i,1,k))+a_a14*(uu(i,4,k)-uu(i,1,k)))

297           310           do is=4,jl-2
298        116188             duu(i,j,k)=duu(i,j,k)+a_inv_et(j,is)
299                         * *(a_c*(uu(i,is+3,k)-uu(i,is-3,k))
```

```
300                          *    +a_b*(uu(i,is+2,k)-uu(i,is-2,k))
301                          *    +a_a*(uu(i,is+1,k)-uu(i,is-1,k)))
302                           enddo

314                           is=j2
315          3715               duu(i,j,k)=duu(i,j,k)-a_inv_et(j,is)*
316                          * (a_a12*(uu(i,jl,k)-uu(i,j2,k))+a_a13*(uu(i,jl-1,k)
317                          * -uu(i,j2,k))+a_a14*(uu(i,jl-2,k)-uu(i,j2,k)))
318           795       101   continue
```

<Serial 최적화 코드>

```
 7                           jl=j2-1
 8                           do k=1,k2
10                           do i=1,i2
11             3             duu_is(1,i) =a_a12*(uu(i,2,k)-uu(i,1,k))
12                          &              +a_a13*(uu(i,3,k)-uu(i,1,k))
13                          &              +a_a14*(uu(i,4,k)-uu(i,1,k))

33            17             duu_is(6,i) =a_a12*(uu(i,jl,k)-uu(i,j2,k))
34                          &              +a_a13*(uu(i,jl-1,k)-uu(i,j2,k))
35                          &              +a_a14*(uu(i,jl-2,k)-uu(i,j2,k))
36                           enddo
38                           do j=1,j2
39                           do i=1,i2
40            555            duu(i,j,k)=a_inv_et(j,1)*duu_is(1,i)
41                          &              +a_inv_et(j,2)*duu_is(2,i)
42                          &              +a_inv_et(j,3)*duu_is(3,i)
43                          &              -a_inv_et(j,j2-2)*duu_is(4,i)
44                          &              -a_inv_et(j,j2-1)*duu_is(5,i)
45                          &              -a_inv_et(j,j2  )*duu_is(6,i)
46                           enddo
47                           enddo
49                           enddo
51                           do k=1,k2
52             6             do is=4,jl-2
54                           do i=1,i2
55            572            uu_temp(i) =a_c*(uu(i,is+3,k)-uu(i,is-3,k))
56                          &              +a_b*(uu(i,is+2,k)-uu(i,is-2,k))
57                          &              +a_a*(uu(i,is+1,k)-uu(i,is-1,k))
58                           enddo
59            94             do j=1,j2
60                           do i=1,i2
61          8908               duu(i,j,k)=duu(i,j,k)+a_inv_et(j,is)*uu_temp(i)
62                           enddo
63           109             enddo
65                           enddo
```

```
66                                    enddo
```

① Original 코드에서 j loop와 상관없는 반복 계산 과정을 j loop 바깥으로 이동하면서 duu_is array 및 uu_temp array에 저장한 후 이를 load하여 사용함으로써 불필요한 계산 과정을 삭제하였다.

② 또한 is loop와 I loop의 순서를 바꿈으로 해서 cache miss rate를 줄였다.

6.8 a_vsflux 루틴

A. 반복되는 계산 과정 삭제

<Original 코드>

```
811          74          a1 = 4./3.*six**2 +siy**2 +siz**2
812                      a2 = six**2 +4./3.*siy**2 +siz**2
813           1          a3 = six**2 +siy**2 +4./3.*siz**2
814                      a4 = six**2 +siy**2 +siz**2
815         227          a5 = six*siy/3.
816         126          a6 = siy*siz/3.
817           1          a7 = six*siz/3.

857          86          f1 = 4./3.*sjx*skx+sjy*sky+sjz*skz
858          27          f2 = sjx*skx+4./3.*sjy*sky+sjz*skz
859         215          f3 = sjx*skx+sjy*sky+4./3.*sjz*skz
860                      f4 = sjx*skx+sjy*sky+sjz*skz
861          33          f5 = sjx*sky-2./3.*sjy*skx
862          69          f6 = sjx*skz-2./3.*sjz*skx
863          99          f7 = sjy*skx-2./3.*sjx*sky
864         186          f8 = sjy*skz-2./3.*sjz*sky
865          97          f9 = sjz*skx-2./3.*sjx*skz
866                      f10= sjz*sky-2./3.*sjy*skz
```

<Serial 최적화 코드>

```
482                      a1 = abcdef(1 ,i,j,k)
483                      a2 = abcdef(2 ,i,j,k)
484                      a3 = abcdef(3 ,i,j,k)
485                      a4 = abcdef(4 ,i,j,k)
```

```
486          38              a5 = abcdef(5 ,i,j,k)
487           1              a6 = abcdef(6 ,i,j,k)
488                          a7 = abcdef(7 ,i,j,k)

528                          f1 = abcdef(42,i,j,k)
529                          f2 = abcdef(43,i,j,k)
530         153              f3 = abcdef(44,i,j,k)
531         164              f4 = abcdef(45,i,j,k)
532                          f5 = abcdef(46,i,j,k)
533                          f6 = abcdef(47,i,j,k)
534         159              f7 = abcdef(48,i,j,k)
535         153              f8 = abcdef(49,i,j,k)
536                          f9 = abcdef(50,i,j,k)
537          13              f10= abcdef(51,i,j,k)
```

① Original 코드에서 반복되는 계산 과정을 Serial 최적화 코드에서는 Jacobi_3d 루틴에서 미리 계산하여 저장해 둔 abcdef array를 바로 사용하여 반복계산 과정을 삭제하였다.

6.9 a_dflux 루틴

A. 불필요한 계산 과정 삭제

<Original 코드>

```
546                          do 101 k=1,k2
547                          do 101 j=1,j2
548                          do 101 i=1,i2
549          65              uu1(i,j,k)=a_e(1,i,j,k)
550         134              uu2(i,j,k)=a_e(2,i,j,k)
551         138              uu3(i,j,k)=a_e(3,i,j,k)
552          92              uu4(i,j,k)=a_e(4,i,j,k)
553         103              uu5(i,j,k)=a_e(5,i,j,k)
554           4       101    continue
555                          call deri_xi_total(uu1,duu1,i2,j2,k2)
556                          call deri_xi_total(uu2,duu2,i2,j2,k2)
557                          call deri_xi_total(uu3,duu3,i2,j2,k2)
558                          call deri_xi_total(uu4,duu4,i2,j2,k2)
559                          call deri_xi_total(uu5,duu5,i2,j2,k2)
560                          do 107 k=1,k2
561           1              do 107 j=1,j2
562                          do 107 i=1,i2
563          48              a_de_xi(1,i,j,k)= duu1(i,j,k)
564         114              a_de_xi(2,i,j,k)= duu2(i,j,k)
```

```
565          50              a_de_xi(3,i,j,k)= duu3(i,j,k)
566         141              a_de_xi(4,i,j,k)= duu4(i,j,k)
567          94              a_de_xi(5,i,j,k)= duu5(i,j,k)
568           4      107     continue
```

<Serial 최적화 코드>

```
 92                          call deri_xi_total_5(a_e,a_de_xi)
```

<deri_xi_total_5 루틴>

```
  7                         il=i2-1
  8                         do k=1,k2
  9                         do j=1,j2
 10                         do ir=1,5
 11          14             duu_is(1,ir)=(a_a12*(uu(ir,2,j,k)-uu(ir,1,j,k))
 12                        & +a_a13*(uu(ir,3,j,k)-uu(ir,1,j,k))
 13                        & +a_a14*(uu(ir,4,j,k)-uu(ir,1,j,k)))

 33          33             duu_is(6,ir)=(a_a12*(uu(ir,il,j,k)-uu(ir,i2,j,k))
 34                        & +a_a13*(uu(ir,il-1,j,k)
 35                        & -uu(ir,i2,j,k))+a_a14*(uu(ir,il-2,j,k)-uu(ir,i2,j,k)))
 36                         enddo
 37
 38                         do i=1,i2
 39                         do ir=1,5
 40         748             duu(ir,i,j,k)=a_inv_xi(i,1)*duu_is(1,ir)
 41                        &              +a_inv_xi(i,2)*duu_is(2,ir)
 42                        &              +a_inv_xi(i,3)*duu_is(3,ir)
 43                        &              -a_inv_xi(i,i2-2)*duu_is(4,ir)
 44                        &              -a_inv_xi(i,i2-1)*duu_is(5,ir)
 45                        &              -a_inv_xi(i,i2   )*duu_is(6,ir)
 46                         enddo
 47                         enddo
 48
 49                         do is=4,i2-3
 50                           do ir=1,5
 51          902                uu_temp(ir)=a_c*(uu(ir,is+3,j,k)-uu(ir,is-3,j,k))
 52                        &           +a_b*(uu(ir,is+2,j,k)-uu(ir,is-2,j,k))
 53                        &           +a_a*(uu(ir,is+1,j,k)-uu(ir,is-1,j,k))
 54                           enddo
 55                           do i=1,i2
 56                             do ir=1,5
 57       18541                   duu(ir,i,j,k)=duu(ir,i,j,k)+a_inv_xi(i,is)
                                   *uu_temp(ir)
```

```
58                                  enddo
59                                enddo
60              22              enddo
61
62               2              enddo
63                              enddo
```

① Original 코드에서 uu1 ~ uu5 array로 저장했다가 deri_xi_total 루틴 call 후 계산된 duu1~duu5에서 다시 a_de_xi array를 계산하는 부분을 Serial 최적화 코드에서는 이미 최적화된 deri_xi_total루틴을 이용하여 새로이 작성된 deri_xi_total_5 루틴을 추가하여 이러한 과정을 삭제하면서 main 코드에서 inline 시켰다.

② 즉 uu1 ~ uu5 array로 저장 및 다시 a_de_xi array를 계산하는 부분 대신 deri_xi_total_5 루틴안에서 ir loop가 돌게 하여 a_e 에서 a_de_xi가 바로 계산되도록 하였다.

B. 불필요한 계산 과정 삭제

<Original 코드>

```
569                             do 102 k=1,k2
570                             do 102 j=1,j2
571                             do 102 i=1,i2
572              45              uu1(i,j,k)=a_f(1,i,j,k)
573             102              uu2(i,j,k)=a_f(2,i,j,k)
574             134              uu3(i,j,k)=a_f(3,i,j,k)
575              70              uu4(i,j,k)=a_f(4,i,j,k)
576             104              uu5(i,j,k)=a_f(5,i,j,k)
577               1      102     continue
578                             call deri_et_total(uu1,duu1,i2,j2,k2)
579                             call deri_et_total(uu2,duu2,i2,j2,k2)
580                             call deri_et_total(uu3,duu3,i2,j2,k2)
581                             call deri_et_total(uu4,duu4,i2,j2,k2)
582                             call deri_et_total(uu5,duu5,i2,j2,k2)
583                             do 108 k=1,k2
584               1             do 108 j=1,j2
585                             do 108 i=1,i2
586              68              a_df_et(1,i,j,k)= duu1(i,j,k)
587             109              a_df_et(2,i,j,k)= duu2(i,j,k)
588              39              a_df_et(3,i,j,k)= duu3(i,j,k)
589             185              a_df_et(4,i,j,k)= duu4(i,j,k)
```

```
590          84              a_df_et(5,i,j,k)= duu5(i,j,k)
591           4      108     continue
```

<Serial 최적화 코드>

```
 93                          call deri_et_total_5(a_f,a_df_et)
```

<deri_et_total_5 루틴>

```
  7                         jl=j2-1
  8                         do k=1,k2
 10                         do i=1,i2
 11                         do ir=1,5
 12          25              duu_is(1,ir,i) =a_a12*(uu(ir,i,2,k)-uu(ir,i,1,k))
 13                        &               +a_a13*(uu(ir,i,3,k)-uu(ir,i,1,k))
 14                        &               +a_a14*(uu(ir,i,4,k)-uu(ir,i,1,k))

 34          17              duu_is(6,ir,i) =a_a12*(uu(ir,i,jl,k)-uu(ir,i,j2,k))
 35                        &               +a_a13*(uu(ir,i,jl-1,k)-uu(ir,i,j2,k))
 36                        &               +a_a14*(uu(ir,i,jl-2,k)-uu(ir,i,j2,k))
 37                         enddo
 38                         enddo
 40                         do j=1,j2
 41                         do i=1,i2
 42                         do ir=1,5
 43          638             duu(ir,i,j,k)=a_inv_et(j,1)*duu_is(1,ir,i)
 44                        &                +a_inv_et(j,2)*duu_is(2,ir,i)
 45                        &                +a_inv_et(j,3)*duu_is(3,ir,i)
 46                        &                -a_inv_et(j,j2-2)*duu_is(4,ir,i)
 47                        &                -a_inv_et(j,j2-1)*duu_is(5,ir,i)
 48                        &                -a_inv_et(j,j2  )*duu_is(6,ir,i)
 49                         enddo
 50                         enddo
 51                         enddo
 53                         enddo
 56                         do k=1,k2
 58           2             do is=4,jl-2
 59                         do i=1,i2
 60                         do ir=1,5
 61          752             uu_temp(ir,i) =a_c*(uu(ir,i,is+3,k)-uu(ir,i,is-3,k))
 62                        &               +a_b*(uu(ir,i,is+2,k)-uu(ir,i,is-2,k))
 63                        &               +a_a*(uu(ir,i,is+1,k)-uu(ir,i,is-1,k))
 64                         enddo
 65                         enddo
 66                         do j=1,j2
```

```
67                          do i=1,i2
68                          do ir=1,5
69          10374             duu(ir,i,j,k)=duu(ir,i,j,k)+a_inv_et(j,is)
                               *uu_temp(ir,i)
70                          enddo
71                          enddo
72             12           enddo
73                          enddo
75                          enddo
```

① Original 코드에서 uu1 ~ uu5 array로 저장했다가 deri_et_total 루틴 call 후 계산된 duu1~duu5에서 다시 a_df_et array를 계산하는 부분을 Serial 최적화 코드에서는 이미 최적화된 deri_et_total루틴을 이용하여 새로이 작성된 deri_et_total_5 루틴을 추가하여 이러한 과정을 삭제하면서 main 코드에서 inline 시켰다.

② 즉 uu1 ~ uu5 array로 저장 및 다시 a_df_xi array를 계산하는 부분 대신 deri_et_total_5 루틴안에서 ir loop가 돌게 하여 a_f 에서 a_df_et가 바로 계산되도록 하였다.

C. 불필요한 계산 과정 삭제

<Original 코드>

```
592                          do 103 k=1,k2
593             4            do 103 j=1,j2
594                          do 103 i=1,i2
595            42            uu1(i,j,k)=a_g(1,i,j,k)
596            97            uu2(i,j,k)=a_g(2,i,j,k)
597           153            uu3(i,j,k)=a_g(3,i,j,k)
598            62            uu4(i,j,k)=a_g(4,i,j,k)
599            69            uu5(i,j,k)=a_g(5,i,j,k)
600                    103    continue
601                          call deri_zt_total(uu1,duu1,i2,j2,k2)
602                          call deri_zt_total(uu2,duu2,i2,j2,k2)
603                          call deri_zt_total(uu3,duu3,i2,j2,k2)
604                          call deri_zt_total(uu4,duu4,i2,j2,k2)
605                          call deri_zt_total(uu5,duu5,i2,j2,k2)
606                          do 109 k=1,k2
607             1            do 109 j=1,j2
608                          do 109 i=1,i2
609            60            a_dg_zt(1,i,j,k)= duu1(i,j,k)
```

```
610          143                 a_dg_zt(2,i,j,k)= duu2(i,j,k)
611           40                 a_dg_zt(3,i,j,k)= duu3(i,j,k)
612          165                 a_dg_zt(4,i,j,k)= duu4(i,j,k)
613           76                 a_dg_zt(5,i,j,k)= duu5(i,j,k)
614                      109     continue
```

<Serial 최적화 코드>

```
 94                               call deri_zt_total_5(a_g,a_dg_zt)
```

<deri_zt_total_5 루틴>

```
  9                              kl=k2-1
 11                              do k=1,k2
 12                              do j=1,j2
 13           85                 do i=1,i2
 14                              do ir=1,5
 16                              is=1
 18         1671                   duu(ir,i,j,k)=a_inv_zt(k,is)*
 19                             & (a_a12*(uu(ir,i,j,2)-uu(ir,i,j,1))
 20                             & +a_a13*(uu(ir,i,j,3)-uu(ir,i,j,1))
 21                             & +a_a14*(uu(ir,i,j,4)-uu(ir,i,j,1)))

 52                              is=k2
 53         1052                   duu(ir,i,j,k)=duu(ir,i,j,k)-a_inv_zt(k,is)*
 54                             & (a_a12*(uu(ir,i,j,kl)-uu(ir,i,j,k2))
 55                             & +a_a13*(uu(ir,i,j,kl-1)-uu(ir,i,j,k2))
 56                             & +a_a14*(uu(ir,i,j,kl-2)-uu(ir,i,j,k2)))
 58                              enddo
 59           20                 enddo
 60            1                 enddo
 61                              enddo
 63                              do k=1,k2
 64                              do is=4,kl-2
 65                              do j=1,j2
 66                              do i=1,i2
 67                              do ir=1,5
 68        48521                   duu(ir,i,j,k)=duu(ir,i,j,k)+a_inv_zt(k,is)
 69                             & *(a_c*(uu(ir,i,j,is+3)-uu(ir,i,j,is-3))
 70                             &   +a_b*(uu(ir,i,j,is+2)-uu(ir,i,j,is-2))
 71                             &   +a_a*(uu(ir,i,j,is+1)-uu(ir,i,j,is-1)))
 72                              enddo
 73                              enddo
 74           45                 enddo
 75            1                 enddo
```

```
 76                                  enddo
```

① Original 코드에서 uu1 ~ uu5 array로 저장했다가 deri_zt_total 루틴 call 후 계산된 duu1~duu5에서 다시 a_dg_zt array를 계산하는 부분을 Serial 최적화 코드에서는 이미 최적화된 deri_zt_total루틴을 이용하여 새로이 작성된 deri_zt_total_5 루틴을 추가하여 이러한 과정을 삭제하면서 main 코드에서 inline 시켰다.

② 즉 uu1 ~ uu5 array로 저장 및 다시 a_dg_zt array를 계산하는 부분 대신 deri_zt_total_5 루틴안에서 ir loop가 돌게 하여 a_g에서 a_dg_zt가 바로 계산되도록 하였다.

6.10 a_dvsflux 루틴

a_dflux 루틴과 동일한 방법으로 최적화 하였다.

6.11 a_disspation 루틴

A. 반복계산 삭제 및 do-loop 내의 if문 삭제

<Original 코드>

```
 64                                  do 202 k=1,k2
 65                                  do 202 j=1,j2
 66              2                   do 202 i=1,i2
 67             58                   if(i.eq.1) then
 68             48                   pimax(j,k)=p(1,j,k)
 69                                  pimin(j,k)=p(1,j,k)
 70              1                   ramdaimax(j,k)=ramdai(1,j,k)
 71                                  ramdaimin(j,k)=ramdai(1,j,k)
 72              4                   ai=sqrt(xix(1,j,k)**2+xiy(1,j,k)**2+xiz(1,j,k)**2)
 73                                  ramdaiJamax(j,k)=ramdai(1,j,k)/ai
 74                                  ramdaiJamin(j,k)=ramdai(1,j,k)/ai
 75                                  endif

 97              8       202     continue
 98
100                                 do 201 k=1,k2
```

```
101                         do 201 j=1,j2
102          12             do 201 i=1,i2

105          154            if(p(i,j,k).gt.pimax(j,k)) pimax(j,k)=p(i,j,k)
106          27             if(p(i,j,k).lt.pimin(j,k)) pimin(j,k)=p(i,j,k)

112          594            if(ramdai(i,j,k).gt.ramdaimax(j,k)) ramdaimax(j,k)
                             =ramdai(i,j,k)
113                         if(ramdai(i,j,k).lt.ramdaimin(j,k)) ramdaimin(j,k)
                             =ramdai(i,j,k)
121          150            ai=sqrt(xix(i,j,k)**2+xiy(i,j,k)**2+xiz(i,j,k)**2)
122          10             if(ramdai(i,j,k)/ai.gt.ramdaiJamax(j,k))
123          3             *       ramdaiJamax(j,k)=ramdai(i,j,k)/ai
124          50             if(ramdai(i,j,k)/ai.lt.ramdaiJamin(j,k))
125          5             *       ramdaiJamin(j,k)=ramdai(i,j,k)/ai

139          39      201    continue
```

<Serial 최적화 코드>

```
 68                         do k=1,k2
 69                         do j=1,j2
 70                         do i=1,i2
 71          364               ramdaiJa(i,j,k)=ramdai(i,j,k)/ai_arr(i,j,k)
 72          22                ramdajJa(i,j,k)=ramdaj(i,j,k)/aj_arr(i,j,k)
 73          564               ramdakJa(i,j,k)=ramdak(i,j,k)/ak_arr(i,j,k)
 74                         enddo
 75          3              enddo
 76                         enddo
 77
 78                         do k=1,k2
 79                         do j=1,j2
 80                            p_max=p(1,j,k)
 81                            p_min=p(1,j,k)
 82                            r_max=ramdai(1,j,k)
 83                            r_min=ramdai(1,j,k)
 84                            rj_max=ramdaiJa(1,j,k)
 85                            rj_min=ramdaiJa(1,j,k)
 86                            do i=1,i2
 87          100                  if(p(i,j,k).gt.p_max) p_max=p(i,j,k)
 88                               if(p(i,j,k).lt.p_min) p_min=p(i,j,k)
 89          92                   if(ramdai(i,j,k).gt.r_max) r_max=ramdai(i,j,k)
 90          44                   if(ramdai(i,j,k).lt.r_min) r_min=ramdai(i,j,k)
 91          37                   if(ramdaiJa(i,j,k).gt.rj_max) rj_max
                                    =ramdaiJa(i,j,k)
 92          93                   if(ramdaiJa(i,j,k).lt.rj_min) rj_min
                                    =ramdaiJa(i,j,k)
```

```
 93                                  enddo
 94                                  pimax(j,k)=p_max
 95                                  pimin(j,k)=p_min
 96              1                   ramdaimax(j,k)=r_max
 97                                  ramdaimin(j,k)=r_min
 98              1                   ramdaiJamax(j,k)=rj_max
 99                                  ramdaiJamin(j,k)=rj_min
100                                enddo
101                                enddo
```

① Original 코드에서 do-loop 안의 if문을 do-loop문 분리를 통해 삭제하였다.

② Original 코드에서 I,j,k에 따라 constant 값을 가지는 ai, aj, ak 변수를 initialization 과정에서 ai_arr, aj_arr, ak_arr array로 저장한 후 바로 사용하였다.

B. do-loop 내의 if문 삭제

<Original 코드>

```
143                                do 151 k=1,kl
144                                do 151 j=1,jl
145              6                 do 151 i=1,il
147             16                 if(i.eq.1) then
148             12                 sigmai=pimax(j,k)/pimin(j,k)
149              1                 alphai=ramdaimax(j,k)/ramdaimin(j,k)
150             12                 betai=ramdaiJamax(j,k)/ramdaiJamin(j,k)
151              2                 ri=(alphai+betai)/(2*alphai*betai)
152                                if(alphai.eq.1.) then
153                                alphai_c=0
154                                else
155             13                 alphai_c=((alphai+1.)/(alphai-1.))*tanh(alphai-1.)
156                                endif
157                                if(betai.eq.1.) then
158                                betai_c=0
159                                else
160              6                 betai_c=((betai+1.)/(betai-1.))*tanh(betai-1.)
161                                endif
162             23                 akappa2i(j,k)=(1+(sigmai-1.)*tanh(alphai/betai-1.))
163                               *    *(sqrt(alphai_c*betai_c))**(1+tanh(sigmai-1.))
164                               *    /sigmai**ri
165                                  akappa4i(j,k)=akappa2i(j,k)
166                                endif
167
```

```
168             78              if(j.eq.1) then

188                             endif

189             13              if(k.eq.1) then

209                             endif
210             148     151     continue
```

<Serial 최적화 코드>

```
155                             do k=1,kl
156                             do j=1,jl
157                             sigmai=pimax(j,k)/pimin(j,k)
158             10              alphai=ramdaimax(j,k)/ramdaimin(j,k)
159             1               betai=ramdaiJamax(j,k)/ramdaiJamin(j,k)
160                             ri=(alphai+betai)/(2*alphai*betai)
161                             if(alphai.eq.1.) then
162                               alphai_c=0
163                             else
164             7                 alphai_c=((alphai+1.)/(alphai-1.))*tanh(alphai-1.)
165                             endif
166                             if(betai.eq.1.) then
167                               betai_c=0
168                             else
169             6                 betai_c=((betai+1.)/(betai-1.))*tanh(betai-1.)
170                             endif
171             30              akappa2i(j,k)=(1+(sigmai-1.)*tanh(alphai/betai-1.))
172                            &   *(sqrt(alphai_c*betai_c))**(1+tanh(sigmai-1.))
173                            &   /sigmai**ri
174                             akappa4i(j,k)=akappa2i(j,k)
175
176                             enddo
177                             enddo

179             1               do k=1,kl
180                             do i=1,il

199                             enddo
200                             enddo

202                             do j=1,jl
203                             do i=1,il

222                             enddo
223                             enddo
```

① Original 코드에서 do-loop 안의 if문을 do-loop문 분리를 통해 삭제하였다.

C. do-loop 내의 if문 삭제

<Original 코드>

```
212                       do 111 k=1,kl
213                       do 111 j=1,jl
214            28         do 111 i=1,il

268                          if(k.eq.1) then
269             1            anukmax=max(anuk(i,j,k),anuk(i,j,k+1)
270                       *          ,anuk(i,j,k+2),anuk(i,j,k+3))
271           137            elseif(k.eq.2) then
272             1            anukmax=max(anuk(i,j,k-1),anuk(i,j,k)
273                       *         ,anuk(i,j,k+1),anuk(i,j,k+2),anuk(i,j,k+3))
274
275            18            elseif (k.eq.kl) then
276             1            anukmax=max(anuk(i,j,k-2),anuk(i,j,k-1)
277                       *          ,anuk(i,j,k),anuk(i,j,k+1))
278                          elseif (k.eq.kl-1) then
279             1            anukmax=max(anuk(i,j,k-2),anuk(i,j,k-1)
280                       *          ,anuk(i,j,k),anuk(i,j,k+1),anuk(i,j,k+2))
281                          else
282           243            anukmax=max(anuk(i,j,k-2),anuk(i,j,k-1),
283                       *anuk(i,j,k),anuk(i,j,k+1),anuk(i,j,k+2),anuk(i,j,k+3))
284                          endif
285
286           218            epsil2k(i,j,k)=akappa2k(i,j)*anukmax     !!2.52
287            14            epsil_temp=akappa4k(i,j)-epsil2k(i,j,k)
288           212            epsil4k(i,j,k)=max(epsil_temp,0.)
289             1     111    continue
```

<Serial 최적화 코드>

```
322                       k=1
323                          do j=1,jl
324                          do i=1,il
325             3               anukmax=max(anuk(i,j,k),anuk(i,j,k+1)
326                       &                 ,anuk(i,j,k+2),anuk(i,j,k+3))
327             2               epsil2k(i,j,k)=akappa2k(i,j)*anukmax     !!2.52
328             3               epsil_temp=akappa4k(i,j)-epsil2k(i,j,k)
329             1               epsil4k(i,j,k)=max(epsil_temp,0.)
```

```
330                         enddo
331                         enddo
332
333                       k=2
334                         do j=1,jl
335                         do i=1,il
336           3               anukmax=max(anuk(i,j,k-1),anuk(i,j,k)
337                     &      ,anuk(i,j,k+1),anuk(i,j,k+2),anuk(i,j,k+3))
338                           epsil2k(i,j,k)=akappa2k(i,j)*anukmax     !!2.52
339                           epsil_temp=akappa4k(i,j)-epsil2k(i,j,k)
340           1               epsil4k(i,j,k)=max(epsil_temp,0.)
341                         enddo
342                         enddo
343
344                       k=kl-1
345                         do j=1,jl
346                         do i=1,il
347           7               anukmax=max(anuk(i,j,k-2),anuk(i,j,k-1)
348                     &          ,anuk(i,j,k),anuk(i,j,k+1),anuk(i,j,k+2))
349                           epsil2k(i,j,k)=akappa2k(i,j)*anukmax     !!2.52
350           1               epsil_temp=akappa4k(i,j)-epsil2k(i,j,k)
351                           epsil4k(i,j,k)=max(epsil_temp,0.)
352                         enddo
353                         enddo
354
355                       k=kl
356                         do j=1,jl
357                         do i=1,il
358           3               anukmax=max(anuk(i,j,k-2),anuk(i,j,k-1)
359                     &                ,anuk(i,j,k),anuk(i,j,k+1))
360           2               epsil2k(i,j,k)=akappa2k(i,j)*anukmax     !!2.52
361                           epsil_temp=akappa4k(i,j)-epsil2k(i,j,k)
362           3               epsil4k(i,j,k)=max(epsil_temp,0.)
363                         enddo
364                         enddo
365
366                       do k=3,kl-2
367           1             do j=1,jl
368                         do i=1,il
369         184               anukmax=max(anuk(i,j,k-2),anuk(i,j,k-1)
370                     &      ,anuk(i,j,k),anuk(i,j,k+1),anuk(i,j,k+2)
                               ,anuk(i,j,k+3))
371          59               epsil2k(i,j,k)=akappa2k(i,j)*anukmax     !!2.52
372           8               epsil_temp=akappa4k(i,j)-epsil2k(i,j,k)
373          13               epsil4k(i,j,k)=max(epsil_temp,0.)
374                         enddo
375                         enddo
376                       enddo
```

① Original 코드에서 do-loop 안의 if문을 do-loop문을 분리함으로써 삭제하였다.

D. do-loop 내의 if문 삭제

<Original 코드>

```
292                       do 117 k=1,kl
293                       do 117 j=1,jl
294           1           do 117 i=1,il
334                         if(j.eq.1) then
335           6             rmax_temp=max(ramdaj(i,j,k),ramdaj(i,j+1,k)
336                     *           ,ramdaj(i,j+2,k),ramdaj(i,j+3,k))
337           1             rmin_temp=min(ramdaj(i,j,k),ramdaj(i,j+1,k)
338                     *           ,ramdaj(i,j+2,k),ramdaj(i,j+3,k))
339                           ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
340         217             elseif(j.eq.2) then
341           5             rmax_temp=max(ramdaj(i,j-1,k),ramdaj(i,j,k)
342                     *    ,ramdaj(i,j+1,k),ramdaj(i,j+2,k),ramdaj(i,j+3,k))
343                           rmin_temp=min(ramdaj(i,j-1,k),ramdaj(i,j,k)
344                     *    ,ramdaj(i,j+1,k),ramdaj(i,j+2,k),ramdaj(i,j+3,k))
345           6             ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
346          24             elseif (j.eq.jl) then          !!check array
347                           rmax_temp=max(ramdaj(i,j-2,k)
348                     *    ,ramdaj(i,j-1,k),ramdaj(i,j,k),ramdaj(i,j+1,k))
349           1             rmin_temp=min(ramdaj(i,j-2,k)
350                     *    ,ramdaj(i,j-1,k),ramdaj(i,j,k),ramdaj(i,j+1,k))
351                           ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
352          18             elseif (j.eq.jl-1) then              !!check array
353           2             rmax_temp=max(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
354                     *    ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k))
355                           rmin_temp=min(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
356                     *    ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k))
357           4             ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
358                         else
359         253             rmax_temp=max(ramdaj(i,j-2,k),ramdaj(i,j-1,k),
360                     * ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k)
                            ,ramdaj(i,j+3,k))
361                           rmin_temp=min(ramdaj(i,j-2,k),ramdaj(i,j-1,k),
362                     * ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k)
                            ,ramdaj(i,j+3,k))
363         262             ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
364                         endif
400         277     117   continue
```

<Serial 최적화 코드>

```
422                          do k=1,kl
423                            j=1
424                            do i=1,il
425           2                  rmax_temp=max(ramdaj(i,j,k),ramdaj(i,j+1,k)
426                         &                ,ramdaj(i,j+2,k),ramdaj(i,j+3,k))
427           7                  rmin_temp=min(ramdaj(i,j,k),ramdaj(i,j+1,k)
428                         &                ,ramdaj(i,j+2,k),ramdaj(i,j+3,k))
429           1                  ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
430                            enddo
431
432                            j=2
433                            do i=1,il
434           3                  rmax_temp=max(ramdaj(i,j-1,k),ramdaj(i,j,k)
435                         &   ,ramdaj(i,j+1,k),ramdaj(i,j+2,k),ramdaj(i,j+3,k))
436                              rmin_temp=min(ramdaj(i,j-1,k),ramdaj(i,j,k)
437                         &   ,ramdaj(i,j+1,k),ramdaj(i,j+2,k),ramdaj(i,j+3,k))
438           1                  ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
439                            enddo
440
441                            j=jl-1
442                            do i=1,il
443           5                  rmax_temp=max(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
444                         &     ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k))
445           4                  rmin_temp=min(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
446                         &     ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k))
447           5                  ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
448                            enddo
449
450                            j=jl
451                            do i=1,il
452           3                  rmax_temp=max(ramdaj(i,j-2,k)
453                         &       ,ramdaj(i,j-1,k),ramdaj(i,j,k),ramdaj(i,j+1,k))
454                              rmin_temp=min(ramdaj(i,j-2,k)
455                         &       ,ramdaj(i,j-1,k),ramdaj(i,j,k),ramdaj(i,j+1,k))
456                              ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
457                            enddo
458
459                            do j=3,jl-2
460           1                do i=1,il
461         143                rmax_temp=max(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
462                         &  ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k)
                                ,ramdaj(i,j+3,k))
463          40                rmin_temp=min(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
                                ,ramdaj(i,j,k)
464                         &  ,ramdaj(i,j+1,k),ramdaj(i,j+2,k),ramdaj(i,j+3,k))
465          56                  ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
466                            enddo
```

```
467             1                 enddo
468
469                             enddo
```

① Original 코드에서 do-loop 안의 if문을 do-loop문을 분리함으로써 삭제하였다.

E. do-loop 순서 변경으로 Cache miss rate를 줄이고 불필요한 계산과정 및 array 저장 삭제

<Original 코드>

```
583           452               a_diss=0
584                             do 112 i=2,il
585                             do 112 j=2,jl
586                             do 112 k=2,kl
587                               do ir=1,5
588          4340               a_diss_xi(ir,i,j,k)=(disi1_2(ir,i,j,k)-disi1_2(ir,i-1,j,k))
589          6277               a_diss_et(ir,i,j,k)=(disj1_2(ir,i,j,k)-disj1_2(ir,i,j-1,k))
590          5972              a_diss_zt(ir,i,j,k)= (disk1_2(ir,i,j,k)-disk1_2(ir,i,j,k-1))
591           411               a_diss(ir,i,j,k)=a_diss_xi(ir,i,j,k)+a_diss_et(ir,i,j,k)
592                            *+a_diss_zt(ir,i,j,k)
596                               enddo
597            45       112     continue
```

<Serial 최적화 코드>

```
782                             if((mod(itr,100).eq.0) .or. itr.eq.IFINAL) then
783                             do 112 k=2,kl
784                             do 112 j=2,jl
785                             do 112 i=2,il
786                             do ir=1,5
787             9               a_diss_xi(ir,i,j,k)=(disi1_2(ir,i,j,k)-disi1_2(ir,i-1,j,k))
788             5               a_diss_et(ir,i,j,k)=(disj1_2(ir,i,j,k)-disj1_2(ir,i,j-1,k))
789            11              a_diss_zt(ir,i,j,k)= (disk1_2(ir,i,j,k)-disk1_2(ir,i,j,k-1))
790             2               a_diss(ir,i,j,k)=a_diss_xi(ir,i,j,k)+a_diss_et(ir,i,j,k)
791                            *+a_diss_zt(ir,i,j,k)
792                             enddo
793                     112     continue
794
795                             do j=1,j2
796                             do i=1,i2
```

```
797                       do ir=1,5
798                         a_diss(ir,i,j,1)=0.0
799                         a_diss(ir,i,j,k2)=0.0
800                       enddo
801                       enddo
802                       enddo
803
804                       do k=1,k2
805                       do i=1,i2
806                       do ir=1,5
807                         a_diss(ir,i,1,k)=0.0
808                         a_diss(ir,i,j2,k)=0.0
809                       enddo
810                       enddo
811                       enddo
812
813                       do k=1,k2
814                       do j=1,j2
815                       do ir=1,5
816                         a_diss(ir,1,j,k)=0.0
817                         a_diss(ir,i2,j,k)=0.0
818                       enddo
819                       enddo
820                       enddo
821
822                       else
823
824                       do k=2,kl
825             1         do j=2,jl
826                       do i=2,il
827                       do ir=1,5
828           742          a_diss(ir,i,j,k)=(disi1_2(ir,i,j,k)-disi1_2(ir,i-1,j,k))
829                      &                 +(disj1_2(ir,i,j,k)-disj1_2(ir,i,j-1,k))
830                      &                 +(disk1_2(ir,i,j,k)-disk1_2(ir,i,j,k-1))
831                       enddo
832                       enddo
833             2         enddo
834                       enddo
835
836                       do j=1,j2
837                       do i=1,i2
838                       do ir=1,5
839            26           a_diss(ir,i,j,1)=0.0
840                         a_diss(ir,i,j,k2)=0.0
841                       enddo
842                       enddo
843                       enddo
844
845                       do k=1,k2
```

```
846                         do i=1,i2
847                         do ir=1,5
848             15            a_diss(ir,i,1,k)=0.0
849                           a_diss(ir,i,j2,k)=0.0
850                         enddo
851                         enddo
852                         enddo
853
854                         do k=1,k2
855                         do j=1,j2
856                         do ir=1,5
857             23            a_diss(ir,1,j,k)=0.0
858                           a_diss(ir,i2,j,k)=0.0
859                         enddo
860                         enddo
861                         enddo
862
863                         endif
```

① Original 코드에서 비연속적인 메모리 access를 피하기 위해서 inner loop와 outer loop의 수행순서를 바꾸어 연속적인 메모리 access를 하도록 수정하였다.

② A_diss_xi, a_diss_et, a_diss_zt array를 항상 저장하지 않고, 필요한 iteration time에만 저장하도록 하였다.

③ A_diss array를 0으로 초기화하는 계산 과정을 필요한 영역만 0으로 초기화 하도록 하였다.

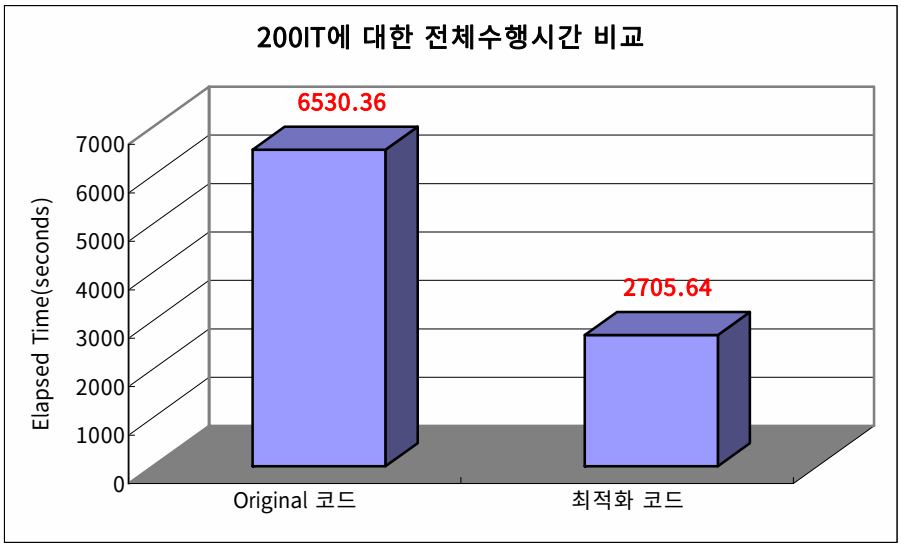
7. Serial 최적화 결과

코드 전체적으로 봤을 때 Original 코드의 do-loop 내에서 불필요하게 반복계산되는 부분을 수정하여 상당한 성능향상이 있었으며, 잘못 사용한 do-loop 순서를 수정하고, do-loop 안에 있는 if문을 분리하여 어느정도 계산시간을 단축하였다.

다음 결과는 iteration 횟수를 200으로 하여 수행한 결과이다.

< 표 IV.2 Original 코드와 최적화 코드의 전체 수행시간 비교 >

p690 1.7GHz 1CPU	전체 수행시간	Elapsed Time / Iteration
Original 코드	6530.36	32.65
최적화 코드	2705.64	13.53
Speedup	2.41	2.41



< 그림 IV.5 Original 코드와 최적화 코드의 코드 성능 비교 그래프 >

8. MPI 병렬화 코드 비교 및 분석

MPI 병렬화를 위해 다음 subroutine들을 수정하였다.

mpi_variables.inc head 파일

A. MPI 병렬 코드를 위한 기본적인 변수들을 선언

<MPI 병렬화 코드>

```
      parameter(max_nprocs=1024)
      parameter(ndims=2)
      parameter(myroot=0)

      integer nprocs,nprock,nprocj,myid,idw,ide,ids,idn,NEW_COMM
      integer idj,idk
      integer mysubnj(ndims,max_nprocs),mysubnk(ndims,max_nprocs)
      integer kstart,kend,jstart,jend,njsize,nksize

      integer MYSUB_COMM_J,MYSUB_COMM_K
      integer mysubjprocs,mysubkprocs,mysubjid,mysubkid

      character file_name*20

      common/mpi_variables/nprocs,nprock,nprocj,idj,idk,
     &     myid,idw,ide,ids,idn,mysubnj,mysubnk,
     &     kstart,kend,jstart,jend,njsize,nksize,
     &     NEW_COMM,
     &     MYSUB_COMM_J,MYSUB_COMM_K,
     &     mysubjprocs,mysubkprocs,mysubjid,mysubkid
```

① MPI 병렬화 코드에서 추가적으로 필요한 전역변수들을 head파일로 선언해 두었다.

a_datain 루틴

A. input data reading 과정

<Serial 최적화 코드>

```
 5                                OPEN(15,FILE='acoustics.d')
 6                                IREAD        = 15
 7                                WRITE (*,2)
 8                               2 FORMAT('     PROGRAM NS3D'/
 9                                .      '     FLOW ANALYSIS IN COMPRESSIBLE FLOW'/
10                                . '     BY SOLUTION OF STEADY 3D N-S EQUATIONS'/
11                                .         '     with k-e turbulence model')
12                                READ   (IREAD,530) TITLE
13                                WRITE (*,630) TITLE
14                                read(iread,500)
15                                read(iread,*) istart,ifinal
16                                write(*,*) istart,ifinal
17                                READ   (IREAD,500)
18                                READ   (IREAD,*) RM, res, iturb
19                                write(*,*) rm,res, iturb
20                                read(iread,500)
21                                read(iread,*) delt
22                                iopt_tur=0   !
23                                read(iread,500)
24                                read(iread,*) number_zone
25                                write(*,*) number_zone
```

<MPI 병렬화 코드>

```
 8                                real*8 buff(6)
 9                                integer*4 ibuff(2,6)
10                                equivalence(ibuff,buff)

14                                if(myid.eq.myroot) then
15                                OPEN(15,FILE='acoustics.d')
16                                IREAD        = 15
17                                WRITE (*,2)
18                               2 FORMAT('     PROGRAM NS3D'/
19                                .      '     FLOW ANALYSIS IN COMPRESSIBLE FLOW'/
20                                . '     BY SOLUTION OF STEADY 3D N-S EQUATIONS'/
21                                .         '     with k-e turbulence model')
22                                READ   (IREAD,530) TITLE
23                                WRITE (*,630) TITLE
24                                read(iread,500)
25                                read(iread,*) istart,ifinal
26                                write(*,*) istart,ifinal
27                                READ   (IREAD,500)
28                                READ   (IREAD,*) RM, res, iturb
29                                write(*,*) rm,res, iturb
30                                read(iread,500)
31                                read(iread,*) delt
32                                iopt_tur=0   !
```

```
 33                          read(iread,500)
 34                          read(iread,*) number_zone
 35                          write(*,*) number_zone
 36
 37                          buff(1)=rm
 38                          buff(2)=res
 39                          buff(3)=delt
 40                          ibuff(1,4)=istart
 41                          ibuff(2,4)=ifinal
 42                          ibuff(1,5)=iturb
 43                          ibuff(2,5)=number_zone
 44                          ibuff(1,6)=iopt_tur

272                          length=6
273                          CALL MPI_BCAST(buff,length,MPI_REAL8,myroot
                              ,MPI_COMM_WORLD,ierr)
274                          if(myid.ne.myroot) then
275                          rm=buff(1)
276                          res=buff(2)
277                          delt=buff(3)
278                          istart=ibuff(1,4)
279                          ifinal=ibuff(2,4)
280                          iturb=ibuff(1,5)
281                          number_zone=ibuff(2,5)
282                          iopt_tur=ibuff(1,6)
283                          endif
```

① Serial 코드와 달리 MPI 코드에서는 0번 rank에서 input data를 읽은 후 그 data를 전체 프로세스로 broadcasting하여 전달한다.

② 이때 input data가 real 변수도 있고, integer 변수도 있기 때문에 equivalence를 사용하여 하나의 변수로 통일한 후 한번의 MPI_BCAST 함수 call로 해당 변수 모두를 broadcasting할 수 있도록 하였다.

③ Broadcasting 후 buff array로부터 해당 변수 이름으로 저장한다.

acoustics 루틴

A. nprocj, nprock reading 과정

<MPI 병렬화 코드>

```
57                              CALL read_nprocj_nprock
58
59                              if(nprocs.ne.nprocj*nprock) then
60                                 if(myid.eq.myroot) then
61                                 write(*,*) 'Number of Processor(=',nprocs,
62                              &   ') should be equal to nprocj(=',nprocj,')
                                     *nprock(=',nprock,')'
63                                 write(*,*) 'Please doublecheck and Rerun'
64                                 endif
65                                CALL MPI_FINALIZE(ierr)
66                                STOP
67                              endif
```

<read_nprocj_nprock 루틴>

```
      if(myid.eq.myroot) then
      OPEN(10,FILE='nprocj_nprock.txt')
      READ(10,50)
      READ(10,50)
      READ(10,50)
      READ(10,50)
      READ(10,50)
      READ(10,50)
      READ(10,*) nprocj, nprock
      CLOSE(10)
50    FORMAT(200x)
      endif

      CALL MPI_BCAST(nprocj,1,MPI_INTEGER,myroot,MPI_COMM_WORLD,ierr)
      CALL MPI_BCAST(nprock,1,MPI_INTEGER,myroot,MPI_COMM_WORLD,ierr)
```

① nprocj, nprock 를 읽어들여 broadcasting하는 과정이다.

② 영역 분할 후 해당 rank가 너무 적은 영역을 차지하게 되면 과다한 통신이 발생하기 때문에 이를 피하기 위해 각 rank가 적당한 영역을 차지할 수 있도록 외부에서 nprocj, nprock를 지정하여 입력할 수 있도록 하였다.

B. Cartesian Topology의 Communicator 생성 및 Communicator 분리

```
 69                          DIMS(1)=nprocj
 70                          DIMS(2)=nprock
 71
 72                          PERIODIC(1) = .FALSE.
 73                          PERIODIC(2) = .FALSE.
 74                          REORDER = .TRUE.
 75
 76                          CALL MPI_CART_CREATE(MPI_COMM_WORLD,
 77                         & ndims,DIMS,PERIODIC,REORDER,NEW_COMM,ierr)
 78                          cALL MPI_COMM_RANK(NEW_COMM,myid,ierr)
 79
 80                          CALL MPI_CART_COORDS(NEW_COMM,myid,ndims,
                              mycoords,ierr)
 81
 82                          CALL MPI_CART_SHIFT(NEW_COMM,0,1,
                              idw,ide,ierr)
 83                          CALL MPI_CART_SHIFT(NEW_COMM,1,1,ids,idn,ierr)
 84
 85                          idj=mycoords(1)+1
 86                          idk=mycoords(2)+1
 87
 88                          CALL MPI_COMM_SPLIT(NEW_COMM,idk,idj,
                             MYSUB_COMM_J,ierr)
 89                          CALL MPI_COMM_SIZE(MYSUB_COMM_J,
                             mysubjprocs,ierr)
 90                          CALL MPI_COMM_RANK(MYSUB_COMM_J,
                             mysubjid,ierr)
 91
 92                          CALL MPI_COMM_SPLIT(NEW_COMM,idj,idk,
                              MYSUB_COMM_K,ierr)
 93                          CALL MPI_COMM_SIZE(MYSUB_COMM_K,
                             mysubkprocs,ierr)
 94                          CALL MPI_COMM_RANK(MYSUB_COMM_K,
                             mysubkid,ierr)
 95
 96                          CALL mypart(1,J2,nprocj,mysubnj)
 97                          CALL mypart(1,K2,nprock,mysubnk)
 98
 99                          jstart=mysubnj(1,idj)
100                          jend=mysubnj(2,idj)
101                          kstart=mysubnk(1,idk)
102                          kend=mysubnk(2,idk)
103
104                          njsize=jend-jstart+1
105                          nksize=kend-kstart+1
```

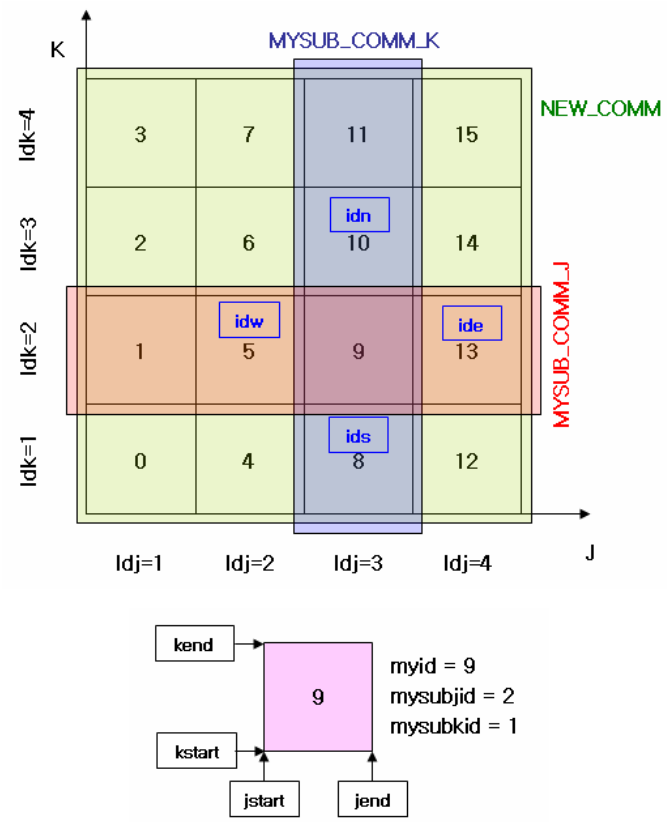
① NEW_COMM이라는 새로운 Communicator를 생성하고

NEW_COMM에서 새로운 Rank(myid)를 지정하였다.

② 이웃하는 rank와의 communication을 위해 mpi_cart_shift 루틴을 이용하여 idw, ide, ids, idn을 미리 지정해 두었다.

③ 효율적인 collective communication을 위해 mpi_comm_split 루틴을 이용하여 NEW_COMM내에서 MYSUB_COMM_J, MYSUB_COMM_K의 Communicator로 각각 분리하였다.

④ 영역분할 및 각 rank의 구성형태에 대한 전체적인 모습은 다음 그림과 같다.



< 그림 IV.6 영역 분할 및 rank 구성형태 >

C. Main iteration 구간

<Serial 최적화 코드>

```
 62                          DO K=1,K2
 63                          DO J=1,J2
 64                          DO I=1,I2
 65                          DO ir=1,5
 66            93              Wn(ir,I,J,K)=W(ir,I,J,K)
 67                          ENDDO
 68                          ENDDO
 69                          ENDDO
 70                          ENDDO

114                          do k=2,kl
115                          do j=2,jl
116                          do i=2,il
117                          do ir=1,5
118          1025             dw(ir,i,j,k)=-(a_de_xi(ir,i,j,k)+a_df_et(ir,i,j,k)
119                          &+a_dg_zt(ir,i,j,k))+a_vis_sorce(ir,i,j,k)
                              +a_diss(ir,i,j,k)*10
120                          enddo
121                          enddo
122                          enddo
123                          enddo

142                          RTRMS = 0.

144                          DO 90 K=2,KL
145                          DO 90 J=2,JL
146                          DO 90 I=2,IL
147                          do ir=1,5
148           219             RTRMS(ir)= RTRMS(ir)+DW(ir,I,J,K) *DW(ir,I,J,K)
149                          enddo
150                       90 CONTINUE

152                          do ir=1,5
153                            RTRMS(ir)=alog10(SQRT(RTRMS(ir))
                                *sqrt_ijk+10e-25)
154                          enddo
```

<MPI 병렬화 코드>

```
152                          DO K=kstart,kend
153                          DO J=jstart,jend
154                          DO I=1,I2
155                          DO ir=1,5
156             6              Wn(ir,I,J,K)=W(ir,I,J,K)
```

```
157                         ENDDO
158                         ENDDO
159                         ENDDO
160                         ENDDO

206                         DO K=max(kstart,2),min(kend,kl)
207                           DO J=max(jstart,2),min(jend,jl)
208                             do i=2,il
209                               do ir=1,5
210           60                    dw(ir,i,j,k)=-(a_de_xi(ir,i,j,k)
211                        &         +a_df_et(ir,i,j,k)+a_dg_zt(ir,i,j,k))
212                        &         +a_vis_sorce(ir,i,j,k)+a_diss(ir,i,j,k)*10
213                  !     &          + source_convec(ir,i,j,k)   !20070707
214                               enddo
215                             enddo
216                           enddo
217                         enddo

239                         RTRMS       = 0.
240                         RTRMS1      = 0.

242                         DO K=max(kstart,2),min(kend,KL)
243                           DO J=max(jstart,2),min(jend,JL)
244                             DO I=2,IL
245                               DO ir=1,5
246            4                  RTRMS1(ir)= RTRMS1(ir)+DW(ir,I,J,K)
                                  *DW(ir,I,J,K)
247                               ENDDO
248                             ENDDO
249                           ENDDO
250                         ENDDO

252                         CALL MPI_ALLREDUCE(RTRMS1,RTRMS,5,
253                        & MPI_REAL,MPI_SUM,NEW_COMM,ierr)

255                         do ir=1,5
256                           RTRMS(ir)=alog10(SQRT(RTRMS(ir))
                               *sqrt_ijk+10e-25)
257                         enddo
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

② RTRMS를 계산하는 과정에서는 각 rank의 해당 영역에서 계산을 한 후 MPI_ALLREDUCE를 이용하여 전 rank에서 다시 summation을 수행하였다.

D. 계산 결과를 write하는 부분

<Serial 최적화 코드>

```
187                         if((mod(itr,100).eq.0)) then
188                           if(iturb.eq.0) then
189                             CALL outputtest(iouttest,jouttest,kouttest)
190                           else if (iturb.eq.2) then
191                             CALL outputtest_turb(iouttest,jouttest,kouttest)
192                           endif
193                         endif

206                         OPEN( 8,FILE='flow.fom')
207                         DO 300 K=1,K2
208                         DO 300 J=1,J2
209                         DO 300 I=1,I2
210               3         WRITE(8,667) W(1,I,J,K),W(2,I,J,K),W(3,I,J,K)
211                        * ,W(4,I,J,K),W(5,I,J,K),wtk(i,j,k),wtom(i,j,k)
212                       300 CONTINUE
213                         close(8)
```

<outputtest 루틴>

```
  4                         tmp=1
  5                         open(17, file='flowxy_job_3d_6th.dat')
  6                         open(18, file='flowxz_job_3d_6th.dat')
  7                         open(19, file='flowzy_job_3d_6th.dat')
  8
  9                         write(17,11)
 10                         write(17,*) 'variables= x, y, u, v, w, p, r, tk, tom'
 11                         write(17,2144) i2,j2
 12                         k=kk
 13                         do 30 j=1,j2
 14                         do 30 i=1,i2
 15                         write(17,109) x(1,i,j,k),x(2,i,j,k),ur(i,j,k),vr(i,j,k)
 16                        *       ,wr(i,j,k),p(i,j,k),w(1,i,j,k),tk(i,j,k),tom(i,j,k)
 17                       30 continue
 18
 19                         write(18,12)
 20                         write(18,*) 'variables= x, z, u, v, w, p, r, tk, tom'
 21                         write(18,2143) i2,k2
 22                         tmp=1
 23
 24                         j=jj
 25                         do 31 k=1,k2
 26                         do 31 i=1,i2
```

```
 27                          write(18,109) x(1,i,j,k),x(3,i,j,k),ur(i,j,k),vr(i,j,k)
 28                        *       ,wr(i,j,k),p(i,j,k),w(1,i,j,k),tk(i,j,k),tom(i,j,k)
 29                       31 continue
 30
 31                          write(19,13)
 32                          write(19,*) 'variables= z, y, u, v, w, p, r, tk, tom'
 33                          write(19,2145) k2,j2
 34                          tmp=1
 35                          i=ii
 36                          do 32 j=1,j2
 37                          do 32 k=1,k2
 38                          write(19,109) x(3,i,j,k),x(2,i,j,k),ur(i,j,k),vr(i,j,k)
 39                        *       ,wr(i,j,k),p(i,j,k),w(1,i,j,k),tk(i,j,k),tom(i,j,k)
 40                       32 continue
 41
 42                          close(17)
 43                          close(18)
 44                          close(19)
```

<MPI 병렬화 코드>

```
299                          if((mod(itr,100).eq.0)) then
300                            if(iturb.eq.0) then
301                              CALL outputtest(iouttest,jouttest,kouttest)
302                            else if (iturb.eq.2) then
303                              CALL outputtest_turb(iouttest,jouttest,kouttest)
304                            endif
305                          endif

320                          CALL gatherv_4d_5(W)
321                          CALL gatherv_3d(wtk)
322                          CALL gatherv_3d(wtom)

324                          if(myid.eq.myroot) then
325                          OPEN( 8,FILE='flow.fom')
326                          DO 300 K=1,K2
327                          DO 300 J=1,J2
328                          DO 300 I=1,I2
329                          WRITE(8,667) W(1,I,J,K),W(2,I,J,K),W(3,I,J,K),
330                        *       W(4,I,J,K),W(5,I,J,K),wtk(i,j,k),wtom(i,j,k)
331                       300 CONTINUE
332                          close(8)
333                          endif
```

<outputtest 루틴>

```
 8                          DO J=1,nprocj
 9                            if(jj.ge.mysubnj(1,J).and.jj.le.mysubnj(2,J)) then
10                              idj_write=J
11                              goto 100
12                            endif
13                          ENDDO
14                  100     continue

17                          DO K=1,nprock
18                          if(kk.ge.mysubnk(1,K).and.kk.le.mysubnk(2,K)) then
19                               idk_write=K
20                               goto 200
21                            endif
22                          ENDDO
23                  200     continue

26                          if(nprock.gt.1) then
27                            CALL gatherv_kk(idk_write,kk,ur,vr,wr,p,w,tk,tom,
28                         &                          a_diss_xi,a_diss_et,a_diss_zt)
29                          endif
30                          if(nprocj.gt.1) then
31                            CALL gatherv_jj(idj_write,jj,ur,vr,wr,p,w,tk,tom,
32                         &                          a_diss_xi,a_diss_et,a_diss_zt)
33                          endif
34                          if(nprocs.gt.1) then
35                            CALL gatherv_ii(ii,ur,vr,wr,p,w,tk,tom,
36                         &                        a_diss_xi,a_diss_et,a_diss_zt)
37                          endif

39                          if(idk.eq.idk_write.and.idj.eq.nprocj) then
41                          open(17, file='flowxy_job_3d_6th.dat')
42                          write(17,11)
43                          write(17,*) 'variables= x, y, u, v, w, p, r, tk, tom'
44                          write(17,2144) i2,j2
45                          k=kk
46                          do 30 j=1,j2
47                          do 30 i=1,i2
48                          write(17,109) x(1,i,j,k),x(2,i,j,k),ur(i,j,k),vr(i,j,k)
49                         *       ,wr(i,j,k),p(i,j,k),w(1,i,j,k),tk(i,j,k),tom(i,j,k)
50                        30 continue
51                          close(17)
92                          endif

95                          if(idj.eq.idj_write.and.idk.eq.nprock) then
97                          open(18, file='flowxz_job_3d_6th.dat')
98                          write(18,12)
99                          write(18,*) 'variables= x, z, u, v, w, p, r, tk, tom'
100                         write(18,2143) i2,k2
```

```
101
102                         j=jj
103                         do 31 k=1,k2
104                         do 31 i=1,i2
105                         write(18,109) x(1,i,j,k),x(3,i,j,k),ur(i,j,k),vr(i,j,k)
106                        *       ,wr(i,j,k),p(i,j,k),w(1,i,j,k),tk(i,j,k),tom(i,j,k)
107                       31 continue
108                         close(18)
149                         endif

152                         if(myid.eq.myroot) then
154                         open(19, file='flowzy_job_3d_6th.dat')
155                         write(19,13)
156                         write(19,*) 'variables= z, y, u, v, w, p, r, tk, tom'
157                         write(19,2145) k2,j2
158                         i=ii
159                         do 32 j=1,j2
160                         do 32 k=1,k2
161                         write(19,109) x(3,i,j,k),x(2,i,j,k),ur(i,j,k),vr(i,j,k)
162                        *       ,wr(i,j,k),p(i,j,k),w(1,i,j,k),tk(i,j,k),tom(i,j,k)
163                       32 continue
164                         close(19)
205                         endif
```

<gathev_kk 루틴>

```
510                         if(idk.eq.myidk) then

513                         if(idj.eq.nprocj) then
514                           DO JJ=1,nprocj
515                             nj=mysubnj(2,JJ)-mysubnj(1,JJ)+1
516                             irank=JJ-1
517                             length=22*i2*nj
518                             ircnt(irank)=length
519                           ENDDO
520
521                           idisp(0)=0
522                           DO irank=1,nprocj-1
523                               idisp(irank)=idisp(irank-1)+ircnt(irank-1)
524                           ENDDO
525                         endif
526
527                         II=0
528                         DO J=jstart,jend
529                         DO I=1,i2
530                             II=II+1
531                             buffs(II)=ur1(I,J,kk)
```

```
532                                 II=II+1
533                                 buffs(II)=vr1(I,J,kk)
534                                 II=II+1
535                                 buffs(II)=wr1(I,J,kk)
536                                 II=II+1
537                                 buffs(II)=p1(I,J,kk)
538                                 II=II+1
539                                 buffs(II)=w11(1,I,J,kk)
540                                 II=II+1
541                                 buffs(II)=tk1(I,J,kk)
542                                 II=II+1
543                                 buffs(II)=tom1(I,J,kk)
544                                 DO ir=1,5
545                                   II=II+1
546                                   buffs(II)=ad_xi1(ir,I,J,kk)
547                                   II=II+1
548                                   buffs(II)=ad_et1(ir,I,J,kk)
549                                   II=II+1
550                                   buffs(II)=ad_zt1(ir,I,J,kk)
551                                 ENDDO
552                               ENDDO
553                               ENDDO
554
555                               iscnt=22*i2*njsize
557                               isrc=nprocj-1
558
559                               CALL MPI_GATHERV(buffs,iscnt,MPI_REAL,
560                              &                 buffr,ircnt,idisp,MPI_REAL,
561                              &                 isrc,MYSUB_COMM_J,ierr)

564                               if(idj.eq.nprocj) then
565                               II=0
566                               DO JJ=1,nprocj
567                               DO J=mysubnj(1,JJ),mysubnj(2,JJ)
568                               DO I=1,i2
569                                 II=II+1
570                                 ur1(I,J,kk)=buffr(II)
571                                 II=II+1
572                                 vr1(I,J,kk)=buffr(II)
573                                 II=II+1
574                                 wr1(I,J,kk)=buffr(II)
575                                 II=II+1
576                                 p1(I,J,kk)=buffr(II)
577                                 II=II+1
578                                 w11(1,I,J,kk)=buffr(II)
579                                 II=II+1
580                                 tk1(I,J,kk)=buffr(II)
581                                 II=II+1
582                                 tom1(I,J,kk)=buffr(II)
```

```
583                                 DO ir=1,5
584                                   II=II+1
585                                   ad_xi1(ir,I,J,kk)=buffr(II)
586                                   II=II+1
587                                   ad_et1(ir,I,J,kk)=buffr(II)
588                                   II=II+1
589                                   ad_zt1(ir,I,J,kk)=buffr(II)
590                                 ENDDO
591                               ENDDO
592                               ENDDO
593                               ENDDO
594                               endif
595
596                               endif
```

<gathev_jj 루틴>

```
618                               if(idj.eq.myidj) then

621                               if(idk.eq.nprock) then
622                                 DO KK=1,nprock
623                                   nk=mysubnk(2,KK)-mysubnk(1,KK)+1
624                                   irank=KK-1
625                                   length=22*i2*nk
626                                   ircnt(irank)=length
627                                 ENDDO
628
629                                 idisp(0)=0
630                                 DO irank=1,nprock-1
631                                     idisp(irank)=idisp(irank-1)+ircnt(irank-1)
632                                 ENDDO
633                               endif
634
635                               II=0
636                               DO K=kstart,kend
637                               DO I=1,i2
638                                  II=II+1
639                                  buffs(II)=ur1(I,jj,K)
640                                  II=II+1
641                                  buffs(II)=vr1(I,jj,K)
642                                  II=II+1
643                                  buffs(II)=wr1(I,jj,K)
644                                  II=II+1
645                                  buffs(II)=p1(I,jj,K)
646                                  II=II+1
647                                  buffs(II)=w11(1,I,jj,K)
648                                  II=II+1
```

```
649                                 buffs(II)=tk1(I,jj,K)
650                                 II=II+1
651                                 buffs(II)=tom1(I,jj,K)
652                                 DO ir=1,5
653                                   II=II+1
654                                   buffs(II)=ad_xi1(ir,I,jj,K)
655                                   II=II+1
656                                   buffs(II)=ad_et1(ir,I,jj,K)
657                                   II=II+1
658                                   buffs(II)=ad_zt1(ir,I,jj,K)
659                                 ENDDO
660                               ENDDO
661                               ENDDO
662
663                               iscnt=22*i2*nksize
665                               isrc=nprock-1
666
667                               CALL MPI_GATHERV(buffs,iscnt,MPI_REAL,
668                              &                 buffr,ircnt,idisp,MPI_REAL,
669                              &                 isrc,MYSUB_COMM_K,ierr)
672                               if(idk.eq.nprock) then
673                               II=0
674                               DO KK=1,nprock
675                               DO K=mysubnk(1,KK),mysubnk(2,KK)
676                               DO I=1,i2
677                                 II=II+1
678                                 ur1(I,jj,K)=buffr(II)
679                                 II=II+1
680                                 vr1(I,jj,K)=buffr(II)
681                                 II=II+1
682                                 wr1(I,jj,K)=buffr(II)
683                                 II=II+1
684                                 p1(I,jj,K)=buffr(II)
685                                 II=II+1
686                                 w11(1,I,jj,K)=buffr(II)
687                                 II=II+1
688                                 tk1(I,jj,K)=buffr(II)
689                                 II=II+1
690                                 tom1(I,jj,K)=buffr(II)
691                                 DO ir=1,5
692                                   II=II+1
693                                   ad_xi1(ir,I,jj,K)=buffr(II)
694                                   II=II+1
695                                   ad_et1(ir,I,jj,K)=buffr(II)
696                                   II=II+1
697                                   ad_zt1(ir,I,jj,K)=buffr(II)
698                                 ENDDO
699                               ENDDO
700                               ENDDO
```

```
701                             ENDDO
702                             endif
703
704                             endif
705
```

<gathev_ii 루틴>

```
726                             if(myid.eq.myroot) then
727                               DO JJ=1,nprocj
728                                 nj=mysubnj(2,JJ)-mysubnj(1,JJ)+1
729                                 DO KK=1,nprock
730                                   nk=mysubnk(2,KK)-mysubnk(1,KK)+1
731                                   irank=KK-1+nprock*(JJ-1)
732                                   length=22*nj*nk
733                                   ircnt(irank)=length
734                                 ENDDO
735                               ENDDO
736
737                               idisp(0)=0
738                               DO irank=1,nprocs-1
739                                   idisp(irank)=idisp(irank-1)+ircnt(irank-1)
740                               ENDDO
741                             endif
742
743                             II=0
744                             DO K=kstart,kend
745                             DO J=jstart,jend
746                                II=II+1
747                                buffs(II)=ur1(myii,J,K)
748                                II=II+1
749                                buffs(II)=vr1(myii,J,K)
750                                II=II+1
751                                buffs(II)=wr1(myii,J,K)
752                                II=II+1
753                                buffs(II)=p1(myii,J,K)
754                                II=II+1
755                                buffs(II)=w11(1,myii,J,K)
756                                II=II+1
757                                buffs(II)=tk1(myii,J,K)
758                                II=II+1
759                                buffs(II)=tom1(myii,J,K)
760                                DO ir=1,5
761                                  II=II+1
762                                  buffs(II)=ad_xi1(ir,myii,J,K)
763                                  II=II+1
764                                  buffs(II)=ad_et1(ir,myii,J,K)
765                                  II=II+1
```

```
766                                 buffs(II)=ad_zt1(ir,myii,J,K)
767                               ENDDO
768                             ENDDO
769                             ENDDO
770
771                             iscnt=22*njsize*nksize
772                             isrc=0
773
774                             CALL MPI_GATHERV(buffs,iscnt,MPI_REAL,
775                            &                 buffr,ircnt,idisp,MPI_REAL,
776                            &                 isrc,NEW_COMM,ierr)
777
778                             if(myid.eq.myroot) then
779                             II=0
780                             DO JJ=1,nprocj
781                             DO KK=1,nprock
782                             DO K=mysubnk(1,KK),mysubnk(2,KK)
783                             DO J=mysubnj(1,JJ),mysubnj(2,JJ)
784                               II=II+1
785                               ur1(myii,J,K)=buffr(II)
786                               II=II+1
787                               vr1(myii,J,K)=buffr(II)
788                               II=II+1
789                               wr1(myii,J,K)=buffr(II)
790                               II=II+1
791                               p1(myii,J,K)=buffr(II)
792                               II=II+1
793                               w11(1,myii,J,K)=buffr(II)
794                               II=II+1
795                               tk1(myii,J,K)=buffr(II)
796                               II=II+1
797                               tom1(myii,J,K)=buffr(II)
798                               DO ir=1,5
799                                 II=II+1
800                                 ad_xi1(ir,myii,J,K)=buffr(II)
801                                 II=II+1
802                                 ad_et1(ir,myii,J,K)=buffr(II)
803                                 II=II+1
804                                 ad_zt1(ir,myii,J,K)=buffr(II)
805                               ENDDO
806                             ENDDO
807                             ENDDO
808                             ENDDO
809                             ENDDO
810
811                             endif
```

<gathev_4d_5 루틴>

```
373                         if(myid.eq.myroot) then
374                           DO JJ=1,nprocj
375                             nj=mysubnj(2,JJ)-mysubnj(1,JJ)+1
376                             DO KK=1,nprock
377                               nk=mysubnk(2,KK)-mysubnk(1,KK)+1
378                               irank=KK-1+nprock*(JJ-1)
379                               length=5*i2*nj*nk
380                               ircnt(irank)=length
381                             ENDDO
382                           ENDDO
383
384                           idisp(0)=0
385                           DO irank=1,nprocs-1
386                               idisp(irank)=idisp(irank-1)+ircnt(irank-1)
387                           ENDDO
388                         endif
389
390                         II=0
391                         DO K=kstart,kend
392                         DO J=jstart,jend
393                         DO I=1,i2
394                         DO ir=1,5
395                             II=II+1
396                             buffs(II)=duu(ir,I,J,K)
397                         ENDDO
398                         ENDDO
399                         ENDDO
400                         ENDDO
401
402                         iscnt=5*i2*njsize*nksize
403
404                         CALL MPI_GATHERV(buffs,iscnt,MPI_REAL,
405                        &                 buffr,ircnt,idisp,MPI_REAL,
406                        &                 0,NEW_COMM,ierr)
407
408                         if(myid.eq.myroot) then
409                         II=0
410                         DO JJ=1,nprocj
411                         DO KK=1,nprock
412                         DO K=mysubnk(1,KK),mysubnk(2,KK)
413                         DO J=mysubnj(1,JJ),mysubnj(2,JJ)
414                         DO I=1,i2
415                         DO ir=1,5
416                             II=II+1
417                             duu(ir,I,J,K)=buffr(II)
418                         ENDDO
419                         ENDDO
```

```
420                         ENDDO
421                         ENDDO
422                         ENDDO
423                         ENDDO
424
425                         endif
```

<gathev_3d 루틴>

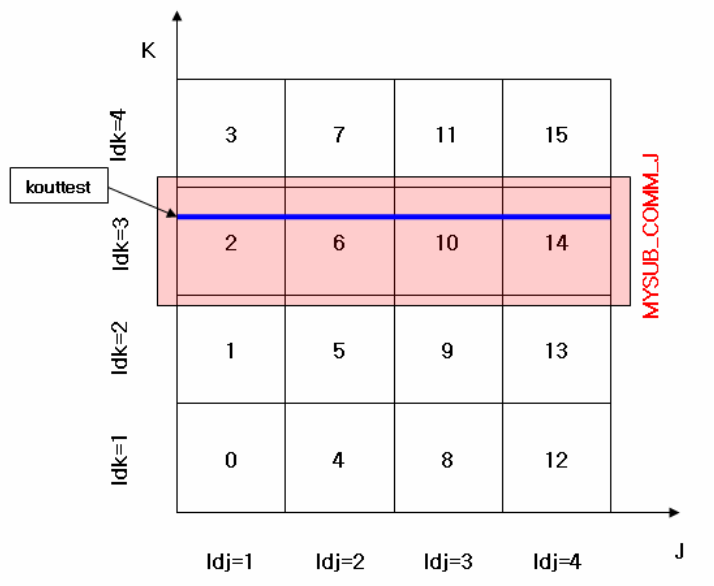
```
440                         if(myid.eq.myroot) then
441                           DO JJ=1,nprocj
442                             nj=mysubnj(2,JJ)-mysubnj(1,JJ)+1
443                             DO KK=1,nprock
444                               nk=mysubnk(2,KK)-mysubnk(1,KK)+1
445                               irank=KK-1+nprock*(JJ-1)
446                               length=i2*nj*nk
447                               ircnt(irank)=length
448                             ENDDO
449                           ENDDO
450
451                           idisp(0)=0
452                           DO irank=1,nprocs-1
453                               idisp(irank)=idisp(irank-1)+ircnt(irank-1)
454                           ENDDO
455                         endif
456
457                         II=0
458                         DO K=kstart,kend
459                         DO J=jstart,jend
460                         DO I=1,i2
461                             II=II+1
462                             buffs(II)=duu(I,J,K)
463                         ENDDO
464                         ENDDO
465                         ENDDO
466
467                         iscnt=i2*njsize*nksize
468
469                         CALL MPI_GATHERV(buffs,iscnt,MPI_REAL,
470                        &                 buffr,ircnt,idisp,MPI_REAL,
471                        &                 0,NEW_COMM,ierr)
472
473                         if(myid.eq.myroot) then
474                         II=0
475                         DO JJ=1,nprocj
476                         DO KK=1,nprock
477                         DO K=mysubnk(1,KK),mysubnk(2,KK)
```

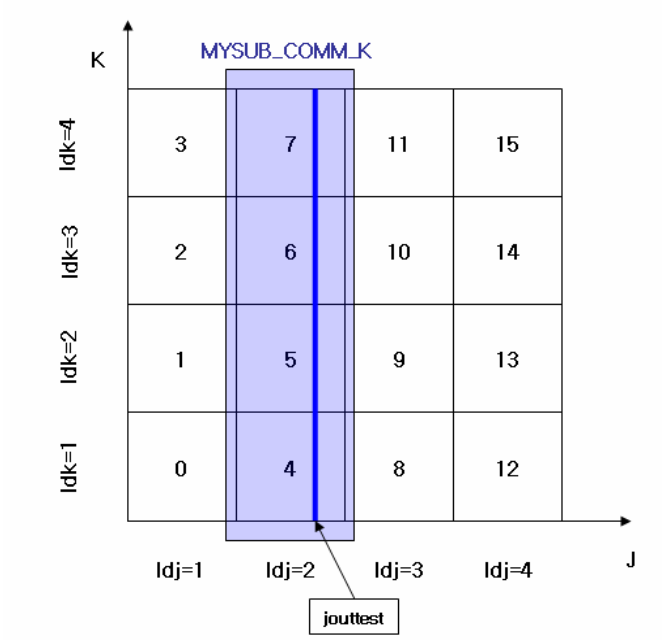
```
478                         DO J=mysubnj(1,JJ),mysubnj(2,JJ)
479                         DO I=1,i2
480                             II=II+1
481                             duu(I,J,K)=buffr(II)
482                         ENDDO
483                         ENDDO
484                         ENDDO
485                         ENDDO
486                         ENDDO
487
488                         endif
```

① 결과를 write할 때 MPI 코드에서는 해당 영역의 결과를 각 rank가 모두 가지고 있지 않기 때문에 임의의 rank로 write하려는 변수 값을 전달해 주어야 한다.

② 즉 iouttest에 대해서 전체 rank가 NEW_COMM communicator에서 mpi_gatherv를 통해 communication을 수행해야 하고, jouttest 및 kouttest 각각에 대해서는 해당 영역을 포함하고 있는 idj, idk를 찾아 그에 해당하는 communicator에서 mpi_gatherv를 통해 임의의 rank로 해당 변수의 결과를 전달해 주어야 한다.

③ 해당 rank가 iouttest, jouttest, koutest 영역에 대한 결과를 모두 전달받은 다음 그 rank만 write를 수행하도록 하는데, 이때 I/O 분산을 고려하여 write를 하는 rank가 최대한 겹치지 않도록 하였다.


K
ldk=4
ldk=3
ldk=2
ldk=1
kouttest
3
7
11
15
2
6
10
14
1
5
9
13
0
4
8
12
MYSUB_COMM_J
ldj=1
ldj=2
ldj=3
ldj=4
J


K
MYSUB_COMM_K
ldk=4
ldk=3
ldk=2
ldk=1
3
7
11
15
2
6
10
14
1
5
9
13
0
4
8
12
ldj=1
ldj=2
ldj=3
ldj=4
J
jouttest

deri_xi_total 루틴

A. do-loop 병렬화

<Serial 최적화 코드>

```
 7                          il=i2-1
 8                          do k=1,k2
 9                          do j=1,j2
10          12              duu_is(1)=(a_a12*(uu(2,j,k)-uu(1,j,k))+a_a13
11                         & *(uu(3,j,k)-uu(1,j,k))+a_a14*(uu(4,j,k)-uu(1,j,k)))

28                          do i=1,i2
29          597             duu(i,j,k)=a_inv_xi(i,1)*duu_is(1)
30                         & +a_inv_xi(i,2)*duu_is(2)+a_inv_xi(i,3)*duu_is(3)
                               -a_inv_xi(i,i2-2)*duu_is(4)
31                         & -a_inv_xi(i,i2-1)*duu_is(5)-a_inv_xi(i,i2)*duu_is(6)
32                          enddo
33
34            5             do is=4,i2-3
35          768             uu_temp=a_c*(uu(is+3,j,k)-uu(is-3,j,k))
36                         &   +a_b*(uu(is+2,j,k)-uu(is-2,j,k))
37                         &   +a_a*(uu(is+1,j,k)-uu(is-1,j,k))
38                          do i=1,i2
39        18699               duu(i,j,k)=duu(i,j,k)+a_inv_xi(i,is)*uu_temp
40                          enddo
41          146             enddo
42
43            4             enddo
44            1             enddo
```

<MPI 병렬화 코드>

```
 9                          il=i2-1
12                          DO K=kstart,kend
13                          DO J=jstart,jend
14
15            2             duu_is(1)=(a_a12*(uu(2,j,k)-uu(1,j,k))+a_a13
16                         & *(uu(3,j,k)-uu(1,j,k))+a_a14*(uu(4,j,k)-uu(1,j,k)))

33                          do i=1,i2
34           20             duu(i,j,k)=a_inv_xi(i,1)*duu_is(1)
35                         & +a_inv_xi(i,2)*duu_is(2)+a_inv_xi(i,3)*duu_is(3)
36                         &-a_inv_xi(i,i2-2)*duu_is(4)-a_inv_xi(i,i2-1)
                               *duu_is(5)-a_inv_xi(i,i2)*duu_is(6)
```

```
37                          enddo
38
39                          do is=4,i2-3
40              12          uu_temp=a_c*(uu(is+3,j,k)-uu(is-3,j,k))
41                         &   +a_b*(uu(is+2,j,k)-uu(is-2,j,k))
42                         &   +a_a*(uu(is+1,j,k)-uu(is-1,j,k))
43                          do i=1,i2
44              527           duu(i,j,k)=duu(i,j,k)+a_inv_xi(i,is)*uu_temp
45                          enddo
46                1         enddo
47
48                          enddo
49                          enddo
50
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

deri_et_total 루틴

A. do-loop 병렬화 및 j 방향 데이터 통신

<acoustics 루틴에서 deri_et_total 루틴 call하는 부분>

```
79                          call deri_et_total(ur,dur_et)
80                          call deri_et_total(vr,dvr_et)
81                          call deri_et_total(wr,dwr_et)
82                          call deri_et_total(t,dt_et)
```

<Serial 최적화 코드의 deri_et_total 루틴>

```
 7                          jl=j2-1
 8                          do k=1,k2
 9
10                          do i=1,i2
11               3          duu_is(1,i) =a_a12*(uu(i,2,k)-uu(i,1,k))
12                         &              +a_a13*(uu(i,3,k)-uu(i,1,k))
13                         &              +a_a14*(uu(i,4,k)-uu(i,1,k))

38                          do j=1,j2
39                          do i=1,i2
40              555         duu(i,j,k)=a_inv_et(j,1)*duu_is(1,i)
41                         &            +a_inv_et(j,2)*duu_is(2,i)
42                         &            +a_inv_et(j,3)*duu_is(3,i)
```

```
43                      &              -a_inv_et(j,j2-2)*duu_is(4,i)
44                      &              -a_inv_et(j,j2-1)*duu_is(5,i)
45                      &              -a_inv_et(j,j2  )*duu_is(6,i)
46                       enddo
47                       enddo
49                       enddo

51                       do k=1,k2
52            6          do is=4,jl-2
54                       do i=1,i2
55          572           uu_temp(i) =a_c*(uu(i,is+3,k)-uu(i,is-3,k))
56                      &               +a_b*(uu(i,is+2,k)-uu(i,is-2,k))
57                      &               +a_a*(uu(i,is+1,k)-uu(i,is-1,k))
58                       enddo
59           94          do j=1,j2
60                       do i=1,i2
61         8908            duu(i,j,k)=duu(i,j,k)+a_inv_et(j,is)*uu_temp(i)
62                       enddo
63          109          enddo
65                       enddo
66                       enddo
```

<MPI 병렬화 코드>

```
10                       jl=j2-1
11                       DO K=kstart,kend
12                       do i=1,i2
13                       duu_is(1,i) =a_a12*(uu(i,2,k)-uu(i,1,k))
14                      &               +a_a13*(uu(i,3,k)-uu(i,1,k))
15                      &               +a_a14*(uu(i,4,k)-uu(i,1,k))

39                       DO j=jstart,jend
41                       do i=1,i2
42            27         duu(i,j,k)=a_inv_et(j,1)*duu_is(1,i)
43                      &              +a_inv_et(j,2)*duu_is(2,i)
44                      &              +a_inv_et(j,3)*duu_is(3,i)
45                      &              -a_inv_et(j,jl-1)*duu_is(4,i)
46                      &              -a_inv_et(j,jl  )*duu_is(5,i)
47                      &              -a_inv_et(j,j2  )*duu_is(6,i)
48                       enddo
49                       enddo
51                       enddo

53                       DO K=kstart,kend
54                       do is=4,jl-2
55                          do i=1,i2
56            67             uu_temp(i) =a_c*(uu(i,is+3,k)-uu(i,is-3,k))
```

```
 57                    &                +a_b*(uu(i,is+2,k)-uu(i,is-2,k))
 58                    &                +a_a*(uu(i,is+1,k)-uu(i,is-1,k))
 59                       enddo
 61          4            DO j=jstart,jend
 62                       do i=1,i2
 63        246              duu(i,j,k)=duu(i,j,k)+a_inv_et(j,is)*uu_temp(i)
 64                       enddo
 65          2            enddo
 66                    enddo
 67                    enddo
```

<bcast_cross 루틴>

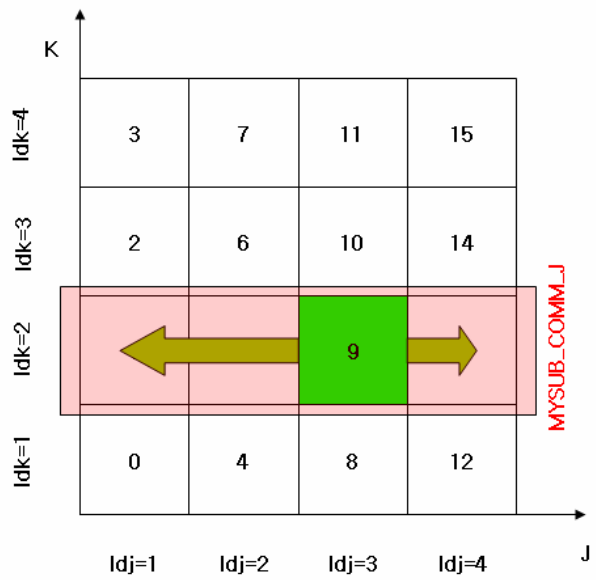
```
 91                    DO JJ=1,mysubjprocs
 93                       II=0
 94                       DO K=kstart,kend
 95                       DO J=mysubnj(1,JJ),mysubnj(2,JJ)
 96                       DO I=1,i2
 97          3               II=II+1
 98         17               buffj(II)=uu(i,J,k)
 99                          II=II+1
100         12               buffj(II)=vv(i,J,k)
101                          II=II+1
102          7               buffj(II)=ww(i,J,k)
103                          II=II+1
104         22               buffj(II)=tt(i,J,k)
105                       ENDDO
106                       ENDDO
107                       ENDDO
108
109                       isrc=JJ-1
110                       length=4*i2*(mysubnj(2,JJ)-mysubnj(1,JJ)+1)
                                *nksize
111                       CALL MPI_BCAST(buffj,length,MPI_REAL,
112                    &                    isrc,MYSUB_COMM_J,ierr)
113
114                       II=0
115                       DO K=kstart,kend
116                       DO J=mysubnj(1,JJ),mysubnj(2,JJ)
117                       DO I=1,i2
118                          II=II+1
119         11               uu(I,J,K)=buffj(II)
120          2               II=II+1
121          3               vv(I,J,K)=buffj(II)
122                          II=II+1
123                          ww(I,J,K)=buffj(II)
124          7               II=II+1
```

```
125             30                  tt(I,J,K)=buffj(II)
126                               ENDDO
127                               ENDDO
128                               ENDDO
130                             ENDDO
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

② deri_et_total 루틴을 수행하기 위해서는 해당 변수에 대해 j방향의 값을 모두 알고 있어야 하기 때문에 deri_et_total 루틴을 call하기 전에 bcast_cross 루틴을 call하여 해당 변수에 대해 j방향의 값을 모두 가지도록 하고 있다.

③ Bcast_cross루틴에서는 MYSUB_COMM_J communicator를 이용하여 동일한 idk사이에서만 broadcasting을 하도록 하면서 각각의 idk에 대해 동시에 broadcasting을 수행할 수 있도록 하였다.



deri_zt_total 루틴

A. do-loop 병렬화 및 k 방향 데이터 통신

<acoustics 루틴에서 deri_zt_total 루틴 call하는 부분>

```
 84                          call deri_zt_total(ur,dur_zt)
 85                          call deri_zt_total(vr,dvr_zt)
 86                          call deri_zt_total(wr,dwr_zt)
 87                          call deri_zt_total(t,dt_zt)
```

<Serial 최적화 코드의 deri_zt_total 루틴>

```
  7                          kl=k2-1
  8
  9                          do k=1,k2
 10              5           do j=1,j2
 11                          do i=1,i2
 13                          is=1
 15            688             duu(i,j,k)=a_inv_zt(k,is)*
 16                         & (a_a12*(uu(i,j,2)-uu(i,j,1)) +a_a13*(uu(i,j,3)
 17                         & -uu(i,j,1))+a_a14*(uu(i,j,4)-uu(i,j,1)))

 45                          enddo
 46              6           enddo
 47                          enddo

 49                          do k=1,k2
 50                          do is=4,kl-2
 51             55           do j=1,j2
 52                          do i=1,i2
 53          37260             duu(i,j,k)=duu(i,j,k)+a_inv_zt(k,is)
 54                         & *(a_c*(uu(i,j,is+3)-uu(i,j,is-3))
 55                         &   +a_b*(uu(i,j,is+2)-uu(i,j,is-2))
 56                         &   +a_a*(uu(i,j,is+1)-uu(i,j,is-1)))
 57                          enddo
 58            235           enddo
 59              1           enddo
 60                          enddo
```

<MPI 병렬화 코드>

```
  8                          kl=k2-1
```

```
 12                      DO K=kstart,kend
 13                      DO J=jstart,jend
 14                      do i=1,i2
 16                      is=1
 18           23           duu(i,j,k)=a_inv_zt(k,is)*
 19                     & (a_a12*(uu(i,j,2)-uu(i,j,1)) +a_a13*(uu(i,j,3)
 20                     & -uu(i,j,1))+a_a14*(uu(i,j,4)-uu(i,j,1)))

 48                      enddo
 49                      enddo
 50                      enddo

 53                      DO K=kstart,kend
 54                      do is=4,kl-2
 56            2         DO J=jstart,jend
 57                      do i=1,i2
 58         1018           duu(i,j,k)=duu(i,j,k)+a_inv_zt(k,is)
 59                     & *(a_c*(uu(i,j,is+3)-uu(i,j,is-3))
 60                     &   +a_b*(uu(i,j,is+2)-uu(i,j,is-2))
 61                     &   +a_a*(uu(i,j,is+1)-uu(i,j,is-1)))
 62                      enddo
 63           14         enddo
 64                      enddo
 65                      enddo
```

<bcass_cross 루틴>

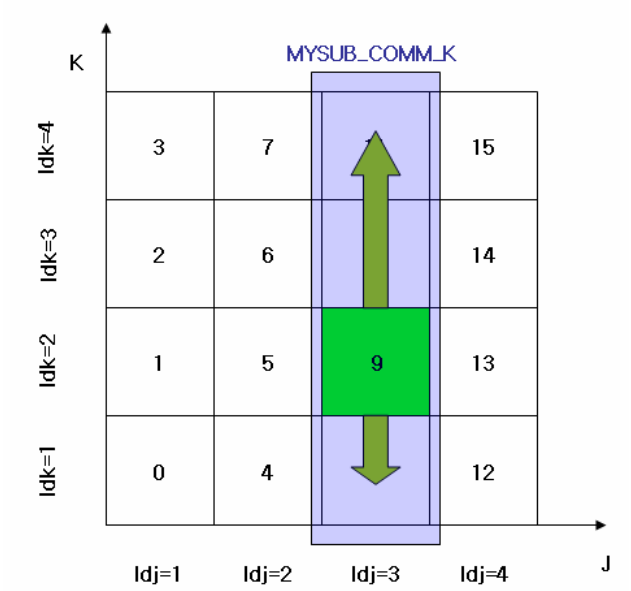
```
 82                      subroutine bcast_cross(uu,vv,ww,tt)

132                      DO KK=1,mysubkprocs
133
134                        II=0
135                        DO K=mysubnk(1,KK),mysubnk(2,KK)
136                        DO J=jstart,jend
137                        DO I=1,i2
138                          II=II+1
139           24             buffk(II)=uu(i,J,K)
140                          II=II+1
141           34             buffk(II)=vv(i,J,K)
142                          II=II+1
143           78             buffk(II)=ww(i,J,K)
144                          II=II+1
145           28             buffk(II)=tt(i,J,K)
146                        ENDDO
147            1           ENDDO
148            1           ENDDO
```

```
149
150                             isrc=KK-1
151                             length=4*i2*(mysubnk(2,KK)-mysubnk(1,KK)+1)
                                       *njsize
152                             CALL MPI_BCAST(buffk,length,MPI_REAL,
153                           &                      isrc,MYSUB_COMM_K,ierr)
154
155                             II=0
156                             DO K=mysubnk(1,KK),mysubnk(2,KK)
157                             DO J=jstart,jend
158                             DO I=1,i2
159                               II=II+1
160              9                uu(I,J,K)=buffk(II)
161              1                II=II+1
162             23                vv(I,J,K)=buffk(II)
163                               II=II+1
164             35                ww(I,J,K)=buffk(II)
165              1                II=II+1
166             11                tt(I,J,K)=buffk(II)
167                             ENDDO
168                             ENDDO
169                             ENDDO
170
171                           ENDDO
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

② deri_zt_total 루틴을 수행하기 위해서는 해당 변수에 대해 k방향의 값을 모두 알고 있어야 하기 때문에 deri_zt_total 루틴을 call하기 전에 bcast_cross 루틴을 call하여 해당 변수에 대해 k방향의 값을 모두 가지도록 하고 있다.

③ Bcast_cross루틴에서는 MYSUB_COMM_K communicator를 이용하여 동일한 idj사이에서만 broadcasting을 하도록 하면서 각각의 idj에 대해 동시에 broadcasting을 수행할 수 있도록 하였다.



a_flux 루틴

A. do-loop 병렬화

<Serial 최적화 코드>

```
381                       do 102 k=1,k2
382          2            do 102 j=1,j2
383                       do 102 i=1,i2
385        283             a_e(1,i,j,k)=vol(i,j,k)*w(1,i,j,k)*(xix(i,j,k)*ur(i,j,k)
386                      *    +xiy(i,j,k)*vr(i,j,k)+xiz(i,j,k)*wr(i,j,k))

449        254             a_g(5,i,j,k)=vol(i,j,k)*
450                      *    (ztx(i,j,k)*(w(5,i,j,k)+p(i,j,k))*ur(i,j,k)
451                      *    +zty(i,j,k)*(w(5,i,j,k)+p(i,j,k))*vr(i,j,k)
452                      *    +ztz(i,j,k)*(w(5,i,j,k)+p(i,j,k))*wr(i,j,k))
453          7     102    continue
```

<MPI 병렬화 코드>

```
  8                       DO K=kstart,kend
  9                       DO J=jstart,jend
```

```
 10                      do i=1,i2
 12          9           a_e(1,i,j,k)=vol(i,j,k)*w(1,i,j,k)*(xix(i,j,k)*ur(i,j,k)
 13                     *    +xiy(i,j,k)*vr(i,j,k)+xiz(i,j,k)*wr(i,j,k))

 76          7           a_g(5,i,j,k)=vol(i,j,k)*
 77                     *    (ztx(i,j,k)*(w(5,i,j,k)+p(i,j,k))*ur(i,j,k)
 78                     *    +zty(i,j,k)*(w(5,i,j,k)+p(i,j,k))*vr(i,j,k)
 79                     *    +ztz(i,j,k)*(w(5,i,j,k)+p(i,j,k))*wr(i,j,k))
 80                      enddo
 81                      enddo
 82                      enddo
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

a_vsflux 루틴

A. do-loop 병렬화

<Serial 최적화 코드>

```
465                     do 104 k=1,k2
466          12         do 104 j=1,j2
467           1          do 10 i=1,i2
468
469         145            vol2=vol(i,j,k)
470         472            visc = (t(i,j,k)/tfree)**vex/res
471
472         187            six=xix(i,j,k)
473                        siy=xiy(i,j,k)
474          52            siz=xiz(i,j,k)

635          66       10   continue
636          16      104 continue
```

<MPI 병렬화 코드>

```
 13                     DO k=kstart,kend
 14           1         DO j=jstart,jend
 15                     do i=1,i2
 16
 17                       vol2=vol(i,j,k)
 18          17           visc = (t(i,j,k)/tfree)**vex/res
```

```
 19
 20          4              six=xix(i,j,k)
 21                         siy=xiy(i,j,k)
 22          1              siz=xiz(i,j,k)

183          2            ENDDO
184                       ENDDO
185                       ENDDO
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

deri_xi_total_5(deri_xi_total_4) 루틴

A. do-loop 병렬화

<Serial 최적화 코드>

```
  7                        il=i2-1
  8                        do k=1,k2
  9                        do j=1,j2
 10                        do ir=1,5
 11          14            duu_is(1,ir)=(a_a12*(uu(ir,2,j,k)-uu(ir,1,j,k))
 12                       & +a_a13*(uu(ir,3,j,k)-uu(ir,1,j,k))
 13                       & +a_a14*(uu(ir,4,j,k)-uu(ir,1,j,k)))

 36                        enddo

 38                        do i=1,i2
 39                        do ir=1,5
 40          748           duu(ir,i,j,k)=a_inv_xi(i,1)*duu_is(1,ir)
 41                       &              +a_inv_xi(i,2)*duu_is(2,ir)
 42                       &              +a_inv_xi(i,3)*duu_is(3,ir)
 43                       &              -a_inv_xi(i,i2-2)*duu_is(4,ir)
 44                       &              -a_inv_xi(i,i2-1)*duu_is(5,ir)
 45                       &              -a_inv_xi(i,i2  )*duu_is(6,ir)
 46                        enddo
 47                        enddo
 48
 49                        do is=4,i2-3
 50                          do ir=1,5
 51          902               uu_temp(ir)=a_c*(uu(ir,is+3,j,k)-uu(ir,is-3,j,k))
 52                       &           +a_b*(uu(ir,is+2,j,k)-uu(ir,is-2,j,k))
 53                       &           +a_a*(uu(ir,is+1,j,k)-uu(ir,is-1,j,k))
 54                          enddo
```

```
 55                                do i=1,i2
 56                                   do ir=1,5
 57          18541                       duu(ir,i,j,k)=duu(ir,i,j,k)+a_inv_xi(i,is)
                                                  *uu_temp(ir)
 58                                   enddo
 59                                enddo
 60             22                enddo
 61
 62              2                enddo
 63                               enddo
 64
```

<MPI 병렬화 코드>

```
  9                               il=i2-1
 12                               do k=kstart,kend
 13                               do j=jstart,jend
 14                               do ir=1,5
 15                               duu_is(1,ir)=(a_a12*(uu(ir,2,j,k)-uu(ir,1,j,k))
 16                              & +a_a13*(uu(ir,3,j,k)-uu(ir,1,j,k))
 17                              & +a_a14*(uu(ir,4,j,k)-uu(ir,1,j,k)))

 40                               enddo

 42                               do i=1,i2
 43                               do ir=1,5
 44             21                duu(ir,i,j,k)=a_inv_xi(i,1)*duu_is(1,ir)
 45                              &                 +a_inv_xi(i,2)*duu_is(2,ir)
 46                              &                 +a_inv_xi(i,3)*duu_is(3,ir)
 47                              &                 -a_inv_xi(i,i2-2)*duu_is(4,ir)
 48                              &                 -a_inv_xi(i,i2-1)*duu_is(5,ir)
 49                              &                 -a_inv_xi(i,i2  )*duu_is(6,ir)
 50                               enddo
 51                               enddo
 52
 53                               do is=4,i2-3
 54                                  do ir=1,5
 55             23                      uu_temp(ir)=a_c*(uu(ir,is+3,j,k)-uu(ir,is-3,j,k))
 56                              &              +a_b*(uu(ir,is+2,j,k)-uu(ir,is-2,j,k))
 57                              &              +a_a*(uu(ir,is+1,j,k)-uu(ir,is-1,j,k))
 58                                  enddo
 59                                  do i=1,i2
 60                                     do ir=1,5
 61            537                      duu(ir,i,j,k)=duu(ir,i,j,k)+a_inv_xi(i,is)
                                                  *uu_temp(ir)
 62                                     enddo
 63                                  enddo
 64                               enddo
```

```
65
66                          enddo
67                          enddo
68
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

deri_et_total_5(deri_et_total_4) 루틴

A. do-loop 병렬화 및 j 방향 데이터 통신

<acoustics 루틴에서 deri_et_total_5, deri_et_total_4 루틴 call하는 부분>

```
185                         call deri_et_total_5(a_f,a_df_et)

189                         call deri_et_total_4(a_fv,a_dfv_et)
```

<Serial 최적화 코드의 deri_et_total 루틴>

```
 7                          jl=j2-1
 8                          do k=1,k2
 9
10                          do i=1,i2
11                          do ir=1,5
12              25          duu_is(1,ir,i) =a_a12*(uu(ir,i,2,k)-uu(ir,i,1,k))
13                         &               +a_a13*(uu(ir,i,3,k)-uu(ir,i,1,k))
14                         &               +a_a14*(uu(ir,i,4,k)-uu(ir,i,1,k))

37                          enddo
38                          enddo
39
40                          do j=1,j2
41                          do i=1,i2
42                          do ir=1,5
43              638         duu(ir,i,j,k)=a_inv_et(j,1)*duu_is(1,ir,i)
44                         &               +a_inv_et(j,2)*duu_is(2,ir,i)
45                         &               +a_inv_et(j,3)*duu_is(3,ir,i)
46                         &               -a_inv_et(j,j2-2)*duu_is(4,ir,i)
47                         &               -a_inv_et(j,j2-1)*duu_is(5,ir,i)
48                         &               -a_inv_et(j,j2  )*duu_is(6,ir,i)
49                          enddo
```

```
 50                          enddo
 51                          enddo

 53                          enddo

 56                          do k=1,k2
 57
 58            2             do is=4,jl-2
 59                          do i=1,i2
 60                          do ir=1,5
 61          752              uu_temp(ir,i) =a_c*(uu(ir,i,is+3,k)-uu(ir,i,is-3,k))
 62                         &               +a_b*(uu(ir,i,is+2,k)-uu(ir,i,is-2,k))
 63                         &               +a_a*(uu(ir,i,is+1,k)-uu(ir,i,is-1,k))
 64                          enddo
 65                          enddo
 66                          do j=1,j2
 67                          do i=1,i2
 68                          do ir=1,5
 69        10374               duu(ir,i,j,k)=duu(ir,i,j,k)+a_inv_et(j,is)
                                *uu_temp(ir,i)
 70                          enddo
 71                          enddo
 72            12            enddo
 73                          enddo
 74
 75                          enddo
 76
```

<MPI 병렬화 코드>

```
  9                         jl=j2-1
 11            1             do k=kstart,kend
 12
 13                          do i=1,i2
 14                          do ir=1,5
 15            4             duu_is(1,ir,i) =a_a12*(uu(ir,i,2,k)-uu(ir,i,1,k))
 16                         &               +a_a13*(uu(ir,i,3,k)-uu(ir,i,1,k))
 17                         &               +a_a14*(uu(ir,i,4,k)-uu(ir,i,1,k))

 40                          enddo
 41                          enddo

 44                          do j=jstart,jend
 45                          do i=1,i2
 46                          do ir=1,5
 47           21             duu(ir,i,j,k)=a_inv_et(j,1)*duu_is(1,ir,i)
 48                         &                +a_inv_et(j,2)*duu_is(2,ir,i)
 49                         &                +a_inv_et(j,3)*duu_is(3,ir,i)
```

```
 50                      &                -a_inv_et(j,j2-2)*duu_is(4,ir,i)
 51                      &                -a_inv_et(j,j2-1)*duu_is(5,ir,i)
 52                      &                -a_inv_et(j,j2  )*duu_is(6,ir,i)
 53                       enddo
 54                       enddo
 55                       enddo
 56
 57                       enddo

 61                       do k=kstart,kend
 62
 63                       do is=4,jl-2
 64                       do i=1,i2
 65                       do ir=1,5
 66            81          uu_temp(ir,i) =a_c*(uu(ir,i,is+3,k)-uu(ir,i,is-3,k))
 67                      &              +a_b*(uu(ir,i,is+2,k)-uu(ir,i,is-2,k))
 68                      &              +a_a*(uu(ir,i,is+1,k)-uu(ir,i,is-1,k))
 69                       enddo
 70                       enddo

 72             1         do j=jstart,jend
 73                       do i=1,i2
 74                       do ir=1,5
 75           291           duu(ir,i,j,k)=duu(ir,i,j,k)+a_inv_et(j,is)
                             *uu_temp(ir,i)
 76                       enddo
 77                       enddo
 78                       enddo
 79                       enddo
 80
 81                       enddo
```

<jk45_bcast 루틴>

```
177                      subroutine jk45_bcast(ff1,gg1,fv1,gv1)

186                      DO JJ=1,nprocj
187
188                        II=0
189                        DO K=kstart,kend
190                        DO J=mysubnj(1,JJ),mysubnj(2,JJ)
191                        DO I=1,i2
192                          DO ir=1,4
193                            II=II+1
194            83              buff(II)=ff1(ir,i,J,k)
195                            II=II+1
196            58              buff(II)=fv1(ir,i,J,k)
```

```
197                                 ENDDO
198             1                   II=II+1
199             2                   buff(II)=ff1(5,i,J,k)
200                               ENDDO
201             1                 ENDDO
202                               ENDDO
203
204                               isrc=JJ-1
205             1                 length=9*i2*(mysubnj(2,JJ)-mysubnj(1,JJ)+1)
                                  *nksize
206                               CALL MPI_BCAST(buff,length,MPI_REAL,
207                            &                      isrc,MYSUB_COMM_J,ierr)
208
209                               II=0
210                               DO K=kstart,kend
211                               DO J=mysubnj(1,JJ),mysubnj(2,JJ)
212                               DO I=1,i2
213                                 DO ir=1,4
214             9                     II=II+1
215            56                     ff1(ir,I,J,K)=buff(II)
216             1                     II=II+1
217            45                     fv1(ir,I,J,K)=buff(II)
218                                 ENDDO
219             5                   II=II+1
220            11                   ff1(5,I,J,K)=buff(II)
221                               ENDDO
222                               ENDDO
223                               ENDDO
224
225                             ENDDO
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

② deri_et_total 루틴과 마찬가지로 deri_et_total_5루틴을 수행하기 위해서는 해당 변수에 대해 j방향의 값을 모두 알고 있어야 하기 때문에 deri_et_total_5 루틴을 call하기 전에 jk45_bcast 루틴을 call하여 해당 변수에 대해 j방향의 값을 모두 가지도록 하고 있다.

③ Jk45_bcast루틴에서는 MYSUB_COMM_J communicator를 이용하여 동일한 idk사이에서만 broadcasting을 하도록 하면서 각각의 idk에 대해 동시에 broadcasting을 수행할 수 있도록 하였다.

④ deri_et_total_4루틴은 ir loop의 크기가 4인 것 외에는

deri_et_total_5루틴과 동일한 형태를 갖는다.

deri_zt_total_5, deri_zt_total_4 루틴

A. do-loop 병렬화 및 k 방향 데이터 통신

<acoustics 루틴에서 deri_zt_total_5, deri_zt_total_4 루틴 call하는 부분>

```
186                         call deri_zt_total_5(a_g,a_dg_zt)

190                         call deri_zt_total_4(a_gv,a_dgv_zt)
```

<Serial 최적화 코드의 deri_zt_total 루틴>

```
  9                         kl=k2-1
 10
 11                         do k=1,k2
 12                         do j=1,j2
 13              85          do i=1,i2
 14                         do ir=1,5
 15
 16                         is=1
 17
 18            1671           duu(ir,i,j,k)=a_inv_zt(k,is)*
 19                        & (a_a12*(uu(ir,i,j,2)-uu(ir,i,j,1))
 20                        & +a_a13*(uu(ir,i,j,3)-uu(ir,i,j,1))
 21                        & +a_a14*(uu(ir,i,j,4)-uu(ir,i,j,1)))

 58                         enddo
 59              20         enddo
 60               1         enddo
 61                         enddo
 62
 63                         do k=1,k2
 64                         do is=4,kl-2
 65                         do j=1,j2
 66                         do i=1,i2
 67                         do ir=1,5
 68           48521            duu(ir,i,j,k)=duu(ir,i,j,k)+a_inv_zt(k,is)
 69                        & *(a_c*(uu(ir,i,j,is+3)-uu(ir,i,j,is-3))
 70                        &   +a_b*(uu(ir,i,j,is+2)-uu(ir,i,j,is-2))
 71                        &   +a_a*(uu(ir,i,j,is+1)-uu(ir,i,j,is-1)))
```

```
 72                          enddo
 73                          enddo
 74          45              enddo
 75           1              enddo
 76                          enddo
 77
```

<MPI 병렬화 코드>

```
  8                          kl=k2-1

 12                          DO K=kstart,kend
 13                          DO J=jstart,jend
 14           6              do i=1,i2
 15                          do ir=1,5
 16
 17                          is=1
 18
 19          29               duu(ir,i,j,k)=a_inv_zt(k,is)*
 20                         & (a_a12*(uu(ir,i,j,2)-uu(ir,i,j,1))
 21                         & +a_a13*(uu(ir,i,j,3)-uu(ir,i,j,1))
 22                         & +a_a14*(uu(ir,i,j,4)-uu(ir,i,j,1)))

 59                          enddo
 60           2              enddo
 61                          enddo
 62                          enddo

 65                          DO K=kstart,kend
 66                          do is=4,kl-2
 68                          DO J=jstart,jend
 69                          do i=1,i2
 70                          do ir=1,5
 71        1334                duu(ir,i,j,k)=duu(ir,i,j,k)+a_inv_zt(k,is)
 72                         & *(a_c*(uu(ir,i,j,is+3)-uu(ir,i,j,is-3))
 73                         &   +a_b*(uu(ir,i,j,is+2)-uu(ir,i,j,is-2))
 74                         &   +a_a*(uu(ir,i,j,is+1)-uu(ir,i,j,is-1)))
 75                          enddo
 76                          enddo
 77                          enddo
 78                          enddo
 79                          enddo
```

<jk45_bcast 루틴>

```
177                         subroutine jk45_bcast(ff1,gg1,fv1,gv1)
```

```
227                       DO KK=1,nprock
228
229                         II=0
230           1             DO K=mysubnk(1,KK),mysubnk(2,KK)
231                         DO J=jstart,jend
232                         DO I=1,i2
233                           DO ir=1,4
234                             II=II+1
235         245               buff(II)=gg1(ir,i,J,K)
236                             II=II+1
237         228               buff(II)=gv1(ir,i,J,K)
238                           ENDDO
239           3               II=II+1
240          15               buff(II)=gg1(5,i,J,K)
241                         ENDDO
242                         ENDDO
243           1             ENDDO
244
245                         isrc=KK-1
246                         length=9*i2*(mysubnk(2,KK)-mysubnk(1,KK)+1)
                                  *njsize
247                         CALL MPI_BCAST(buff,length,MPI_REAL,
248                        &                    isrc,MYSUB_COMM_K,ierr)
249
250                         II=0
251                         DO K=mysubnk(1,KK),mysubnk(2,KK)
252           2             DO J=jstart,jend
253                         DO I=1,i2
254                           DO ir=1,4
255          20               II=II+1
256         119               gg1(ir,I,J,K)=buff(II)
257          12               II=II+1
258         317               gv1(ir,I,J,K)=buff(II)
259                           ENDDO
260           8               II=II+1
261          45               gg1(5,I,J,K)=buff(II)
262                         ENDDO
263                         ENDDO
264                         ENDDO
265
266                       ENDDO
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

② deri_zt_total 루틴과 마찬가지로 deri_zt_total_5루틴을 수행하기 위해서는 해당 변수에 대해 k방향의 값을 모두 알고 있어야 하

기 때문에 deri_zt_total_5 루틴을 call하기 전에 jk45_bcast 루틴을 call하여 해당 변수에 대해 k방향의 값을 모두 가지도록 하고 있다.

③ Jk45_bcast루틴에서는 MYSUB_COMM_K communicator를 이용하여 동일한 idj사이에서만 broadcasting을 하도록 하면서 각각의 idj에 대해 동시에 broadcasting을 수행할 수 있도록 하였다.

④ deri_zt_total_4루틴은 ir loop의 크기가 4인 것 외에는 deri_zt_total_5루틴과 동일한 형태를 갖는다.

a_disspation 루틴

A. do-loop 병렬화 및 이웃하는 rank와의 데이터 통신

<Serial 최적화 코드>

```
13                      do 110 k=1,k2
14          1           do 110 j=1,j2
15          1           do 110 i=1,i2

25        598            cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))
26        260            ramdai(i,j,k)=abs(ur(i,j,k)*six+vr(i,j,k)*siy
27                      *    +wr(i,j,k)*siz)+cc*sqrt(six**2+siy**2+siz**2)
28        172            ramdaj(i,j,k)=abs(ur(i,j,k)*sjx+vr(i,j,k)*sjy+wr(i,j,k)
29                      *    *sjz)+cc*sqrt(sjx**2+sjy**2+sjz**2)
30        173            ramdak(i,j,k)=abs(ur(i,j,k)*skx+vr(i,j,k)*sky
31                      *    +wr(i,j,k)*skz)+cc*sqrt(skx**2+sky**2+skz**2)

41        836            anui(i,j,k)=abs(p(ia-1,j,k)-2*p(ia,j,k)+p(ia+1,j,k))
42                      * /(p(ia-1,j,k)+2*p(ia,j,k)+p(ia+1,j,k))
43
44                      if(j.eq.1) then
45          9              ja=2
46                      elseif(j.eq.j2) then
47          6              ja=jl
48                      else
49        243              ja=j
50                      endif
51
52        535            anuj(i,j,k)=abs(p(i,ja-1,k)-2*p(i,ja,k)+p(i,ja+1,k))
53                      * /(p(i,ja-1,k)+2*p(i,ja,k)+p(i,ja+1,k))
54
```

```
 55                                  if(k.eq.1) then
 56                                    ka=2
 57              23                  elseif(k.eq.k2) then
 58                                    ka=kl
 59                                  else
 60              21                    ka=k
 61                                  endif
 62
 63             250                  anuk(i,j,k)=abs(p(i,j,ka-1)-2*p(i,j,ka)+p(i,j,ka+1))
 64                                 * /(p(i,j,ka-1)+2*p(i,j,ka)+p(i,j,ka+1))
 65
 66               6         110     continue
```

<MPI 병렬화 코드>

```
 41                                  do 110 k=max(kstart-3,1),min(kend+3,k2)
 42                                  do 110 j=max(jstart-3,1),min(jend+3,j2)
 43                                  do 110 i=1,i2

 53              16                  cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))
 54              15                  ramdai(i,j,k)=abs(ur(i,j,k)*six+vr(i,j,k)*siy+
 55                                 * wr(i,j,k)*siz)+cc*sqrt(six**2+siy**2+siz**2)
 56              44                  ramdaj(i,j,k)=abs(ur(i,j,k)*sjx+vr(i,j,k)*sjy+
 57                                 * wr(i,j,k)*sjz)+cc*sqrt(sjx**2+sjy**2+sjz**2)
 58              27                  ramdak(i,j,k)=abs(ur(i,j,k)*skx+vr(i,j,k)*sky
 59                                 *    +wr(i,j,k)*skz)+cc*sqrt(skx**2+sky**2+skz**2)

 69              16                  anui(i,j,k)=abs(p(ia-1,j,k)-2*p(ia,j,k)+p(ia+1,j,k))
 70                                 * /(p(ia-1,j,k)+2*p(ia,j,k)+p(ia+1,j,k))
 71
 72                                  if(j.eq.1) then
 73                                    ja=2
 74               1                  elseif(j.eq.j2) then
 75                                    ja=jl
 76                                  else
 77               3                    ja=j
 78                                  endif
 79
 80              27                  anuj(i,j,k)=abs(p(i,ja-1,k)-2*p(i,ja,k)+p(i,ja+1,k))
 81                                 * /(p(i,ja-1,k)+2*p(i,ja,k)+p(i,ja+1,k))
 82
 83                                  if(k.eq.1) then
 84                                    ka=2
 85                                  elseif(k.eq.k2) then
 86                                    ka=kl
 87                                  else
 88              24                    ka=k
```

```
 89                          endif
 90
 91          17              anuk(i,j,k)=abs(p(i,j,ka-1)-2*p(i,j,ka)+p(i,j,ka+1))
 92                         * /(p(i,j,ka-1)+2*p(i,j,ka)+p(i,j,ka+1))
 93
 94           2      110     continue
 95
```

<mpi_exchange 루틴>

```
817                         subroutine mpi_exchange(w1,p1)

835                   !!!!! 1 :   east   -> west, 5*3+1
836                         if(idj.ne.1) THEN
837                           II=0
838                           do K=kstart,kend
839                             do J=jstart,jstart+2
840                               do I=1,I2
841                                 do ir=1,5
842           1                       II=II+1
843           8                       buffs1(II)=w1(ir,i,j,k)
844                                 enddo
845                               enddo
846                             enddo
847                             do J=jstart,jstart+3
848                               do I=1,I2
849                                 II=II+1
850           1                     buffs1(II)=p1(i,j,k)
851           1                   enddo
852                             enddo
853                           enddo
854                         endif

857                   !!!!! 2 :   west   -> east, 5*2+1
858                         if(idj.ne.nprocj) THEN
859                           II=0
860                           do K=kstart,kend
861                             do J=jend-2,jend
862                               do I=1,I2
863                                 do ir=1,5
864                                   II=II+1
865                                   buffs2(II)=w1(ir,i,j,k)
866                                 enddo
867                               enddo
868                             enddo
869                             do J=jend-3,jend
870                               do I=1,I2
871                                 II=II+1
```

```
                        buffs2(II)=p1(i,j,k)
                      enddo
                    enddo
                  enddo
                endif

                itag=1
                length1=I2*nksize*(5*3+4)
                length2=I2*nksize*(5*3+4)

                CALL MPI_ISEND(buffs1,length1,MPI_REAL,
               &               idw,itag,NEW_COMM,isend1,ierr)
                CALL MPI_ISEND(buffs2,length2,MPI_REAL,
               &               ide,itag,NEW_COMM,isend2,ierr)

                CALL MPI_IRECV(buffr1,length1,MPI_REAL,
               &               ide,itag,NEW_COMM,irecv1,ierr)
                CALL MPI_IRECV(buffr2,length2,MPI_REAL,
               &               idw,itag,NEW_COMM,irecv2,ierr)

                CALL MPI_WAIT(isend1,istatus,ierr)
                CALL MPI_WAIT(isend2,istatus,ierr)
                CALL MPI_WAIT(irecv1,istatus,ierr)
                CALL MPI_WAIT(irecv2,istatus,ierr)

                if(idj.ne.nprocj) THEN
                  II=0
                  do K=kstart,kend
                    do J=jend+1,jend+3
                      do I=1,I2
                        do ir=1,5
                          II=II+1
                          w1(ir,i,j,k)=buffr1(II)
                        enddo
                      enddo
                    enddo
                    do J=jend+1,jend+4
                      do I=1,I2
                        II=II+1
                        p1(i,j,k)=buffr1(II)
                      enddo
                    enddo
                  enddo
                endif

                if(idj.ne.1) THEN
                  II=0
                  do K=kstart,kend
                    do J=jstart-3,jstart-1
```

```
921                                 do I=1,I2
922                                   do ir=1,5
923              1                      II=II+1
924             12                      w1(ir,i,j,k)=buffr2(II)
925                                   enddo
926                                 enddo
927                               enddo
928                               do J=jstart-4,jstart-1
929              1                  do I=1,I2
930                                   II=II+1
931              1                    p1(i,j,k)=buffr2(II)
932                                 enddo
933                               enddo
934                             enddo
935                           endif

939                     !!!!! 3 :   north -> south, 5*3+1
940                           if(idk.ne.1) THEN
941                             II=0
942                             do K=kstart,kstart+2
943                               do J=jstart,jend
944                                 do I=1,I2
945                                   do ir=1,5
946              6                    II=II+1
947             15                    buffs3(II)=w1(ir,i,j,k)
948                                   enddo
949                                 enddo
950                               enddo
951                             enddo
952                             do K=kstart,kstart+3
953                               do J=jstart,jend
954                                 do I=1,I2
955                                   II=II+1
956              5                    buffs3(II)=p1(i,j,k)
957                                 enddo
958                               enddo
959                             enddo
960                           endif

963                     !!!!! 4 :   south -> north, 5*2+1
964                           if(idk.ne.nprock) THEN
965                             II=0
966                             do K=kend-2,kend
967                               do J=jstart,jend
968                                 do I=1,I2
969                                   do ir=1,5
970                                     II=II+1
971                                     buffs4(II)=w1(ir,i,j,k)
972                                   enddo
```

```
 973                                 enddo
 974                               enddo
 975                             enddo
 976                             do K=kend-3,kend
 977                               do J=jstart,jend
 978                                 do I=1,I2
 979                                   II=II+1
 980                                   buffs4(II)=p1(i,j,k)
 981                                 enddo
 982                               enddo
 983                             enddo
 984                           endif
 985
 986                           itag=2
 987                           length3=i2*njsize*(5*3+4)
 988                           length4=i2*njsize*(5*3+4)
 989
 990                           CALL MPI_ISEND(buffs4,length4,MPI_REAL,
 991                          &                idn,itag,NEW_COMM,isend4,ierr)
 992                           CALL MPI_ISEND(buffs3,length3,MPI_REAL,
 993                          &                ids,itag,NEW_COMM,isend3,ierr)
 994                           CALL MPI_IRECV(buffr3,length3,MPI_REAL,
 995                          &                idn,itag,NEW_COMM,irecv3,ierr)
 996                           CALL MPI_IRECV(buffr4,length4,MPI_REAL,
 997                          &                ids,itag,NEW_COMM,irecv4,ierr)
 998
 999                           CALL MPI_WAIT(isend3,istatus,ierr)
1000                           CALL MPI_WAIT(isend4,istatus,ierr)
1001                           CALL MPI_WAIT(irecv3,istatus,ierr)
1002                           CALL MPI_WAIT(irecv4,istatus,ierr)
1003
1004                           if(idk.ne.nprock) THEN
1005                             II=0
1006                             do K=kend+1,kend+3
1007                               do J=jstart,jend
1008                                 do I=1,I2
1009                                   do ir=1,5
1010                                     II=II+1
1011                                     w1(ir,i,j,k)=buffr3(II)
1012                                   enddo
1013                                 enddo
1014                               enddo
1015                             enddo
1016                             do K=kend+1,kend+4
1017                               do J=jstart,jend
1018                                 do I=1,I2
1019                                   II=II+1
1020                                   p1(i,j,k)=buffr3(II)
1021                                 enddo
```

```
1022                              enddo
1023                            enddo
1024                          endif
1025
1026                          if(idk.ne.1) THEN
1027                            II=0
1028                            do K=kstart-3,kstart-1
1029                              do J=jstart,jend
1030                                do I=1,I2
1031                                  do ir=1,5
1032          5                         II=II+1
1033         10                         w1(ir,i,j,k)=buffr4(II)
1034                                  enddo
1035                                enddo
1036                              enddo
1037                            enddo
1038                            do K=kstart-4,kstart-1
1039                              do J=jstart,jend
1040                                do I=1,I2
1041                                  II=II+1
1042                                  p1(i,j,k)=buffr4(II)
1043                                enddo
1044                              enddo
1045                            enddo
1046                          endif
```

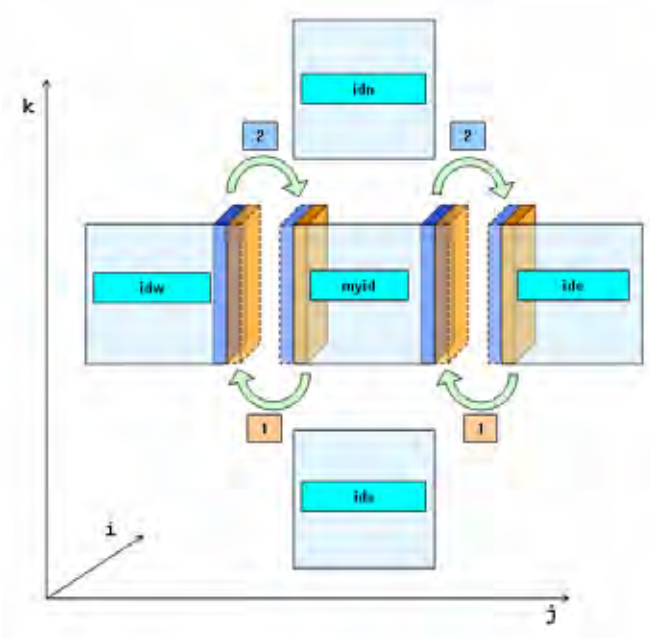
① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

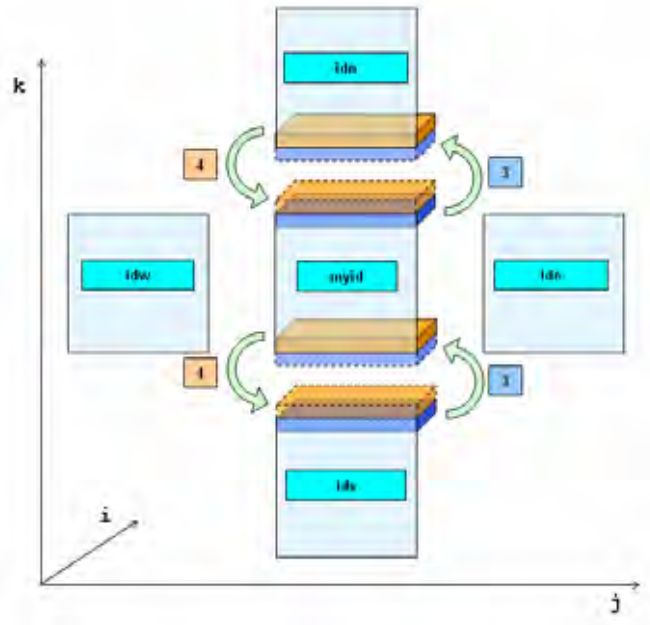
② 이어지는 다음 계산과정에서 추가적인 통신을 피하기 위해 k 및 j loop의 영역을 max(kstart-3,1) ~ min(kend+3,k2) 및 max(jstart-3,1) ~ min(jend+3,j2)로 지정하여 계산을 하게 하였다.

③ 이렇게 계산영역이 확대됨에 따라 자신의 영역 바깥의 데이터 값이 필요하게 되는데, ur, vr, wr array의 경우 이미 이전 계산과정에서 j 및 k 방향으로의 값을 모두 가지고 있는 상태이고, p, w array에 대해서만 필요한 영역의 데이터를 서로 전송하면 되는데 이를 위해 mpi_exchange 루틴을 추가하여 사용하였다.

④ p array의 경우 j 및 k 방향으로 이웃하는 4개 영역의 값을 주고 받게 하였으며, w array의 경우 j 및 k 방향으로 이웃하는 3개 영역의 값을 주고 받게 하여 확대된 계산영역에서 모든 데이터

값을 가질 수 있도록 하였다.





B. do-loop 병렬화

<Serial 최적화 코드>

```
 68                          do k=1,k2
 69                          do j=1,j2
 70                          do i=1,i2
 71          364               ramdaiJa(i,j,k)=ramdai(i,j,k)/ai_arr(i,j,k)
 72           22               ramdajJa(i,j,k)=ramdaj(i,j,k)/aj_arr(i,j,k)
 73          564               ramdakJa(i,j,k)=ramdak(i,j,k)/ak_arr(i,j,k)
 74                          enddo
 75            3             enddo
 76                          enddo
 77
 78                          do k=1,k2
 79                          do j=1,j2
 80                            p_max=p(1,j,k)
 81                            p_min=p(1,j,k)
 82                            r_max=ramdai(1,j,k)
 83                            r_min=ramdai(1,j,k)
 84                            rj_max=ramdaiJa(1,j,k)
 85                            rj_min=ramdaiJa(1,j,k)
 86                            do i=1,i2
 87          100                 if(p(i,j,k).gt.p_max) p_max=p(i,j,k)
 88                              if(p(i,j,k).lt.p_min) p_min=p(i,j,k)
 89           92                 if(ramdai(i,j,k).gt.r_max) r_max=ramdai(i,j,k)
 90           44                 if(ramdai(i,j,k).lt.r_min) r_min=ramdai(i,j,k)
 91           37                 if(ramdaiJa(i,j,k).gt.rj_max) rj_max
                                        =ramdaiJa(i,j,k)
 92           93                 if(ramdaiJa(i,j,k).lt.rj_min) rj_min
                                        =ramdaiJa(i,j,k)
 93                            enddo
 94                            pimax(j,k)=p_max
 95                            pimin(j,k)=p_min
 96            1               ramdaimax(j,k)=r_max
 97                            ramdaimin(j,k)=r_min
 98            1               ramdaiJamax(j,k)=rj_max
 99                            ramdaiJamin(j,k)=rj_min
100                          enddo
101                          enddo
```

<MPI 병렬화 코드>

```
 98                          do k=kstart,kend
 99                          do j=jstart,jend
100                          do i=1,i2
```

```
101          8              ramdaiJa(i,j,k)=ramdai(i,j,k)/ai_arr(i,j,k)
102                         ramdajJa(i,j,k)=ramdaj(i,j,k)/aj_arr(i,j,k)
103         19              ramdakJa(i,j,k)=ramdak(i,j,k)/ak_arr(i,j,k)
104                       enddo
105                       enddo
106                       enddo

110                       do k=kstart,kend
111          1            do j=jstart,jend
112                         p_max=p(1,j,k)
113                         p_min=p(1,j,k)
114                         r_max=ramdai(1,j,k)
115                         r_min=ramdai(1,j,k)
116                         rj_max=ramdaiJa(1,j,k)
117                         rj_min=ramdaiJa(1,j,k)
118                         do i=2,i2
119          3              if(p(i,j,k).gt.p_max) p_max=p(i,j,k)
120                         if(p(i,j,k).lt.p_min) p_min=p(i,j,k)
121          3              if(ramdai(i,j,k).gt.r_max) r_max=ramdai(i,j,k)
122          2              if(ramdai(i,j,k).lt.r_min) r_min=ramdai(i,j,k)
123                         if(ramdaiJa(i,j,k).gt.rj_max) rj_max=ramdaiJa(i,j,k)
124          2              if(ramdaiJa(i,j,k).lt.rj_min) rj_min=ramdaiJa(i,j,k)
125                         enddo
126                         pimax(j,k)=p_max
127                         pimin(j,k)=p_min
128                         ramdaimax(j,k)=r_max
129                         ramdaimin(j,k)=r_min
130                         ramdaiJamax(j,k)=rj_max
131                         ramdaiJamin(j,k)=rj_min
132                       enddo
133                       enddo
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

C. do-loop 병렬화

<Serial 최적화 코드>

```
422                       do k=1,kl
423                         j=1
424                         do i=1,il
425          2                rmax_temp=max(ramdaj(i,j,k),ramdaj(i,j+1,k)
426                        &              ,ramdaj(i,j+2,k),ramdaj(i,j+3,k))
427          7                rmin_temp=min(ramdaj(i,j,k),ramdaj(i,j+1,k)
428                        &              ,ramdaj(i,j+2,k),ramdaj(i,j+3,k))
429          1                ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
```

```
 430                              enddo

 441                              j=jl-1
 442                              do i=1,il
 443            5                   rmax_temp=max(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
 444                           &       ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k))
 445            4                   rmin_temp=min(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
 446                           &       ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k))
 447            5                   ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
 448                              enddo

 459                              do j=3,jl-2
 460            1                 do i=1,il
 461          143                  rmax_temp=max(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
 462                           &       ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k)
                                    ,ramdaj(i,j+3,k))
 463           40                   rmin_temp=min(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
 464                           & ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k)
                                    ,ramdaj(i,j+3,k))
 465           56                   ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
 466                              enddo
 467            1                 enddo
 468
 469                            enddo
```

<MPI 병렬화 코드>

```
 182                            do k=kstart,min(kend,kl)
 183                              if(idj.eq.1) then
 184                                j=1
 185                                do i=1,il
 186                                  rmax_temp=max(ramdaj(i,j,k),ramdaj(i,j+1,k)
 187                           &                    ,ramdaj(i,j+2,k),ramdaj(i,j+3,k))
 188                                  rmin_temp=min(ramdaj(i,j,k),ramdaj(i,j+1,k)
 189                           &                    ,ramdaj(i,j+2,k),ramdaj(i,j+3,k))
 190                                  ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
 191                                enddo

 201                              endif
 202
 203                              if(idj.eq.nprocj) then
 204                                j=jl-1
 205                                do i=1,il
 206            2                   rmax_temp=max(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
 207                           &      ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k))
 208                                      rmin_temp=min(ramdaj(i,j-2,k),ramdaj(i,j-
1,k)
```

```
209                     &        ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k))
210                              ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
211                           enddo

221                        endif

224                        do j=max(jstart-1,3),min(jend,jl-2)
225                        do i=1,il
226           4            rmax_temp=max(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
227                     &   ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k)
                           ,ramdaj(i,j+3,k))
228           4              rmin_temp=min(ramdaj(i,j-2,k),ramdaj(i,j-1,k)
229                     & ,ramdaj(i,j,k),ramdaj(i,j+1,k),ramdaj(i,j+2,k)
                           ,ramdaj(i,j+3,k))
230           2              ramdaj1_2st(i,j,k)=rmax_temp-rmin_temp
231                        enddo
232                        enddo
233
234                     enddo
235
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

② 앞부분에서 계산영역을 확대하여 ramdaj를 계산했기 때문에 이 계산과정에서는 추가적인 통신이 필요없게 된다.

D. 최대, 최소값 계산 부분

<Serial 최적화 코드>

```
103                     do k=1,k2
104                     do i=1,i2
105           2            pjmax(i,k)=p(i,1,k)
106           2            pjmin(i,k)=p(i,1,k)
107           2            ramdajmax(i,k)=ramdaj(i,1,k)
108                        ramdajmin(i,k)=ramdaj(i,1,k)
109           4            ramdajJamax(i,k)=ramdajJa(i,1,k)
110                        ramdajJamin(i,k)=ramdajJa(i,1,k)
111                     enddo
112                     do j=2,j2
113                        do i=1,i2
114          34              if(p(i,j,k).gt.pjmax(i,k)) pjmax(i,k)=p(i,j,k)
115          18              if(p(i,j,k).lt.pjmin(i,k)) pjmin(i,k)=p(i,j,k)
116          26              if(ramdaj(i,j,k).gt.ramdajmax(i,k))
117                     &          ramdajmax(i,k)=ramdaj(i,j,k)
118          56              if(ramdaj(i,j,k).lt.ramdajmin(i,k))
```

```
119          32          &            ramdajmin(i,k)=ramdaj(i,j,k)
120         223                    if(ramdajJa(i,j,k).gt.ramdajJamax(i,k))
121           1          &            ramdajJamax(i,k)=ramdajJa(i,j,k)
122           6                    if(ramdajJa(i,j,k).lt.ramdajJamin(i,k))
123           8          &            ramdajJamin(i,k)=ramdajJa(i,j,k)
124                                 enddo
125           1                   enddo
126                               enddo

128                               do j=1,j2
129                               do i=1,i2
130                                  pkmax(i,j)=p(i,j,1)
131                                  pkmin(i,j)=p(i,j,1)
132           6                      ramdakmax(i,j)=ramdak(i,j,1)
133           3                      ramdakmin(i,j)=ramdak(i,j,1)
134           4                      ramdakJamax(i,j)=ramdakJa(i,j,1)
135           2                      ramdakJamin(i,j)=ramdakJa(i,j,1)
136                               enddo
137                               enddo
138                               do k=1,k2
139           1                   do j=1,j2
140           1                   do i=1,i2
141          76                       if(p(i,j,k).gt.pkmax(i,j)) pkmax(i,j)=p(i,j,k)
142          49                       if(p(i,j,k).lt.pkmin(i,j)) pkmin(i,j)=p(i,j,k)
143          85                       if(ramdak(i,j,k).gt.ramdakmax(i,j))
144           5          &            ramdakmax(i,j)=ramdak(i,j,k)
145          96                       if(ramdak(i,j,k).lt.ramdakmin(i,j))
146           1          &            ramdakmin(i,j)=ramdak(i,j,k)
147         157                       if(ramdakJa(i,j,k).gt.ramdakJamax(i,j))
148                     &            ramdakJamax(i,j)=ramdakJa(i,j,k)
149           2                       if(ramdakJa(i,j,k).lt.ramdakJamin(i,j))
150                     &            ramdakJamin(i,j)=ramdakJa(i,j,k)
151                               enddo
152           1                   enddo
153                               enddo
154
```

<MPI 병렬화 코드>

```
300                               CALL kj1_bcast(p,ramdaj,ramdak,ramdajJa,
301                              &          ramdakJa,ramdaj1_2st,ramdak1_2st)

304                               do k=kstart,kend
305                                  do i=1,i2
306           1                         pjmax2(i,k)=p(i,1,k)
307                                     pjmin2(i,k)=p(i,1,k)
308                                     ramdajmax2(i,k)=ramdaj(i,1,k)
309                                     ramdajmin2(i,k)=ramdaj(i,1,k)
```

```
310         1                  ramdajJamax2(i,k)=ramdajJa(i,1,k)
311                            ramdajJamin2(i,k)=ramdajJa(i,1,k)
312                         enddo

314                         do j=max(jstart,2),jend
315                            do i=1,i2
316         1                     if(p(i,j,k).gt.pjmax2(i,k)) pjmax2(i,k)=p(i,j,k)
317         2                     if(p(i,j,k).lt.pjmin2(i,k)) pjmin2(i,k)=p(i,j,k)
318                               if(ramdaj(i,j,k).gt.ramdajmax2(i,k))
319                        &         ramdajmax2(i,k)=ramdaj(i,j,k)
320         1                     if(ramdaj(i,j,k).lt.ramdajmin2(i,k))
321         1              &         ramdajmin2(i,k)=ramdaj(i,j,k)
322         2                     if(ramdajJa(i,j,k).gt.ramdajJamax2(i,k))
323                        &         ramdajJamax2(i,k)=ramdajJa(i,j,k)
324                               if(ramdajJa(i,j,k).lt.ramdajJamin2(i,k))
325                        &         ramdajJamin2(i,k)=ramdajJa(i,j,k)
326                            enddo
327                         enddo
328                       enddo

330                        CALL MPI_ALLREDUCE(pjmax2(1,mysubnk(1,idk)),
331                        &       pjmax(1,mysubnk(1,idk)),imx*nksize,
332                        &       MPI_REAL,MPI_MAX,MYSUB_COMM_J,ierr)
333                        CALL MPI_ALLREDUCE(pjmin2(1,mysubnk(1,idk)),
334                        &       pjmin(1,mysubnk(1,idk)),imx*nksize,
335                        &       MPI_REAL,MPI_MIN,MYSUB_COMM_J,ierr)
336                        CALL MPI_ALLREDUCE(ramdajmax2(1,mysubnk
337                        & (1,idk)),ramdajmax(1,mysubnk(1,idk)),imx*nksize,
338                        &       MPI_REAL,MPI_MAX,MYSUB_COMM_J,ierr)
339                        CALL MPI_ALLREDUCE(ramdajmin2(1,mysubnk
340                        &(1,idk)),ramdajmin(1,mysubnk(1,idk)),imx*nksize,
341                        &       MPI_REAL,MPI_MIN,MYSUB_COMM_J,ierr)
342                        CALL MPI_ALLREDUCE(ramdajJamax2(1,mysubnk
343                        &(1,idk)),ramdajJamax(1,mysubnk(1,idk)),imx
344                        & *nksize,MPI_REAL,MPI_MAX,MYSUB_COMM_J,ierr)
345                        CALL MPI_ALLREDUCE(ramdajJamin2(1,mysubnk
346                        &(1,idk)),ramdajJamin(1,mysubnk(1,idk)),imx*nksize,
347                        &       MPI_REAL,MPI_MIN,MYSUB_COMM_J,ierr)

350                        do j=jstart,jend
351                        do i=1,i2
352         1                 pkmax2(i,j)=p(i,j,1)
353                           pkmin2(i,j)=p(i,j,1)
354         2                 ramdakmax2(i,j)=ramdak(i,j,1)
355                           ramdakmin2(i,j)=ramdak(i,j,1)
356                           ramdakJamax2(i,j)=ramdakJa(i,j,1)
357                           ramdakJamin2(i,j)=ramdakJa(i,j,1)
358                        enddo
359                        enddo
```

```
362                       do k=max(kstart,2),kend
363                       do j=jstart,jend
364                       do i=1,i2
365           2               if(p(i,j,k).gt.pkmax2(i,j)) pkmax2(i,j)=p(i,j,k)
366                           if(p(i,j,k).lt.pkmin2(i,j)) pkmin2(i,j)=p(i,j,k)
367           1               if(ramdak(i,j,k).gt.ramdakmax2(i,j))
368           2     &             ramdakmax2(i,j)=ramdak(i,j,k)
369           1               if(ramdak(i,j,k).lt.ramdakmin2(i,j))
370                 &             ramdakmin2(i,j)=ramdak(i,j,k)
371           8               if(ramdakJa(i,j,k).gt.ramdakJamax2(i,j))
372                 &             ramdakJamax2(i,j)=ramdakJa(i,j,k)
373           2               if(ramdakJa(i,j,k).lt.ramdakJamin2(i,j))
374                 &             ramdakJamin2(i,j)=ramdakJa(i,j,k)
375                       enddo
376                       enddo
377                       enddo

379                       CALL MPI_ALLREDUCE(pkmax2(1,mysubnj(1,idj)),
380                 &       pkmax(1,mysubnj(1,idj)),imx*njsize,
381                 &       MPI_REAL,MPI_MAX,MYSUB_COMM_K,ierr)
382                       CALL MPI_ALLREDUCE(pkmin2(1,mysubnj(1,idj)),
383                 &       pkmin(1,mysubnj(1,idj)),imx*njsize,
384                 &       MPI_REAL,MPI_MIN,MYSUB_COMM_K,ierr)
385                       CALL MPI_ALLREDUCE(ramdakmax2(1,mysubnj
386                 &(1,idj)),ramdakmax(1,mysubnj(1,idj)),imx*njsize,
387                 &       MPI_REAL,MPI_MAX,MYSUB_COMM_K,ierr)
388                       CALL MPI_ALLREDUCE(ramdakmin2(1,mysubnj
389                 & (1,idj)),ramdakmin(1,mysubnj(1,idj)),imx*njsize,
390                 &       MPI_REAL,MPI_MIN,MYSUB_COMM_K,ierr)
391                       CALL MPI_ALLREDUCE(ramdakJamax2(1,mysubnj
392                 &(1,idj)),ramdakJamax(1,mysubnj(1,idj)),imx*njsize,
393                 &       MPI_REAL,MPI_MAX,MYSUB_COMM_K,ierr)
394                       CALL MPI_ALLREDUCE(ramdakJamin2(1,mysubnj
395                 &(1,idj)),ramdakJamin(1,mysubnj(1,idj)),imx*njsize,
396                 &       MPI_REAL,MPI_MIN,MYSUB_COMM_K,ierr)
```

<kj1_bcast 루틴>

```
272                       subroutine kj1_bcast(p1,ramdaj1,ramdak1,
273                 &ramdajJa1,ramdakJa1,ramdaj1_2st1,ramdak1_2st1)

284                       if(idj.eq.1) then
285                          II=0
286                          DO K=kstart,kend
287                          DO I=1,i2
288                             II=II+1
```

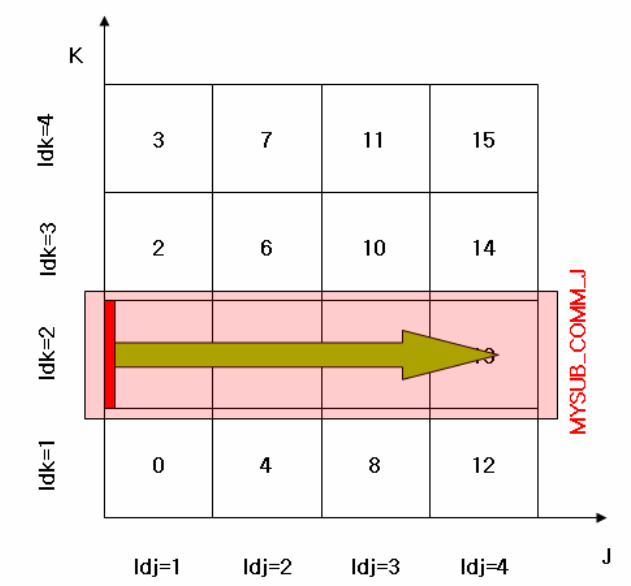
```
289                                 buffj(II)=p1(i,1,k)
290                                 II=II+1
291                                 buffj(II)=ramdaj1(i,1,k)
292                                 II=II+1
293                                 buffj(II)=ramdajJa1(i,1,k)
294                                 II=II+1
295                                 buffj(II)=ramdaj1_2st1(i,1,k)
296                               ENDDO
297                               ENDDO
298                             endif
299
300                             isrc=0
301                             length=4*i2*nksize
302                             CALL MPI_BCAST(buffj,length,MPI_REAL,
303                            &                        isrc,MYSUB_COMM_J,ierr)
304
305                             if(idj.ne.1) then
306                               II=0
307                               DO K=kstart,kend
308                               DO I=1,i2
309                                 II=II+1
310                                 p1(i,1,k)=buffj(II)
311           1                     II=II+1
312                                 ramdaj1(i,1,k)=buffj(II)
313                                 II=II+1
314                                 ramdajJa1(i,1,k)=buffj(II)
315           1                     II=II+1
316                                 ramdaj1_2st1(i,1,k)=buffj(II)
317                               ENDDO
318                               ENDDO
319                             endif

323                             if(idk.eq.1) then
324                               II=0
325                               DO J=jstart,jend
326                               DO I=1,i2
327                                 II=II+1
328                                 buffk(II)=p1(i,j,1)
329                                 II=II+1
330                                 buffk(II)=ramdak1(i,j,1)
331                                 II=II+1
332                                 buffk(II)=ramdakJa1(i,j,1)
333                                 II=II+1
334                                 buffk(II)=ramdak1_2st1(i,j,1)
335                               ENDDO
336                               ENDDO
337                             endif
338
339                             isrc=0
```

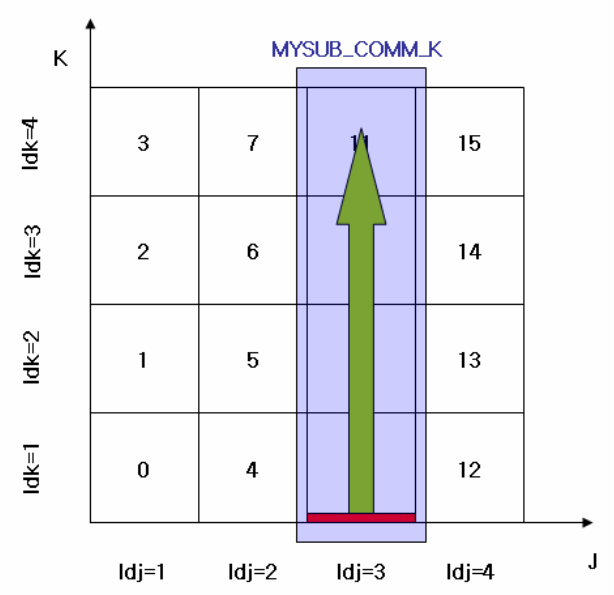
```
340                         length=4*i2*njsize
341                         CALL MPI_BCAST(buffk,length,MPI_REAL,
342                        &                    isrc,MYSUB_COMM_K,ierr)
343
344                         if(idk.ne.1) then
345                           II=0
346                           DO J=jstart,jend
347                           DO I=1,i2
348                             II=II+1
349                             p1(i,j,1)=buffk(II)
350                             II=II+1
351                             ramdak1(i,j,1)=buffk(II)
352                             II=II+1
353           2                 ramdakJa1(i,j,1)=buffk(II)
354                             II=II+1
355                             ramdak1_2st1(i,j,1)=buffk(II)
356                           ENDDO
357                           ENDDO
358                         endif
359
```

① MPI 코드에서 각 rank의 해당 영역부분만 계산을 수행하도록 do-loop의 길이를 수정하였다.

② p, ramdaj, ramdak, ramdajJa, ramdakJa, ramdaj1_2st, ramdak1_2st array에 대해 j=1, k=1일때의 값을 각 rank가 알고 있어야 하기 때문에 jk1_bcast 루틴을 만들어 추가하였다.

③ Jk1_bcast 루틴에서는 해당 변수에 대해 j=1일때의 값을 동일한 MYSUB_COMM_J communicator를 가지는 rank에게 broadcasting을 해주고, 또한 k=1일때의 값을 동일한 MYSUB_COMM_K communicator를 가지는 rank에게 broadcasting을 해준다.





④ 각 영역별 최대(max), 최소(min)값을 계산한 다음 MPI_ALLREDUCE를 통하여 전영역에서 최대, 최소값을 계산하여 각 rank에 다시 저장하게 된다.

※ 위 과정은 j,k 방향으로 p, ramdaj, ramdak, ramdajJa, ramdakJa, ramdaj1_2st, ramdak1_2st array의 최대,최소값을 계산하는 과정이기 때문에 j=1, k=1일때의 값으로 초기화하는 과정 대신 j=jstart, k=kstart로 초기화하도록 수정하게 되면 추가적인 통신이 필요없게 되며, 따라서 jk1_bcast 루틴을 call하여 사용할 필요도 없게 된다.

E. do-loop 병렬화

<Serial 최적화 코드>

```
272                          do k=1,kl
273                            j=1
274                            do i=1,il
275              5               anujmax=max(anuj(i,j,k),anuj(i,j+1,k)
276                            &              ,anuj(i,j+2,k),anuj(i,j+3,k))
277                              epsil2j(i,j,k)=akappa2j(i,k)*anujmax     !!2.52
278              1               epsil_temp=akappa4j(i,k)-epsil2j(i,j,k)
279              1               epsil4j(i,j,k)=max(epsil_temp,0.)
280                            enddo

309                            do j=3,jl-2
310                            do i=1,il
311            192               anujmax=max(anuj(i,j-2,k),anuj(i,j-1,k)
312                            &,anuj(i,j,k),anuj(i,j+1,k),anuj(i,j+2,k) ,anuj(i,j+3,k))
313             13               epsil2j(i,j,k)=akappa2j(i,k)*anujmax     !!2.52
314             28               epsil_temp=akappa4j(i,k)-epsil2j(i,j,k)
315             31               epsil4j(i,j,k)=max(epsil_temp,0.)
316                            enddo
317              4             enddo
318                          enddo
```

<MPI 병렬화 코드>

```
521                          do k=kstart,min(kend,kl)
522                            if(idj.eq.1) then
523                              j=1
524                              do i=1,il
525                                anujmax=max(anuj(i,j,k),anuj(i,j+1,k)
526                            &              ,anuj(i,j+2,k),anuj(i,j+3,k))
527                                epsil2j(i,j,k)=akappa2j(i,k)*anujmax     !!2.52
528                                epsil_temp=akappa4j(i,k)-epsil2j(i,j,k)
529                                epsil4j(i,j,k)=max(epsil_temp,0.)
```

```
530                                 enddo

540                               endif

563                               do j=max(jstart-1,3),min(jend,jl-2)
564                               do i=1,il
565           4                   anujmax=max(anuj(i,j-2,k),anuj(i,j-1,k),anuj(i,j,k)
566                         &      ,anuj(i,j+1,k),anuj(i,j+2,k),anuj(i,j+3,k))
567                                 epsil2j(i,j,k)=akappa2j(i,k)*anujmax     !!2.52
568                                 epsil_temp=akappa4j(i,k)-epsil2j(i,j,k)
569           2                     epsil4j(i,j,k)=max(epsil_temp,0.)
570                               enddo
571                               enddo
572                             enddo
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

F. do-loop 병렬화 및 최대, 최소값 계산 부분

<Serial 최적화 코드>

```
557                             do j=1,jl
558                             do i=1,il
559           3                   ramdak_stmax(i,j)=ramdak1_2st(i,j,1)
560                               ramdak_stmin(i,j)=ramdak1_2st(i,j,1)
561                             enddo
562                             enddo
563
564                             do k=1,kl
565           2                 do j=1,jl
566                             do i=1,il
567         132                   if(ramdak1_2st(i,j,k).gt.ramdak_stmax(i,j))
568          28                &       ramdak_stmax(i,j)=ramdak1_2st(i,j,k)
569          23                   if(ramdak1_2st(i,j,k).lt.ramdak_stmin(i,j))
570           1                &       ramdak_stmin(i,j)=ramdak1_2st(i,j,k)
571                             enddo
572           1                 enddo
573                             enddo
```

<MPI 병렬화 코드>

```
682                             do j=jstart,min(jend,jl)
683                             do i=1,il
```

```
684             1                ramdak_stmax2(i,j)=ramdak1_2st(i,j,1)
685                              ramdak_stmin2(i,j)=ramdak1_2st(i,j,1)
686                           enddo
687                           enddo

691                           do k=max(kstart,2),min(kend,kl)
692                           do j=jstart,min(jend,jl)
693                           do i=1,il
694             4                if(ramdak1_2st(i,j,k).gt.ramdak_stmax2(i,j))
695             1             &      ramdak_stmax2(i,j)=ramdak1_2st(i,j,k)
696             3                if(ramdak1_2st(i,j,k).lt.ramdak_stmin2(i,j))
697                           &      ramdak_stmin2(i,j)=ramdak1_2st(i,j,k)
698                           enddo
699                           enddo
700                           enddo

702                        CALL MPI_ALLREDUCE(ramdak_stmax2(1,mysubnj(1,idj)),
703             &                    ramdak_stmax(1,mysubnj(1,idj)),imx*njsize,
704             &                    MPI_REAL,MPI_MAX,MYSUB_COMM_K,ierr)
705                        CALL MPI_ALLREDUCE(ramdak_stmin2(1,mysubnj(1,idj)),
706             &                    ramdak_stmin(1,mysubnj(1,idj)),imx*njsize,
707             &                    MPI_REAL,MPI_MIN,MYSUB_COMM_K,ierr)
708
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

② J=1, k=1일때의 해당 array 결과는 전 rank에서 값을 가지도록 앞부분에서 jk1_bcast루틴을 이용하여 broadcasting을 해 주었다.

③ 각 영역별 최대(max), 최소(min)값을 계산한 다음 MPI_ALLREDUCE를 통하여 전영역에서 최대, 최소값을 계산하여 각 rank마다 다시 저장하게 된다.

G. do-loop 병렬화

<Serial 최적화 코드>

```
640                           do k=1,kl
641                           do j=1,jl
642                           do i=3,il-2
643                           do ir=1,5
644          1127                disi1_2(ir,i,j,k)=abs(ramdai1_2st(i,j,k))*voli(i,j,k)*
645                           &      (epsil2i(i,j,k)*(w(ir,i+1,j,k)-w(ir,i,j,k)))-
646                           & abs(ramdai1_2st(i,j,k)+ramdai_ba(j,k))*voli(i,j,k)*
```

```
647                                &      (epsil4i(i,j,k)*(diss_b1*(w(ir,i+1,j,k)-w(ir,i,j,k))
648                                &      +diss_b2*(w(ir,i+2,j,k)-w(ir,i-1,j,k))
649                                &      +diss_b3*(w(ir,i+3,j,k)-w(ir,i-2,j,k))))
650                                 enddo
651                                 enddo
652                2                enddo
653                                 enddo

700                                 do k=1,kl
701                                 do j=3,jl-2
702                                 do i=1,il
703                                 do ir=1,5
704             1146                  disj1_2(ir,i,j,k)=abs(ramdaj1_2st(i,j,k))*volj(i,j,k)*
705                                &      (epsil2j(i,j,k)*(w(ir,i,j+1,k)-w(ir,i,j,k)))-
706                                & abs(ramdaj1_2st(i,j,k)+ramdaj_ba(i,k))*volj(i,j,k)*
707                                &      (epsil4j(i,j,k)*(diss_b1*(w(ir,i,j+1,k)-w(ir,i,j,k))
708                                &      +diss_b2*(w(ir,i,j+2,k)-w(ir,i,j-1,k))
709                                &      +diss_b3*(w(ir,i,j+3,k)-w(ir,i,j-2,k))))
710                                 enddo
711                                 enddo
712                1                enddo
713                                 enddo

767                                 do k=3,kl-2
768                                 do j=1,jl
769                                 do i=1,il
770                                 do ir=1,5
771             1364                  disk1_2(ir,i,j,k)=abs(ramdak1_2st(i,j,k))*volk(i,j,k)*
772                                &      (epsil2k(i,j,k)*(w(ir,i,j,k+1)-w(ir,i,j,k)))-
773                             &      abs(ramdak1_2st(i,j,k)+ramdak_ba(i,j))*volk(i,j,k)*
774                                &      (epsil4k(i,j,k)*(diss_b1*(w(ir,i,j,k+1)-w(ir,i,j,k))
775                                &      +diss_b2*(w(ir,i,j,k+2)-w(ir,i,j,k-1))
776                                &      +diss_b3*(w(ir,i,j,k+3)-w(ir,i,j,k-2))))
777                                 enddo
778                                 enddo
779                                 enddo
780                                 enddo
```

<MPI 병렬화 코드>

```
781                                 do k=kstart,min(kend,kl)
782                                 do j=jstart,min(jend,jl)
783                                 do i=3,il-2
784                                 do ir=1,5
785                30                disi1_2(ir,i,j,k)=abs(ramdai1_2st(i,j,k))*voli(i,j,k)*
786                                &      (epsil2i(i,j,k)*(w(ir,i+1,j,k)-w(ir,i,j,k)))-
787                             &      abs(ramdai1_2st(i,j,k)+ramdai_ba(j,k))*voli(i,j,k)*
```

```
788                          &      (epsil4i(i,j,k)*(diss_b1*(w(ir,i+1,j,k)-w(ir,i,j,k))
789                          &      +diss_b2*(w(ir,i+2,j,k)-w(ir,i-1,j,k))
790                          &      +diss_b3*(w(ir,i+3,j,k)-w(ir,i-2,j,k))))
791                           enddo
792                           enddo
793                           enddo
794                           enddo

847                           do k=kstart,min(kend,kl)
848                           do j=max(jstart-1,3),min(jend,jl-2)
849                           do i=1,il
850                           do ir=1,5
851            34               disj1_2(ir,i,j,k)=abs(ramdaj1_2st(i,j,k))*volj(i,j,k)*
852                          &      (epsil2j(i,j,k)*(w(ir,i,j+1,k)-w(ir,i,j,k)))-
853                      &      abs(ramdaj1_2st(i,j,k)+ramdaj_ba(i,k))*volj(i,j,k)*
854                          &      (epsil4j(i,j,k)*(diss_b1*(w(ir,i,j+1,k)-w(ir,i,j,k))
855                          &      +diss_b2*(w(ir,i,j+2,k)-w(ir,i,j-1,k))
856                          &      +diss_b3*(w(ir,i,j+3,k)-w(ir,i,j-2,k))))
857                           enddo
858                           enddo
859                           enddo
860                           enddo

923             1             do k=max(kstart-1,3),min(kend,kl-2)
924                           do j=jstart,min(jend,jl)
925                           do i=1,il
926                           do ir=1,5
927          37               disk1_2(ir,i,j,k)=abs(ramdak1_2st(i,j,k))*volk(i,j,k)*
928                          &      (epsil2k(i,j,k)*(w(ir,i,j,k+1)-w(ir,i,j,k)))-
929                      &      abs(ramdak1_2st(i,j,k)+ramdak_ba(i,j))*volk(i,j,k)*
930                          &      (epsil4k(i,j,k)*(diss_b1*(w(ir,i,j,k+1)-w(ir,i,j,k))
931                          &      +diss_b2*(w(ir,i,j,k+2)-w(ir,i,j,k-1))
932                          &      +diss_b3*(w(ir,i,j,k+3)-w(ir,i,j,k-2))))
933                           enddo
934                           enddo
935                           enddo
936                           enddo
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

H. do-loop 병렬화

<Serial 최적화 코드>

```
824                           do k=2,kl
825             1             do j=2,jl
```

```
826                         do i=2,il
827                         do ir=1,5
828           742           a_diss(ir,i,j,k)=(disi1_2(ir,i,j,k)-disi1_2(ir,i-1,j,k))
829                        &                 +(disj1_2(ir,i,j,k)-disj1_2(ir,i,j-1,k))
830                        &                 +(disk1_2(ir,i,j,k)-disk1_2(ir,i,j,k-1))
831                         enddo
832                         enddo
833             2           enddo
834                         enddo
835
836                         do j=1,j2
837                         do i=1,i2
838                         do ir=1,5
839            26             a_diss(ir,i,j,1)=0.0
840                           a_diss(ir,i,j,k2)=0.0
841                         enddo
842                         enddo
843                         enddo
844
845                         do k=1,k2
846                         do i=1,i2
847                         do ir=1,5
848            15             a_diss(ir,i,1,k)=0.0
849                           a_diss(ir,i,j2,k)=0.0
850                         enddo
851                         enddo
852                         enddo
```

<MPI 병렬화 코드>

```
1012                        do k=max(kstart,2),min(kend,kl)
1013                        do j=max(jstart,2),min(jend,jl)
1014                        do i=2,il
1015                        do ir=1,5
1016          31            a_diss(ir,i,j,k)=(disi1_2(ir,i,j,k)-disi1_2(ir,i-1,j,k))
1017                       &                 +(disj1_2(ir,i,j,k)-disj1_2(ir,i,j-1,k))
1018                       &                 +(disk1_2(ir,i,j,k)-disk1_2(ir,i,j,k-1))
1019                        enddo
1020                        enddo
1021                        enddo
1022                        enddo

1025                        if(idk.eq.1) then
1027                          do j=jstart,jend
1028                          do i=1,i2
1029                          do ir=1,5
1030                            a_diss(ir,i,j,1)=0.0
```

```
1031                                    enddo
1032                                    enddo
1033                                    enddo
1034                                  else if(idk.eq.nprock) then
1036                                    do j=jstart,jend
1037                                    do i=1,i2
1038                                    do ir=1,5
1039              2                       a_diss(ir,i,j,k2)=0.0
1040                                    enddo
1041                                    enddo
1042                                    enddo
1043                                  endif

1045                                  if(idj.eq.1) then
1047                                    do k=kstart,kend
1048                                    do i=1,i2
1049                                    do ir=1,5
1050                                      a_diss(ir,i,1,k)=0.0
1051                                    enddo
1052                                    enddo
1053                                    enddo
1054                                  else if(idj.eq.nprocj) then
1056                                    do k=kstart,kend
1057                                    do i=1,i2
1058                                    do ir=1,5
1059              2                       a_diss(ir,i,j2,k)=0.0
1060                                    enddo
1061                                    enddo
1062                                    enddo
1063                                  endif
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

a_flux_bound 루틴

A. do-loop 병렬화

<Serial 최적화 코드>

```
  3                                   do 101 k=1,k2
  4                                   do 101 j=1,j2
  5                                   i=1
  6             11                    eb(1,j,k,1)=w(1,i,j,k)*ur(i,j,k)
  7                                   eb(2,j,k,1)=w(1,i,j,k)*ur(i,j,k)**2+p(i,j,k)
```

```
 8          49          eb(3,j,k,1)=w(1,i,j,k)*ur(i,j,k)*vr(i,j,k)
 9           2          eb(4,j,k,1)=w(1,i,j,k)*ur(i,j,k)*wr(i,j,k)
10          10          eb(5,j,k,1)=(w(5,i,j,k)+p(i,j,k))*ur(i,j,k)

40           2   101    continue
```

<MPI 병렬화 코드>

```
 8                      do 101 k=kstart,kend
 9                      do 101 j=jstart,jend
10                      i=1
11           2          eb(1,j,k,1)=w(1,i,j,k)*ur(i,j,k)
12                      eb(2,j,k,1)=w(1,i,j,k)*ur(i,j,k)**2+p(i,j,k)
13           5          eb(3,j,k,1)=w(1,i,j,k)*ur(i,j,k)*vr(i,j,k)
14                      eb(4,j,k,1)=w(1,i,j,k)*ur(i,j,k)*wr(i,j,k)
15                      eb(5,j,k,1)=(w(5,i,j,k)+p(i,j,k))*ur(i,j,k)

45                 101  continue
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

a_bound_total 루틴

A. do-loop 병렬화

<Serial 최적화 코드>

```
 4                      do 202 k=1,k2
 5           1          do 202 i=1,i2
 6                      j=1
 7           9          if(iopt_surface(i,k,3).eq.FAR) then
 8                      call non_reflect_far(i,j,k,3)
10           3          elseif(iopt_surface(j,k,3).eq.FAR_WALL) then
11                      call non_reflect_far_wall(i,j,k,3)
13           7          elseif(iopt_surface(i,k,3).eq.WALL) then
14           4          call non_reflect_wall(i,j,k,3)
16           1          elseif(iopt_surface(i,k,3).eq.EXTR) then
17                      call bound_extrap(i,j,k,3)
19                      elseif(iopt_surface(i,k,3).eq.FREE) then
20                      call bound_freestream(i,j,k,3)
21                      elseif(iopt_surface(i,k,3).eq.SYMM) then
22                      call bound_symmetric(i,j,k,3)
```

```
 23                              endif

 43          1       202     continue
 44
```

<MPI 병렬화 코드>

```
  8                             do 202 k=kstart,kend
  9                             do 202 i=1,i2
 10                              if(idj.eq.1) then
 11                                j=1
 12                                if(iopt_surface(i,k,3).eq.FAR) then
 13                                  call non_reflect_far(i,j,k,3)
 14                                elseif(iopt_surface(j,k,3).eq.FAR_WALL) then
 15                                  call non_reflect_far_wall(i,j,k,3)
 16                                elseif(iopt_surface(i,k,3).eq.WALL) then
 17                                  call non_reflect_wall(i,j,k,3)
 18                                elseif(iopt_surface(i,k,3).eq.EXTR) then
 19                                  call bound_extrap(i,j,k,3)
 20                                elseif(iopt_surface(i,k,3).eq.FREE) then
 21                                  call bound_freestream(i,j,k,3)
 22                                elseif(iopt_surface(i,k,3).eq.SYMM) then
 23                                  call bound_symmetric(i,j,k,3)
 24                                endif
 25                              endif

 43                     202     continue
```

① MPI 코드에서 do-loop를 병렬화하여 각 rank의 해당 영역부분만 계산을 수행하도록 하였다.

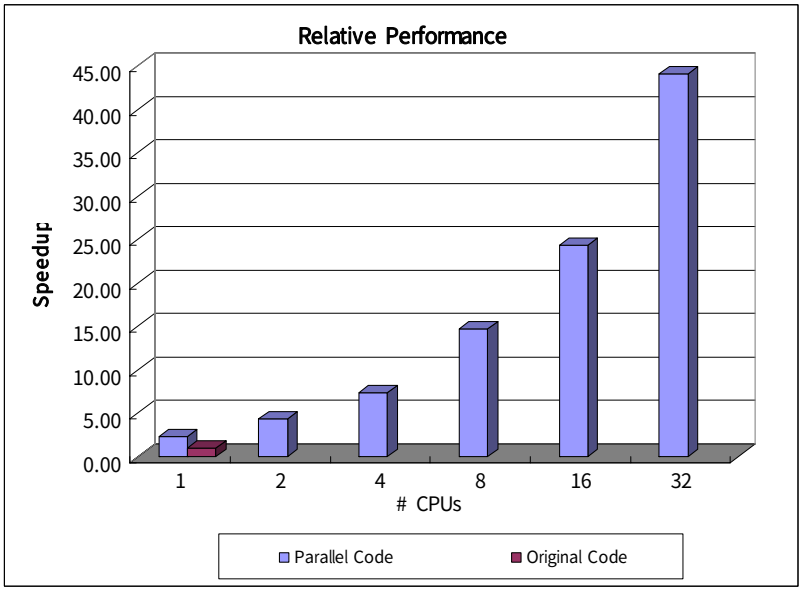
9. MPI 병렬화 결과

코드 전체적으로 봤을 때 deri_et_total, deri_et_total_5, deri_et_total_4 및 deri_zt_total, deri_zt_total_5, deri_zt_total_4 루틴을 수행하기 위해 많은 양의 communication을 수행하게 되어 전체적으로 통신량이 상당히 많게 나타났다. 최대한 통신시간을 줄이기 위해 한 노드에서 shared memory 형태로 통신을 수행하도록 테스트를 하였다.

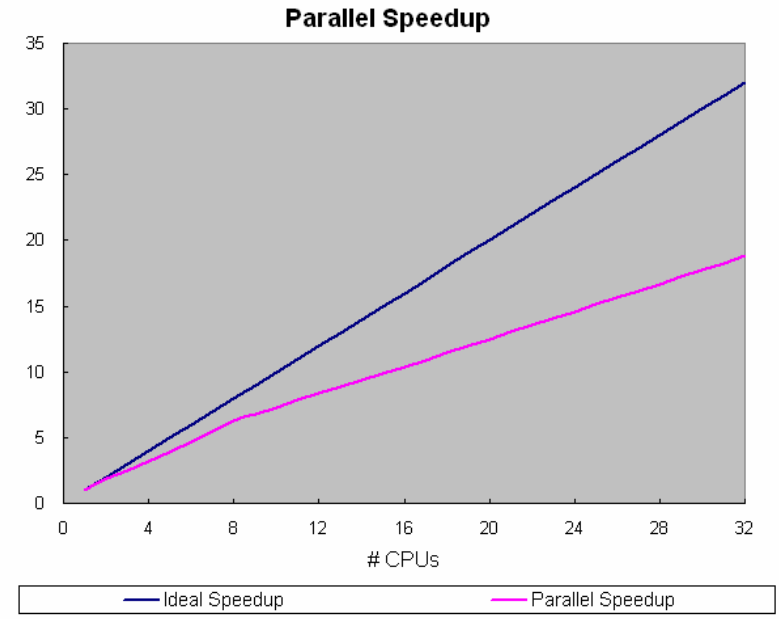
다음 결과는 p690 1.7GHz 32way 시스템에서 iteration 횟수를 200으로 하여 수행한 결과를 정리한 것이다.

< 표 IV. 3 Original 코드와 최적화 및 병렬화 코드의 수행시간 비교 및 speed-up >

#CPUs	Original Code	Tuned Code	MPI Parallelized Code	Parallel Speedup vs Orig Code
1	6530.36	2705.64	2781.36	2.35
2	-	-	1484.23	4.40
4	-	-	885.22	7.38
8	-	-	442.01	14.77
16	-	-	267.68	24.40
32	-	-	147.96	44.13



<그림 IV.7 Original 코드와 병렬코드의 속도 비교 >



< 그림 IV.8 Ideal speed-up과 real speed-up의 비교 >

10. Appendix

Mpitrace 결과 비교 – 32tasks 작업

(rank = 0)

```
-----------------------------------------------------------------
MPI Routine                        #calls        avg. bytes         time(sec)
-----------------------------------------------------------------
MPI_Comm_size                           3              0.0             0.000
MPI_Comm_rank                           4              0.0             0.000
MPI_Isend                            3200          92796.0             0.067
MPI_Irecv                            3200          92796.0             0.042
MPI_Wait                             6400              0.0             1.162
MPI_Bcast                           20804         299369.2            32.831
MPI_Gatherv                             6          59010.0             0.616
MPI_Allreduce                       13600           4639.3             2.934
-----------------------------------------------------------------
total communication time = 37.652 seconds.
total elapsed time       = 146.917 seconds.
user cpu time            = 135.990 seconds.
system time              = 5.230 seconds.
maximum memory size      = 208724 KBytes.

-----------------------------------------------------------------
Message size distributions:

MPI_Isend                    #calls       avg. bytes         time(sec)
                               1600          59052.0             0.055
                               1600         126540.0             0.012

MPI_Irecv                    #calls       avg. bytes         time(sec)
                               1600          59052.0             0.036
                               1600         126540.0             0.006

MPI_Bcast                    #calls       avg. bytes         time(sec)
                                  2              4.0             0.000
                                  1             48.0             0.001
                                  1            160.0             0.009
                                800          12432.0             4.418
                                800          26640.0             8.875
                               9600         198616.0             2.699
                               9600         446886.0            16.829

MPI_Gatherv                  #calls       avg. bytes         time(sec)
                                  3           9240.0             0.301
```

```
                              2          46620.0             0.153
                              1         233100.0             0.162

MPI_Allreduce              #calls       avg. bytes        time(sec)
                             800            20.0             1.299
                            6400          3136.0             0.585
                            6400          6720.0             1.050
```

(rank = 16)

```
-----------------------------------------------------------------
MPI Routine                  #calls       avg. bytes        time(sec)
-----------------------------------------------------------------
MPI_Comm_size                     3              0.0            0.000
MPI_Comm_rank                     4              0.0            0.000
MPI_Isend                      3200          92796.0            0.062
MPI_Irecv                      3200          92796.0            0.032
MPI_Wait                       6400              0.0            1.659
MPI_Bcast                     20804         299369.2           30.603
MPI_Gatherv                       9          62132.0            0.285
MPI_Allreduce                 13600           4639.3            2.198
-----------------------------------------------------------------
total communication time = 34.841 seconds.
total elapsed time       = 146.917 seconds.
user cpu time            = 140.180 seconds.
system time              = 5.070 seconds.
maximum memory size      = 252856 KBytes.

-----------------------------------------------------------------
Message size distributions:

MPI_Isend                  #calls       avg. bytes        time(sec)
                             1600          59052.0             0.049
                             1600         126540.0             0.014

MPI_Irecv                  #calls       avg. bytes        time(sec)
                             1600          59052.0             0.026
                             1600         126540.0             0.007

MPI_Bcast                  #calls       avg. bytes        time(sec)
                                2              4.0             0.021
                                1             48.0             0.023
                                1            160.0             0.009
                              800          12432.0             4.059
                              800          26640.0             8.744
```

```
                          9600        198616.0             2.774
                          9600        446886.0            14.974

MPI_Gatherv             #calls       avg. bytes        time(sec)
                             3          9240.0             0.007
                             2         46620.0             0.005
                             3         68376.0             0.123
                             1        233100.0             0.151

MPI_Allreduce           #calls       avg. bytes        time(sec)
                           800            20.0             0.685
                          6400          3136.0             0.480
                          6400          6720.0             1.033
```

(rank = 31)

```
------------------------------------------------------------------
MPI Routine                #calls       avg. bytes        time(sec)
------------------------------------------------------------------
MPI_Comm_size                   3              0.0             0.000
MPI_Comm_rank                   4              0.0             0.000
MPI_Isend                    3200          97014.0             0.080
MPI_Irecv                    3200          97014.0             0.168
MPI_Wait                     6400              0.0             1.071
MPI_Bcast                   20804         312976.9            20.742
MPI_Gatherv                     6          67440.0             0.613
MPI_Allreduce               13600           4850.1             2.156
------------------------------------------------------------------
total communication time = 24.829 seconds.
total elapsed time       = 146.917 seconds.
user cpu time            = 136.060 seconds.
system time              = 5.330 seconds.
maximum memory size      = 210432 KBytes.

------------------------------------------------------------------
Message size distributions:

MPI_Isend               #calls       avg. bytes        time(sec)
                          3200          97014.0             0.080

MPI_Irecv               #calls       avg. bytes        time(sec)
                          3200          97014.0             0.168

MPI_Bcast               #calls       avg. bytes        time(sec)
                             2              4.0             0.011
                             1             48.0             0.023
                             1            160.0             0.019
```

```
                          800        14208.0          4.987
                          800        26640.0          0.798
                         9600       207644.0          2.612
                         9600       467199.0         12.291

MPI_Gatherv             #calls     avg. bytes      time(sec)
                            3        10560.0          0.297
                            2        53280.0          0.151
                            1       266400.0          0.164

MPI_Allreduce           #calls     avg. bytes      time(sec)
                          800           20.0          0.850
                         6400         3584.0          0.618
                         6400         6720.0          0.687
```