ISBN 89-5884-630-5 93560

슈퍼컴퓨터에서의 CFD 코드의 최적화/병렬화 : IBM POWER4

염민선, 정영균

목 차

CHAPTER I. DOUBLE_CAV 코드

1. Code Information
2. Makefile
3. Flow Chart
4. Profiling 결과
5. hpmcount 결과
6. Serial 최적화 코드 비교 및 분석
    6.1 double_cav 루틴
    6.2 nastok 루틴
    6.3 nastokzone2 루틴, nastokzone3 루틴
    6.4 vsflux 루틴
    6.5 vsfluxz2 루틴, vsfluxz3 루틴
    6.6 bound 루틴
    6.7 turbkom 루틴
    6.8 turbkomzone2 루틴
    6.9 turhs 루틴

6.10 turhszone2 루틴, turhszone3 루틴

6.11 boundturhs 루틴

7. Serial 최적화 결과

8. MPI 병렬화 코드 비교 및 분석

8.1 param_mpi.inc

8.2 decomp_2d 루틴

8.3 mpi_com_pm2 루틴

8.4 mpi_com_pm2 루틴

8.5 mpi_com_output 루틴

9. MPI 병렬화 결과

10. Summary

CHAPTER II. CTBL 코드

1. Code information

2. Makefile

3. Flowchart

4. Profiling 결과

5. hpmcount 결과

6. 순차코드 최적화 비교 및 분석

6.1 indices 루틴

6.2 iniup 루틴

6.3 divcheck 루틴

6.4 cfl 루틴

6.5 getup 루틴

6.6 uhcalc 루틴

6.7 rhs1 루틴

6.8 rhs2 루틴

6.9 rhs3 루틴

6.10 getduh1 루틴

6.11 getduh2 루틴

6.12 tdma 루틴

6.13 rhsdp 루틴

6.14 takedp 루틴

6.15 upcalc 루틴

7. 순차코드 최적화 결과

8. OpenMP 병렬화 코드 비교 및 분석

8.1 iniup 루틴

8.2 divcheck 루틴

8.3 cfl 루틴

8.4 getup 루틴

8.5 bcont 루틴

8.6 uhcalc 루틴

8.7 rhs1 루틴

8.8 getduh1 루틴

9. OpenMP 병렬화 결과

10. Summary

그림 목차

그림. **I.1 Double_cav flow chart**

그림 **I.2 K** 방향으로의 데이터 상호 교환도

그림 **I.3 J**방향으로의 데이터 상호 교환도

그림 **I.4 Zone 2**에서의 데이터 상호 교환도

그림 **I.5 Zone 3**에서의 데이터 상호 교환도

그림.**II.1 DCTBL flow chart**

그림.**II.2** 코드의 성능 비교

그림 **II.3** 실행 시간 비교

그림 **II.4** 병렬화 성능

CHAPTER Ⅰ. DOUBLE_CAV 코드

1. Code information

CFD Simulation 코드, Double_cav 코드

2. Makefile

Original 코드의 Makefile

```
OBJS = bound.o boundzone2.o initcotom.o nastokzone2.o vsflux.o ₩
        boundturb.o boundzone3.o metric.o nastokzone3.o vsfluxz2.o ₩
        boundturbzone2.o datain.o metriczone2.o outputtest.o vsfluxz3.o ₩
        boundturbzone3.o double_cav.o metriczone3.o turb_ca_rktom.o ₩
        grid.o nastok.o turb_ca_tktomz2.o

FFLAGS = -pg -g -O3   -qrealsize=8 -qdpc=e -qsource -qattr=full
FFLAG =   -pg -bmaxdata:0x80000000 -bmaxstack:0x10000000
FC = xlf
double_cav: $(OBJS)
       $(FC) $(FFLAG) -o $@ $(OBJS)
clean:
       rm *.o
```

최적화 코드의 Makefile

```
OBJS = bound.o boundzone2.o initcotom.o nastokzone2.o vsflux.o ₩
        boundturb.o boundzone3.o metric.o nastokzone3.o vsfluxz2.o ₩
        boundturbzone2.o datain.o metriczone2.o outputtest.o vsfluxz3.o ₩
        boundturbzone3.o double_cav.o metriczone3.o turb_ca_rktom.o ₩
        grid.o nastok.o turb_ca_tktomz2.o

FFLAGS = -pg -g -O   -qrealsize=8 -qdpc=e -qsource -qattr=full
FFLAG =   -pg -bmaxdata:0x80000000 -bmaxstack:0x10000000
FC = xlf
double_cav: $(OBJS)
       $(FC) $(FFLAG) -o $@ $(OBJS)
clean:
       rm *.o
```

MPI 병렬화 코드의 Makefile

```
OBJS = datain.o grid.o initcotom.o ₩
        outputtest.o turb_ca_rktom.o turb_ca_tktomz2.o ₩
```

```
          metric.o metriczone2.o metriczone3.o nastok.o   ₩
          nastokzone2.o nastokzone3.o vsflux.o vsfluxz2.o vsfluxz3.o ₩
          bound.o boundturb.o boundzone2.o boundturbzone2.o boundzone3.o ₩
          boundturbzone3.o₩
          mpi_subroutines.o double_cav.o
FFLAGS = -q64 -pg -g -O3 -qsave -qrealsize=8 -qdpc=e -qarch=pwr4
FFLAG =   -q64 -pg -lmass -L/applic/mpitrace/lib -lmpitrace
FC = mpxlf
LD = mpxlf_r
double_cav: $(OBJS)
         $(LD) $(FFLAG) -o $@ $(OBJS)
clean:
         rm *.o
```

실제 Original 코드 및 Serial 최적화 코드에서도 MPI 병렬화 코드의Makefile처럼 컴파일 옵션에 –qarch=pwr4(–O3) 옵션을 추가하고, 링크 옵션에 –lmass를 추가하여 MASS library를 사용하게 될 경우 추가적인 성능향상을 기대할 수 있다.

3. Flow Chart

프로그램의 전반적인 구조는 다음과 같이 Initializing Process, Main Calculation Process인 double_cav 루틴 및 finalizing Process로 구성된다. 일반적인 코드와 마찬가지로 대부분이 계산시간이 Man process인 double_cav 루틴과 그 하위 subroutine들에서 소요되며 특히 NASTOK, NASTOKzone2, NASTOKzone3 및 Turbkom, turbkomzone2, turbkomzone3 루틴에서 대부분의 시간을 차지하여 이 부분에 대한 최적화가 상대적으로 집중적으로 이루어졌다.

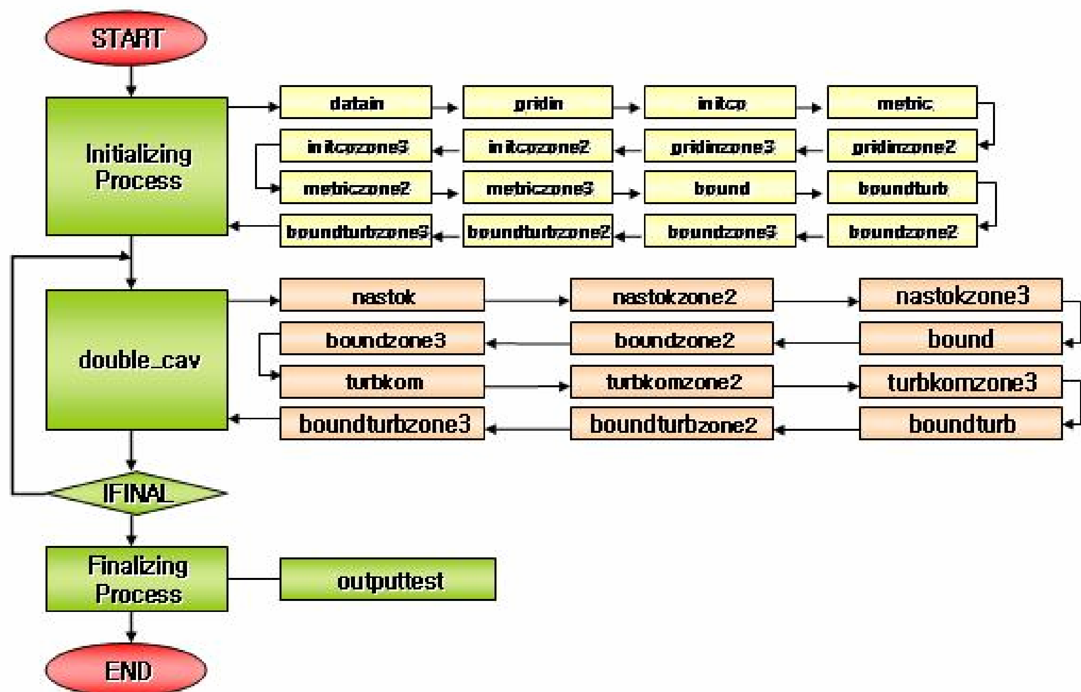


그림. I.1 Double_cav flow chart

4. Profiling 결과

Iteration 횟수를 20으로 했을 경우의 수행 결과이다.

<Original 코드의 profiling 결과>

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
23.4	803.46	803.46	80	10043.25	11740.41	.turhs [5]
22.5	1577.17	773.71	80	9671.38	11368.53	.vsflux [6]
19.9	2261.58	684.41	80	8555.12	19923.66	.nastok [3]
6.0	2466.27	204.69	80	2558.62	14299.03	.turbkom [4]
5.3	2646.52	180.25				._sqrt [8]
4.6	2804.17	157.65	1209856000	0.00	0.00	._pow [7]
2.5	2890.62	86.45	1209856000	0.00	0.00	.expinner2 [10]
2.2	2965.20	74.58	1209856000	0.00	0.00	.loginner2 [12]
2.1	3037.39	72.19	1	72190.00	3123450.00	._main [1]
2.1	3108.02	70.63	80	882.88	1665.58	.nastokzone2 [9]
1.9	3174.07	66.05				.__mcount [14]
1.5	3226.81	52.74	80	659.25	864.70	.turhszone2 [13]
1.3	3272.99	46.18	80	577.25	782.70	.vsfluxz2 [15]
0.9	3302.36	29.37	80	367.12	695.26	.nastokzone3 [16]
0.7	3325.06	22.70	80	283.75	372.89	.turhszone3 [18]
0.6	3344.18	19.12	80	239.00	328.14	.vsfluxz3 [19]
0.3	3355.87	11.69	80	146.12	1010.83	.turbkomzone2 [11]
0.3	3365.44	9.57	45234697	0.00	0.00	.cvtloop [23]
0.2	3373.90	8.46				.__mcount [24]
0.2	3380.92	7.02	45311020	0.00	0.00	.__cvt_r [22]
0.2	3386.43	5.51				.FmtRToQED [20]
0.1	3390.85	4.42	80	55.25	428.14	.turbkomzone3 [17]
0.1	3394.72	3.87				.FormatControl [25]
0.1	3398.17	3.45				._moveeq [26]
0.1	3401.47	3.30	81	40.74	40.74	.bound [27]
0.1	3404.70	3.23				.__stack_pointer [28]
0.1	3407.19	2.49				.qincrement1 [29]
0.1	3409.66	2.47	1	2470.00	2470.00	.metric [30]
0.1	3412.08	2.42				._xlfWriteFmt [31]
0.1	3414.47	2.39				.qincrement [32]
0.1	3416.62	2.15				.WriteUnit [33]
0.1	3418.62	2.00	81	24.69	24.69	.boundturb [34]
0.0	3420.04	1.42	57004536	0.00	0.00	.mf2x1 [35]
0.0	3421.34	1.30				._fill [36]
0.0	3422.53	1.19				.IOWrite [37]

<Serial 최적화 코드의 profiling 결과>

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
26.8	458.47	458.47	80	5730.88	10549.76	.nastok [3]
19.8	796.32	337.85	80	4223.12	4818.88	.vsflux [4]
17.9	1102.75	306.43	80	3830.38	4426.13	.turhs [6]
10.8	1287.65	184.90				._sqrt [7]
3.3	1343.25	55.60	80	695.00	1148.12	.nastokzone2 [9]
3.1	1396.23	52.98	419306240	0.00	0.00	._pow [8]
2.0	1429.98	33.75	419306240	0.00	0.00	.expinner2 [14]

1.8	1461.29	31.31	80	391.38	465.62	.turhszone2 [12]
1.8	1491.60	30.31	80	378.88	453.12	.vsfluxz2 [13]
1.6	1519.49	27.89				.__mcount [16]
1.5	1545.24	25.75	419306240	0.00	0.00	.loginner2 [18]
1.4	1569.64	24.40	80	305.00	600.62	.nastokzone3 [10]
1.4	1592.88	23.24	80	290.50	323.50	.turhszone3 [17]
1.2	1613.89	21.01	80	262.62	295.62	.vsfluxz3 [20]
0.8	1627.08	13.19	1	13190.00	1440890.00	._main.GL [1]
0.8	1640.25	13.17	80	164.62	4590.76	.turbkom [5]
0.6	1649.88	9.63	45234697	0.00	0.00	.cvtloop [23]
0.5	1658.07	8.19				.__mcount [24]
0.4	1665.74	7.67	45311020	0.00	0.00	.__cvt_r [22]
0.3	1671.11	5.37				.FmtRToQED [19]
0.3	1675.46	4.35	80	54.38	377.87	.turbkomzone3 [15]
0.2	1679.08	3.62				._moveeq [25]
0.2	1682.61	3.53				.FormatControl [26]
0.1	1685.11	2.50	1	2500.00	2500.00	.metric [27]
0.1	1687.26	2.15				._xlfWriteFmt [28]
0.1	1689.29	2.03	80	25.38	490.99	.turbkomzone2 [11]
0.1	1691.20	1.91				.WriteUnit [29]
0.1	1693.07	1.87	81	23.09	23.09	.bound [30]
0.1	1694.55	1.48	57027661	0.00	0.00	.mf2x1 [31]
0.1	1695.84	1.29				.IOWrite [32]
0.1	1697.03	1.19				._fill [33]
0.1	1697.95	0.92	81	11.36	11.36	.boundturb [34]
0.1	1698.86	0.91	114055322	0.00	0.00	.pwr10 [35]

<MPI 병렬화 코드의 profiling 결과>

16개 CPU를 이용하여 한 노드에서 수행한 16개의 profiling 결과를 summation하여 나타낸 결과이다.(iteration 횟수 : 20)

%time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
40.3	1052.26	1052.26				.kickpipes [3]
14.5	1429.93	377.67	1280	295.05	544.95	.nastok [4]
12.3	1749.79	319.86	1280	249.89	249.89	.vsflux [5]
9.7	2002.47	252.68	1280	197.41	197.41	.turhs [7]
4.6	2123.18	120.71				.mpci_wait [8]
3.9	2225.27	102.09				pow [9]
1.9	2273.72	48.45	1280	37.85	58.92	.nastokzone2 [10]
1.2	2305.77	32.05	1280	25.04	25.04	.turhszone2 [13]
1.0	2332.74	26.97	1280	21.07	21.07	.vsfluxz2 [14]
0.9	2356.90	24.16	1280	18.88	44.43	.turbkomzone2 [11]
0.8	2378.83	21.93				.mpci_recv [16]
0.8	2400.40	21.57	1280	16.85	28.28	.nastokzone3 [12]
0.8	2421.22	20.82	16	1301.25	1301.25	.initco [17]
0.7	2439.54	18.32	16	1145.00	76186.88	._main [1]
0.6	2455.61	16.07	1280	12.55	210.48	.turbkom [6]
0.6	2471.46	15.85	1280	12.38	12.38	.turhszone3 [21]
0.6	2486.09	14.63	1280	11.43	11.43	.vsfluxz3 [22]
0.4	2496.67	10.58				.gettime [23]
0.4	2507.01	10.34				.__mcount [24]
0.4	2516.91	9.90	45235834	0.00	0.00	.cvtloop [25]
0.3	2524.43	7.52	45312300	0.00	0.00	.__cvt_r [19]
0.2	2530.07	5.64				.__mcount [26]

0.2	2535.42	5.35				.FormatControl [27]
0.2	2540.52	5.10				.FmtRToQED [15]
0.2	2545.50	4.98	3840	1.30	1.30	.mpi_com_pm2 [28]
0.2	2550.15	4.65				.cpfromdev [29]
0.2	2554.56	4.41	7776	0.57	0.57	.cast10 [30]
0.2	2558.89	4.33	16	270.62	270.62	.gridin [31]
0.1	2562.71	3.82				._moveeq [33]
0.1	2566.01	3.30				.cptodevX [34]
0.1	2569.07	3.06				._xlfWriteFmt [36]
0.1	2571.83	2.76				.global_unlock_ppc_mp [37]
0.1	2574.40	2.57	16	160.62	160.62	.initcozone2 [38]
0.1	2576.77	2.37	1296	1.83	2.96	.bound [32]
0.1	2578.80	2.03				.read_real_time [39]
0.1	2580.79	1.99	3840	0.52	0.52	.mpi_com_pm2_2 [40]
0.1	2582.50	1.71	48	35.62	35.62	.mpi_com_output [41]
0.1	2584.19	1.69	1296	1.30	2.44	.boundturb [35]
0.1	2585.82	1.63	16	101.88	101.88	.metric [42]
0.1	2587.40	1.58				.IOWrite [43]
0.0	2588.63	1.23	114009914	0.00	0.00	.pwr10 [45]
0.0	2589.78	1.15				._fill [46]
0.0	2590.92	1.14				.WriteUnit [47]
0.0	2592.00	1.08				.global_lock_ppc_mp [48]
0.0	2592.94	0.94	57004957	0.00	0.00	.mf2x1 [52]
0.0	2593.85	0.91	16	56.88	56.88	.initcozone3 [53]

위의 결과는 iteration수가 적은 경우로 이 경우엔 전체 프로그램 수행시간에서 프로그램 수행시 마지막에 진행되는 output을 write하는 부분 수행 시간이 상대적으로 높게 나온 것이다. 그래서 상대적으로 통신시간이 많아져서 kickpipes 루틴의 비율이 높게 나왔다.
만약 마지막에 진행되는 output을 write하는 부분을 없앨 경우(실제 코드를 수행할 경우엔 아주 많은 iteration을 마친 후 마지막에 한번 이루어지는 결과를 write하는 부분의 비율이 무시될 수 있다.) 다음과 같은 profile 결과가 나온다. 아래의 결과는 iteration 횟수를 200으로 한 경우이다.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
22.5	3852.66	3852.66	12800	300.99	556.79	.nastok [3]
19.1	7126.85	3274.19	12800	255.80	255.80	.vsflux [4]
18.8	10344.38	3217.53				.kickpipes [5]
14.9	12893.35	2548.97	12800	199.14	199.14	.turhs [7]
6.0	13920.85	1027.50				._pow [8]
3.0	14433.36	512.51	12800	40.04	62.34	.nastokzone2 [9]
2.0	14784.29	350.93	12800	27.42	27.42	.turhszone2 [12]
1.7	15069.74	285.45	12800	22.30	22.30	.vsfluxz2 [13]
1.5	15332.44	262.70				.mpci_wait [14]
1.4	15576.28	243.84	12800	19.05	47.02	.turbkomzone2 [10]
1.3	15804.61	228.33	12800	17.84	29.46	.nastokzone3 [11]
1.0	15980.68	176.07	16	11004.38	761395.62	._main [1]
1.0	16149.95	169.27	12800	13.22	13.22	.turhszone3 [16]
1.0	16319.05	169.10				.mpci_recv [17]
0.9	16479.91	160.86	12800	12.57	212.26	.turbkom [6]
0.9	16628.72	148.81	12800	11.63	11.63	.vsfluxz3 [18]
0.3	16682.82	54.10	38400	1.41	1.41	.mpi_com_pm2 [19]

0.3	16726.67	43.85				.cpfromdev [21]
0.3	16770.34	43.67	76896	0.57	0.57	.cast10 [22]
0.2	16802.87	32.53				.cptodevX [26]
0.2	16834.75	31.88				.gettime [27]
0.2	16866.23	31.48	16	1967.50	1967.50	.initco [28]
0.1	16889.68	23.45	12816	1.83	2.97	.bound [25]
0.1	16912.32	22.64				.__mcount [30]
0.1	16933.63	21.31	38400	0.55	0.55	.mpi_com_pm2_2 [31]
0.1	16953.07	19.44	90312290	0.00	0.00	.cvtloop [32]
0.1	16969.66	16.59	12816	1.29	2.43	.boundturb [29]
0.1	16985.96	16.30	90464540	0.00	0.00	.__cvt_r [24]
0.1	16997.76	11.80				.__mcount [34]
0.1	17008.93	11.17				.FormatControl [35]
0.1	17019.95	11.02				.FmtRToQED [20]
0.0	17028.31	8.36	12800	0.65	14.43	.turbkomzone3 [15]
0.0	17036.14	7.83				._xlfWriteFmt [37]
0.0	17043.33	7.19				.memcpy [38]
0.0	17049.42	6.09				.read_real_time [39]
0.0	17055.18	5.76	16	360.00	360.00	.gridin [40]
0.0	17060.56	5.38	12816	0.42	0.99	.boundzone2 [33]
0.0	17065.85	5.29				.global_unlock_ppc_mp [41]
0.0	17070.84	4.99				.cptodev [42]
0.0	17074.73	3.89	16	243.12	243.12	.initcozone2 [43]
0.0	17078.58	3.85	113899412	0.00	0.00	.mf2x1 [44]

각각의 profiling 결과를 보면서 해당 subroutine에 대해 대략적으로 성능 향상이 얼마나 되었는지 파악해 볼 수 있는데, 특히 위에서는 turbs, vsflux, nastok, turbkom 등의 subroutine에서 성능향상이 크게 되었음을 알 수 있다.

5. hpmcount 결과

컴파일된 실행파일을 hpmcount와 같이 수행하면 다음과 같은 코드 수행에 관한 정보를 볼 수 있다.

<Original 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 3435.172295 seconds

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

Total amount of time in user mode              : 3431.540000 seconds
Total amount of time in system mode            : 1.800000 seconds
Maximum resident set size                      : 858780 Kbytes
Average shared memory use in text segment      : 1375783 Kbytes*sec
Average unshared memory use in data segment    : 2147483647 Kbytes*sec
Number of page faults without I/O activity     : 214828
Number of page faults with I/O activity        : 19
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           : 2403
Number of involuntary context switches         : 3399

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

 PM_FPU_FDIV (FPU executed FDIV instruction)                    :    24159504153
 PM_FPU_FMA (FPU executed multiply-add instruction)             :   392718097185
 PM_FPU0_FIN (FPU0 produced a result)                           :   560558058362
 PM_FPU1_FIN (FPU1 produced a result)                           :   547644045427
 PM_CYC (Processor cycles)                                      :  5850328258706
 PM_FPU_STF (FPU executed store instruction)                    :   190280736037
 PM_INST_CMPL (Instructions completed)                          :  2116927321666
 PM_LSU_LDF (LSU executed Floating Point load instruction)      :   495057916598

 Utilization rate                          :       99.946 %
 Load and store operations                 :   685338.653 M
 MIPS                                      :      616.251
 Instructions per cycle                    :        0.362
 HW Float points instructions per Cycle    :        0.189
 Floating point instructions + FMAs        :  1310639.465 M
 Float point instructions + FMA rate       :      381.535 Mflip/s
 FMA percentage                            :       59.928 %
 Computation intensity                     :        1.912
```

<Serial 최적화 코드의 hpmcount 결과>

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 1713.916778 seconds

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

Total amount of time in user mode              : 1706.740000 seconds
Total amount of time in system mode            : 2.780000 seconds
Maximum resident set size                      : 1450336 Kbytes
Average shared memory use in text segment      : 527682 Kbytes*sec
Average unshared memory use in data segment    : 2147483647 Kbytes*sec
Number of page faults without I/O activity     : 362785
Number of page faults with I/O activity        : 17
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           : 2527
Number of involuntary context switches         : 1696

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

 PM_FPU_FDIV (FPU executed FDIV instruction)                        :     51697911259
 PM_FPU_FMA (FPU executed multiply-add instruction)                 :    388444889895
 PM_FPU0_FIN (FPU0 produced a result)                               :    512064683762
 PM_FPU1_FIN (FPU1 produced a result)                               :    484707872045
 PM_CYC (Processor cycles)                                          :   2913083861638
 PM_FPU_STF (FPU executed store instruction)                        :    161491119527
 PM_INST_CMPL (Instructions completed)                              :   1806708137807
 PM_LSU_LDF (LSU executed Floating Point load instruction)          :    423403344143

 Utilization rate                          :          99.746 %
 Load and store operations                 :      584894.464 M
 MIPS                                      :        1054.140
 Instructions per cycle                    :           0.620
 HW Float points instructions per Cycle    :           0.342
 Floating point instructions + FMAs        :     1223726.326 M
 Float point instructions + FMA rate       :         713.994 Mflip/s
 FMA percentage                            :          63.486 %
 Computation intensity                     :           2.092
```

<MPI 병렬화 코드의 hpmcount 결과>

16개의 CPU를 사용하여 수행했을 때 0번 rank의 hpmcount 결과이다.

```
hpmcount (V 2.4.3) summary

Total execution time (wall clock time): 168.441202 seconds

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

Total amount of time in user mode              : 159.510000 seconds
```

```
Total amount of time in system mode              : 3.050000 seconds
Maximum resident set size                        : 782256 Kbytes
Average shared memory use in text segment        : 75337 Kbytes*sec
Average unshared memory use in data segment      : 96757984 Kbytes*sec
Number of page faults without I/O activity       : 196015
Number of page faults with I/O activity          : 10
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                      : 1
Number of voluntary context switches             : 3556
Number of involuntary context switches           : 165

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

 PM_FPU_FDIV (FPU executed FDIV instruction)                      :    1675349197
 PM_FPU_FMA (FPU executed multiply-add instruction)               :    24748546598
 PM_FPU0_FIN (FPU0 produced a result)                             :    39481932506
 PM_FPU1_FIN (FPU1 produced a result)                             :    37085071423
 PM_CYC (Processor cycles)                                        :    277081326850
 PM_FPU_STF (FPU executed store instruction)                      :    15938383770
 PM_INST_CMPL (Instructions completed)                            :    225997300230
 PM_LSU_LDF (LSU executed Floating Point load instruction)        :    33617856069

 Utilization rate                              :     96.537 %
 Load and store operations                     :     49556.240 M
 MIPS                                          :     1341.698
 Instructions per cycle                        :     0.816
 HW Float points instructions per Cycle        :     0.276
 Floating point instructions + FMAs            :     85377.167 M
 Float point instructions + FMA rate           :     506.866 Mflip/s
 FMA percentage                                :     57.975 %
 Computation intensity                         :     1.723
```

Original 코드와 Serial 최적화 코드의 hpmcount 결과를 비교해 보면 먼저 original 코드의 경우 부동소수점 연산횟수가 1310639M정도 되고 부동소수점 연산 속도가 381.5MFLOPS 정도 되며, Serial 최적화 코드의 경우 부동소수점 연산횟수가 1223726M 정도 되고 부동소수점 연산 속도가 714.0MFLOPS 정도 된다. 최적화 결과 연산회수가 조금 줄어들었고 연산속도는 거의 2배가까이 빨라졌음을 알수 있다.

MPI 병렬화 코드의 경우 Serial 최적화 코드에 비해 부동소수점 연산 속도가 714.0MFLOPS에서 506.9MFLOPS로 감소했지만 MPI 병렬화로 인해 0번 rank가 계산해야 하는 부동 소수점 연산횟수가 약 14배 정도 줄어들어 전체적인 계산시간이 10배정도 빨라졌다.

6. Serial 최적화 코드 비교 및 분석

다음 subroutine들에 대해서 최적화를 진행하였다.

Main Calculation Process

	Subroutine 명	Original 코드의 tick수	최적화 코드의 tick수	Speed-up
1	double_cav	7,219	1,319	5.47
2	nastok	68,441	45,847	1.49
3	vsflux	77,371	33,785	2.29
4	nastokzone2	7,063	5,560	1.27
5	vsfluxz2	4,618	3,031	1.52
6	nastokzone3	2,937	2,440	1.20
7	vsfluxz3	1,912	2,101	0.91
8	bound	330	187	1.76
9	turbkom	20,469	1,317	15.54
10	turhs	80,346	30,643	2.62
11	turbkomzone2	1,169	203	5.76
12	turhszone2	5,274	3,131	1.68
13	turhszone3	2,270	2,324	0.98
14	boundturb	200	92	2.17
total	-	279,619	131,980	2.12

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 2.12배의 성능향상이 되었음을 알 수 있다. 각 subroutine에 대해서 하나씩 분석해 보면 다음과 같다.

1. double_cav 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
119                     do 100 i=2,il
```

```
120        11        do 100 j=2,jl
121                  do 100 k=2,kl
122      2828        ur(i,j,k)=w(2,i,j,k)/w(1,i,j,k)
123        87        vr(i,j,k)=w(3,i,j,k)/w(1,i,j,k)
124      2083        wr(i,j,k)=w(4,i,j,k)/w(1,i,j,k)
125      1410        p(i,j,k)=(w(5,i,j,k)-0.5*w(1,i,j,k)*(ur(i,j,k)**2
126                 *    +vr(i,j,k)**2+wr(i,j,k)**2))*(gamm)
127         7        t(i,j,k)=p(i,j,k)/w(1,i,j,k)
128         8    100 continue
```

```
135                  do 101 i=2,ilz2
136         4        do 101 j=2,jlz2
137                  do 101 k=2,klz2
138       204        urz2(i,j,k)=wz2(2,i,j,k)/wz2(1,i,j,k)
139         9        vrz2(i,j,k)=wz2(3,i,j,k)/wz2(1,i,j,k)
140       105        wrz2(i,j,k)=wz2(4,i,j,k)/wz2(1,i,j,k)
141        75        pz2(i,j,k)=(wz2(5,i,j,k)-0.5*wz2(1,i,j,k)*(urz2(i,j,k)**2
142                 *    +vrz2(i,j,k)**2+wrz2(i,j,k)**2))*(gamm)
143         1        tz2(i,j,k)=pz2(i,j,k)/wz2(1,i,j,k)
144         1    101 continue
```

```
149                  do 102 i=2,ilz3
150         1        do 102 j=2,jlz3
151                  do 102 k=2,klz3
152        51        urz3(i,j,k)=wz3(2,i,j,k)/wz3(1,i,j,k)
153         9        vrz3(i,j,k)=wz3(3,i,j,k)/wz3(1,i,j,k)
154        26        wrz3(i,j,k)=wz3(4,i,j,k)/wz3(1,i,j,k)
155        15        pz3(i,j,k)=(wz3(5,i,j,k)-0.5*wz3(1,i,j,k)*(urz3(i,j,k)**2
156                 *    +vrz3(i,j,k)**2+wrz3(i,j,k)**2))*(gamm)
157                  tz3(i,j,k)=pz3(i,j,k)/wz3(1,i,j,k)
158              102 continue
```

<최적화 코드>

```
119                  do 100 k=2,kl
120                  do 100 j=2,jl
121                  do 100 i=2,il
122       142        ur(i,j,k)=w(2,i,j,k)/w(1,i,j,k)
123       353        vr(i,j,k)=w(3,i,j,k)/w(1,i,j,k)
124        23        wr(i,j,k)=w(4,i,j,k)/w(1,i,j,k)
125       110        p(i,j,k)=(w(5,i,j,k)-0.5*w(1,i,j,k)*(ur(i,j,k)**2
126                 *    +vr(i,j,k)**2+wr(i,j,k)**2))*(gamm)
127       264        t(i,j,k)=p(i,j,k)/w(1,i,j,k)
128              100 continue
```

```
135                  do 101 k=2,klz2
136                  do 101 j=2,jlz2
137                  do 101 i=2,ilz2
138         6        urz2(i,j,k)=wz2(2,i,j,k)/wz2(1,i,j,k)
```

```
139          48           vrz2(i,j,k)=wz2(3,i,j,k)/wz2(1,i,j,k)
140           4           wrz2(i,j,k)=wz2(4,i,j,k)/wz2(1,i,j,k)
141           3           pz2(i,j,k)=(wz2(5,i,j,k)-0.5*wz2(1,i,j,k)*(urz2(i,j,k)**2
142                      *    +vrz2(i,j,k)**2+wrz2(i,j,k)**2))*(gamm)
143          30           tz2(i,j,k)=pz2(i,j,k)/wz2(1,i,j,k)
144                   101 continue
```

```
149                       do 102 k=2,klz3
150                       do 102 j=2,jlz3
151                       do 102 i=2,ilz3
152           4           urz3(i,j,k)=wz3(2,i,j,k)/wz3(1,i,j,k)
153          19           vrz3(i,j,k)=wz3(3,i,j,k)/wz3(1,i,j,k)
154           1           wrz3(i,j,k)=wz3(4,i,j,k)/wz3(1,i,j,k)
155           6           pz3(i,j,k)=(wz3(5,i,j,k)-0.5*wz3(1,i,j,k)*(urz3(i,j,k)**2
156                      *    +vrz3(i,j,k)**2+wrz3(i,j,k)**2))*(gamm)
157          15           tz3(i,j,k)=pz3(i,j,k)/wz3(1,i,j,k)
```

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

2. nastok 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
274                       do 200 k=2,kl
275                        do 200 i=2,il
276           7            do 20 j=1,j2
277
278         757              pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
279        1101              pi=sqrt(etx(i,j,k)**2+ety(i,j,k)**2+etz(i,j,k)**2)
280         310              ex=etx(i,j,k)/pi
281          64              ey=ety(i,j,k)/pi
282                          ez=etz(i,j,k)/pi

430                       do 823 ir=1,idim

440         530             g(ir,j)=fprc*vol(i,j,k)
441         152             gm(ir,j)=fmrc*vol(i,j,k)
442                   823 continue
443          50        20 continue

446           1            do 21 j=2,jl
447                        m=j-1
448           4            if((j.eq.2)) then
449                        do ir=1,5
450          33              dw(ir,i,j,k)=dw(ir,i,j,k)+g(ir,j)-g(ir,j-1)
451                        enddo

477                       endif
```

```
478                        21 continue

513           1           200 continue
```

```
514                             do 300 i=2,il
515                               do 300 j=2,jl
516           2                   do 30 k=1,k2
517        4035                     pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
518        6609                     pi=sqrt(ztx(i,j,k)**2+zty(i,j,k)**2+ztz(i,j,k)**2)
519        2675                     zx=ztx(i,j,k)/pi
520          59                     zy=zty(i,j,k)/pi
521                                 zz=ztz(i,j,k)/pi

668                             do 833 ir=1,idim

678           8                   h(ir,k)=gprc*vol(i,j,k)
679         150                   hm(ir,k)=gmrc*vol(i,j,k)
680                       833 continue

682          22            30 continue

717           1                   do 33 k=2,kl
718          12                   n=k+1
719         159                   if((k.eq.kl)) then
720                               do ir=1,5
721                                 dw(ir,i,j,k)=dw(ir,i,j,k)+(hm(ir,n)-hm(ir,k))
722                               enddo

748                             endif
749                        33 continue
750           1           300 continue
```

<최적화 코드>

```
276                             do 200 k=2,kl
277                               do 20 j=1,j2
278           1                   do i=2,il
279
280          51                     pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
281         304                     pi=sqrt(etx(i,j,k)**2+ety(i,j,k)**2+etz(i,j,k)**2)
282                                 ex=etx(i,j,k)/pi
283         380                     ey=ety(i,j,k)/pi
284          39                     ez=etz(i,j,k)/pi

432                             do 823 ir=1,idim

442          87                   gj(ir,i,j)=fprc*vol(i,j,k)
443         297                   gmj(ir,i,j)=fmrc*vol(i,j,k)
444                       823 continue
445          24                 enddo
446                        20 continue

449                               do 21 j=2,jl
450                               m=j-1
451                               do i=2,il
```

```
452         4            if((j.eq.2)) then
453                      do ir=1,5
454        11              dw(ir,i,j,k)=dw(ir,i,j,k)+gj(ir,i,j)-gj(ir,i,j-1)
455                      enddo

481                     endif
482                     enddo
483                  21 continue

521                 200 continue
```

```
522                      do 30 k=1,k2
523                      do j=2,jl
524                      do i=2,il
525       118              pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
526       334              pi=sqrt(ztx(i,j,k)**2+zty(i,j,k)**2+ztz(i,j,k)**2)
527                        zx=ztx(i,j,k)/pi
528       433              zy=zty(i,j,k)/pi
529        34              zz=ztz(i,j,k)/pi

676                     do 833 ir=1,idim

686        88            hk(ir,i,j,k)=gprc*vol(i,j,k)
687       356            hmk(ir,i,j,k)=gmrc*vol(i,j,k)
688                 833 continue

690        28           enddo
691         1           enddo
692                  30 continue

731                      do 33 k=2,kl
732                      n=k+1
733                      do j=2,jl
734                      do i=2,il
735         6            if((k.eq.kl)) then
736                      do ir=1,5
737         4              dw(ir,i,j,k)=dw(ir,i,j,k)+(hmk(ir,i,j,n)-hmk(ir,i,j,k))
738                      enddo

766                     enddo
767                     enddo
768                  33 continue
```

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

② Loop 순서를 변경하기 위해 original 코드의 2차원 g, gm 및 h, hm array 대신 Serial 최적화 코드에서는 3차원 gj, gmj array 및 4차원 hk, hmk array로 선언하여 사용하였다.

3. nastokzone2 루틴, nastokzone3 루틴

Nastokzone2, nastokzone3 루틴은 nastok 루틴과 변수이름 정도만 차이날뿐 거의 비슷한 루틴이기 때문에 아래의 nastok 루틴을 최적화 한것과 비슷하게 수정하였다.

4. vsflux 루틴

A. 반복 계산되는 visc array를 한번만 계산

<Original 코드>

```
 22                       do 100 k=2,kl
 26          1            do 100 j=2,jl
 30                       do i = 1,i2
 31         30            visc(i) = (t(i,j,k)/tfree)**vex/res
 32                       enddo

170         34            vco=(visc(i)*fac1+visc(ip)*fac2)/pr
171                     *    +(emu(i,j,k)*fac1+emu(ip,j,k)*fac2)/prt

196          1        100 continue

200                       do 200 k=2,kl
203                       do 200 i=2,il
207                       do j = 1,j2
208        138            visc(j) = (t(i,j,k)/tfree)**vex/res
209                       enddo

346        271              vco=(visc(j)*fac1+visc(jp)*fac2)/pr
347                     *    +(emu(i,j,k)*fac1+emu(i,jp,k)*fac2)/prt

372          9        200 continue

376                       do 300 i=2,il
379                       do 300 j=2,jl
383          2            do k = 1,k2
384        420            visc(k) = (t(i,j,k)/tfree)**vex/res
385                       enddo

523        118            vco=(visc(k)*fac1+visc(kp)*fac2)/pr
524                     *    +(emu(i,j,k)*fac1+emu(i,j,kp)*fac2)/prt

548          4        300 continue
```

<최적화 코드>

```
 23                       do k = 1,k2
 24                       do j = 1,j2
 25                       do i = 1,i2
 26        357            visc(i,j,k) = (t(i,j,k)/tfree)**vex/res
 27                       enddo
 28                       enddo
 29                       enddo

 30                       do 100 k=2,kl
```

```
34          6           do 100 j=2,jl
39                      do 10 i=1,il

175       120            vco=(visc(i,j,k)*fac1+visc(ip,j,k)*fac2)/pr
176                    *    +(emu(i,j,k)*fac1+emu(ip,j,k)*fac2)/prt

195        44        10 continue
202        12       100 continue

206                     do 200 k=2,kl
216         3           do 20 j=1,jl
218                     do i=2,il

355        58             vco=(visc(i,j,k)*fac1+visc(i,jp,k)*fac2)/pr
356                    *    +(emu(i,j,k)*fac1+emu(i,jp,k)*fac2)/prt

376         1           enddo
377         3        20 continue
385                 200 continue

398                     do 30 k=1,kl
400         5           do j=2,jl
403                     do i=2,il

540       180            vco=(visc(i,j,k)*fac1+visc(i,j,kp)*fac2)/pr
541                    *    +(emu(i,j,k)*fac1+emu(i,j,kp)*fac2)/prt

559                     enddo
560         3           enddo
561                  30 continue
```

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

B. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
200                     do 200 k=2,kl
201                     kp=k+1
202                     km=k-1
203                     do 200 i=2,il
204                     ip=i+1
205                     im=i-1

211         7           do 20 j=1,jl
212                     jp=j+1
213       109            vol2=vol(i,j,k)+vol(i,jp,k)

352       191            fv(1,j)=dn*(d1*ui+d5*vi+d6*wi+b1*uj+b5*vj+b7*wj
353                    *       +f1*uk+f7*vk+f9*wk) -2./3.*r1*tk1*sjx*vol22
354       211            fv(2,j)=dn*(d7*ui+d2*vi+d8*wi+b5*uj+b2*vj+b6*wj
355                    *        +f5*uk+f2*vk+f10*wk) -2./3.*r1*tk1*sjy*vol22
```

```
356       184        fv(3,j)=dn*(d9*ui+d10*vi+d3*wi+b7*uj+b6*vj+b3*wj
357                *         +f6*uk+f8*vk+f3*wk)    -2./3.*r1*tk1*sjz*vol22
358      1308    fv(4,j)=dn*(0.5*(d1*uui+d2*vvi+d3*wwi)+d5*u1*vi+d6*u1*wi+d7*v1*ui
359                *    +d8*v1*wi+d9*w1*ui+d10*w1*vi
360                *  +0.5*(b1*uuj+b2*vvj+b3*wwj)+b5*uvj+b6*vwj+b7*uwj
361                *  +0.5*(f1*uuk+f2*vvk+f3*wwk)
362                *  +f5*v1*uk+f6*w1*uk+f7*u1*vk+f8*w1*vk+f9*u1*wk+f10*v1*wk
363                * )
364                *  +ck*vol22*(d4*tti+b4*ttj+f4*ttk)
365                *  -2./3.*r1*tk1*(u1*sjx+v1*sjy+w1*sjz)*vol22
366              20 continue
367         4        do 21 j=2,jl
368                  do ir=1,4
369      1692        dw(ir+1,i,j,k)=dw(ir+1,i,j,k)-(fv(ir,j)-fv(ir,j-1))
370                  enddo
371              21 continue
372         9     200 continue
```

```
376                  do 300 i=2,il
377                  ip=i+1
378                  im=i-1
379                  do 300 j=2,jl
380                  jp=j+1
381                  jm=j-1

387         8        do 30 k=1,kl
388                  kp=k+1
389      3406        vol2=vol(i,j,k)+vol(i,j,kp)

529       253        fv(1,k)=dn*(e1*ui+e5*vi+e6*wi+f1*uj+f5*vj+f6*wj
530                *       +c1*uk+c5*vk+c7*wk) -2./3.*r1*tk1*skx*vol22
531       161        fv(2,k)=dn*(e7*ui+e2*vi+e8*wi+f7*uj+f2*vj+f8*wj
532                *         +c5*uk+c2*vk+c6*wk) -2./3.*r1*tk1*sky*vol22
533       210        fv(3,k)=dn*(e9*ui+e10*vi+e3*wi+f9*uj+f10*vj+f3*wj
534                *         +c7*uk+c6*vk+c3*wk) -2./3.*r1*tk1*skz*vol22
535       901     fv(4,k)=dn*(0.5*(e1*uui+e2*vvi+e3*wwi)+e5*u1*vi+e6*u1*wi+e7*v1*ui
536                *    +e8*v1*wi+e9*w1*ui+e10*w1*vi
537             *  +0.5*(f1*uuj+f2*vvj+f3*wwj)+f5*u1*vj+f6*u1*wj+f7*v1*uj+f8*v1*wj
538                *  +f9*w1*uj+f10*w1*vj  +0.5*(c1*uuk+c2*vvk+c3*wwk)
539                *  +c5*uvk+c6*vwk+c7*uwk)
540                *  +ck*vol22*(e4*tti+f4*ttj+c4*ttk)
541                *       -2./3.*r1*tk1*(u1*skx+v1*sky+w1*skz)*vol22
542       186      30 continue
543        15        do 31 k=2,kl
544                  do ir=1,4
545      3777        dw(ir+1,i,j,k)=dw(ir+1,i,j,k)-(fv(ir,k)-fv(ir,k-1))
546                  enddo
547              31 continue
548         4     300 continue
```

<최적화 코드>

```
206                  do 200 k=2,kl
207                  kp=k+1
208                  km=k-1
```

```
216          3          do 20 j=1,jl
217                     jp=j+1
218                     do i=2,il
219                     ip=i+1
220                     im=i-1
221                 !
222          8          vol2=vol(i,j,k)+vol(i,jp,k)

362        492          fvj(1,i,j)=dn*(d1*ui+d5*vi+d6*wi+b1*uj+b5*vj+b7*wj
363                   *        +f1*uk+f7*vk+f9*wk) -2./3.*r1*tk1*sjx*vol22
364        416          fvj(2,i,j)=dn*(d7*ui+d2*vi+d8*wi+b5*uj+b2*vj+b6*wj
365                   *          +f5*uk+f2*vk+f10*wk) -2./3.*r1*tk1*sjy*vol22
366        149          fvj(3,i,j)=dn*(d9*ui+d10*vi+d3*wi+b7*uj+b6*vj+b3*wj
367                   *          +f6*uk+f8*vk+f3*wk)    -2./3.*r1*tk1*sjz*vol22
368        908          fvj(4,i,j)=dn*(0.5*(d1*uui+d2*vvi+d3*wwi)+d5*u1*vi
369                   *    +d6*u1*wi+d7*v1*ui+d8*v1*wi+d9*w1*ui+d10*w1*vi
370                   *  +0.5*(b1*uuj+b2*vvj+b3*wwj)+b5*uvj+b6*vwj+b7*uwj
371                   *  +0.5*(f1*uuk+f2*vvk+f3*wwk)
372                   *  +f5*v1*uk+f6*w1*uk+f7*u1*vk+f8*w1*vk+f9*u1*wk+f10*v1*wk
373                   * )
374                   *  +ck*vol22*(d4*tti+b4*ttj+f4*ttk)
375                   *  -2./3.*r1*tk1*(u1*sjx+v1*sjy+w1*sjz)*vol22
376          1          enddo
377          3       20 continue
378                     do 21 j=2,jl
379                     do i=2,il
380                     do ir=1,4
381        465          dw(ir+1,i,j,k)=dw(ir+1,i,j,k)-(fvj(ir,i,j)-fvj(ir,i,j-1))
382                     enddo
383                     enddo
384                  21 continue
385                 200 continue
```

```
398                     do 30 k=1,kl
399                     kp=k+1
400          5          do j=2,jl
401                     jp=j+1
402                     jm=j-1
403                     do i=2,il
404                     ip=i+1
405                     im=i-1
406         59          vol2=vol(i,j,k)+vol(i,j,kp)

546        435          fvk(1,i,j,k)=dn*(e1*ui+e5*vi+e6*wi+f1*uj+f5*vj+f6*wj
547                   *        +c1*uk+c5*vk+c7*wk) -2./3.*r1*tk1*skx*vol22
548        273          fvk(2,i,j,k)=dn*(e7*ui+e2*vi+e8*wi+f7*uj+f2*vj+f8*wj
549                   *          +c5*uk+c2*vk+c6*wk) -2./3.*r1*tk1*sky*vol22
550        176          fvk(3,i,j,k)=dn*(e9*ui+e10*vi+e3*wi+f9*uj+f10*vj+f3*wj
551                   *          +c7*uk+c6*vk+c3*wk) -2./3.*r1*tk1*skz*vol22
552        945          fvk(4,i,j,k)=dn*(0.5*(e1*uui+e2*vvi+e3*wwi)+e5*u1*vi
553                   *    +e6*u1*wi+e7*v1*ui+e8*v1*wi+e9*w1*ui+e10*w1*vi
554                * +0.5*(f1*uuj+f2*vvj+f3*wwj)+f5*u1*vj+f6*u1*wj+f7*v1*uj+f8*v1*wj
555                   *  +f9*w1*uj+f10*w1*vj  +0.5*(c1*uuk+c2*vvk+c3*wwk)
556                   *  +c5*uvk+c6*vwk+c7*uwk)
557                   *  +ck*vol22*(e4*tti+f4*ttj+c4*ttk)
558                   *       -2./3.*r1*tk1*(u1*skx+v1*sky+w1*skz)*vol22
```

```
559                            enddo
560         3                  enddo
561                         30 continue
562                            do 31 k=2,kl
563                            do j=2,jl
564                            do i=2,il
565                            do ir=1,4
566       616                  dw(ir+1,i,j,k)=dw(ir+1,i,j,k)-(fvk(ir,i,j,k)-fvk(ir,i,j,k-1))
567                            enddo
568                            enddo
569                            enddo
570                         31 continue
571                        300 continue
```

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

② Do-loop 순서를 바꾸기 위해 몇몇 array들에 대해서 dimension을 맞게 변경하여 사용하였다.

5. vsfluxz2 루틴, vsfluxz3 루틴

vsfluxz2 루틴은 vsflux 루틴과 변수이름 정도만 차이날뿐 거의 비슷한 루틴이기 때문에 vsflux 루틴을 최적화 한것과 비슷하게 수정하였으며, 상대적으로 do-loop의 크기가 작은 vsfluxz3 루틴은 반복 계산되는 visc array 부분만 수정했다.

6. bound 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
 3                             do 100 j=2,jl
 4                             do 100 k=2,kl
 5                             do ir=1,5
 6         76                  w(ir,i2,j,k)=w(ir,il,j,k)
 7                             enddo
 8          1                  ur(i2,j,k)=ur(il,j,k)
 9                             vr(i2,j,k)=vr(il,j,k)
10          1                  wr(i2,j,k)=wr(il,j,k)
11         21                  p(i2,j,k)=p(il,j,k)
12          1                  t(i2,j,k)=p(i2,j,k)/w(1,i2,j,k)
13                         100 continue
```

```
15                             do 300 i=1,i2
16                             do 300 j=2,jl
```

```
17                       do ir=1,5
18          21            w(ir,i,j,k2)=w(ir,i,j,kl)
19           3            w(ir,i,j,1)=w(ir,i,j,2)
20                       enddo
21           1            ur(i,j,k2)=ur(i,j,kl)
22                        vr(i,j,k2)=vr(i,j,kl)
23           2            wr(i,j,k2)=wr(i,j,kl)
24           8            t(i,j,k2)=t(i,j,kl)
25           1            p(i,j,k2)=p(i,j,kl)
26           8            ur(i,j,1)=ur(i,j,2)
27           2            vr(i,j,1)=vr(i,j,2)
28           4            wr(i,j,1)=wr(i,j,2)
29           5            t(i,j,1)=t(i,j,2)
30           6            p(i,j,1)=p(i,j,2)
31                   300 continue
```

```
34                       do 200 i=1,i2
35                       do 200 k=1,k2
36                 !
37                       do ir=1,5
38          18            w(ir,i,j2,k)=w(ir,i,jl,k)
39          26            w(ir,i,1,k)=w(ir,i,2,k)
40                       enddo
41           7            ur(i,j2,k)=ur(i,jl,k)
42          11            vr(i,j2,k)=vr(i,jl,k)
43           4            wr(i,j2,k)=wr(i,jl,k)
44          14            t(i,j2,k)=t(i,jl,k)
45          10            p(i,j2,k)=p(i,jl,k)
46                        ur(i,1,k)=ur(i,2,k)
47                        vr(i,1,k)=vr(i,2,k)
48          13            wr(i,1,k)=wr(i,2,k)
49                        t(i,1,k)=t(i,2,k)
50          15            p(i,1,k)=p(i,2,k)
51                   200 continue
```

<최적화 코드>

```
 3                       do 100 k=2,kl
 4                       do 100 j=2,jl
 5                       do ir=1,5
 6          78            w(ir,i2,j,k)=w(ir,il,j,k)
 7                       enddo
 8                        ur(i2,j,k)=ur(il,j,k)
 9                        vr(i2,j,k)=vr(il,j,k)
10                        wr(i2,j,k)=wr(il,j,k)
11          14            p(i2,j,k)=p(il,j,k)
12          15            t(i2,j,k)=p(i2,j,k)/w(1,i2,j,k)
13                   100 continue
```

```
15                       do 300 j=2,jl
```

```
16                       do 300 i=1,i2
17                       do ir=1,5
18          2            w(ir,i,j,k2)=w(ir,i,j,kl)
19          5            w(ir,i,j,1)=w(ir,i,j,2)
20                       enddo
21          2            ur(i,j,k2)=ur(i,j,kl)
22          1            vr(i,j,k2)=vr(i,j,kl)
23          3            wr(i,j,k2)=wr(i,j,kl)
24                       t(i,j,k2)=t(i,j,kl)
25                       p(i,j,k2)=p(i,j,kl)
26                       ur(i,j,1)=ur(i,j,2)
27          5            vr(i,j,1)=vr(i,j,2)
28                       wr(i,j,1)=wr(i,j,2)
29          2            t(i,j,1)=t(i,j,2)
30          4            p(i,j,1)=p(i,j,2)
31                   300 continue
```

```
34                       do 200 k=1,k2
35                       do 200 i=1,i2
36                 !
37                       do ir=1,5
38          9            w(ir,i,j2,k)=w(ir,i,jl,k)
39         11            w(ir,i,1,k)=w(ir,i,2,k)
40                       enddo
41          3            ur(i,j2,k)=ur(i,jl,k)
42          1            vr(i,j2,k)=vr(i,jl,k)
43          2            wr(i,j2,k)=wr(i,jl,k)
44                       t(i,j2,k)=t(i,jl,k)
45                       p(i,j2,k)=p(i,jl,k)
46          6            ur(i,1,k)=ur(i,2,k)
47          1            vr(i,1,k)=vr(i,2,k)
48          9            wr(i,1,k)=wr(i,2,k)
49          2            t(i,1,k)=t(i,2,k)
50                       p(i,1,k)=p(i,2,k)
51                   200 continue
```

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

7. turbkom 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임, do-loop Merging

<Original 코드>

```
18                       do 600 i=2,il
19          1            do 600 j=2,jl
20                       do 600 k=2,kl
21       5374            wtk(i,j,k)=wtkn(i,j,k)
22                      *          +rungefactor*tw(1,i,j,k)/vol(i,j,k)
23       2062            wtom(i,j,k)=wtomn(i,j,k)
```

```
24                          *              +rungefactor*tw(2,i,j,k)/vol(i,j,k)
25          11          600 continue
26                          do 601 i=2,il
27                          do 601 j=2,jl
28                          do 601 k=2,kl
29        4056              tk(i,j,k)=wtk(i,j,k)/w(1,i,j,k)
30        2055              tom(i,j,k)=wtom(i,j,k)/w(1,i,j,k)
31         573              if(tom(i,j,k).lt.tomfree) then
32                          tom(i,j,k)=tomfree
33         332              wtom(i,j,k)=w(1,I,J,K)*tom(i,j,k)
34                          endif
35          11          601 continue
36
37                          do 611 i=2,il
38           1              do 611 j=2,jl
39                          do 611 k=2,kl
40        2533              emu(i,j,k)=tk(i,j,k)/tom(i,j,k)
41         828              if(emu(i,j,k).gt.emumax) then
42                          emu(i,j,k)=emumax
43          25              tk(i,j,k)=emu(i,j,k)*tom(i,j,k)
44                          wtk(i,j,k)=tk(i,j,k)*w(1,i,j,k)
45                          endif
46           9          611 continue
47
48                           tkm=0
49                           tem=0
50                          do 410 i=2,il
51           1                 do 410 j=2,jl
52                                do 410 k=2,kl
53        1422              if(tk(i,j,k).gt.tkm) then
54                          tkm=tk(i,j,k)
55                          ikm=i
56                          jkm=j
57                          kkm=k
58                          endif
59        1172              if(tom(i,j,k).gt.tem) then
60                          tem=tom(i,j,k)
61                          iem=i
62                          jem=j
63                          kem=k
64                          endif
65
66           3          410 continue
```

<최적화 코드>

```
18                           tkm=0
19                           tem=0
20                          do 600 k=2,kl
21          16              do 600 j=2,jl
22                          do 600 i=2,il
23         183              wtk(i,j,k)=wtkn(i,j,k)
24                          *              +rungefactor*tw(1,i,j,k)/vol(i,j,k)
25         315              wtom(i,j,k)=wtomn(i,j,k)
26                          *              +rungefactor*tw(2,i,j,k)/vol(i,j,k)

31          16              tk(i,j,k)=wtk(i,j,k)/w(1,i,j,k)
```

```
32          154              tom(i,j,k)=wtom(i,j,k)/w(1,i,j,k)
33          224              if(tom(i,j,k).lt.tomfree) then
34            4              tom(i,j,k)=tomfree
35                           wtom(i,j,k)=w(1,I,J,K)*tom(i,j,k)
36                           endif

42            8              emu(i,j,k)=tk(i,j,k)/tom(i,j,k)
43          354              if(emu(i,j,k).gt.emumax) then
44                           emu(i,j,k)=emumax
45                           tk(i,j,k)=emu(i,j,k)*tom(i,j,k)
46                           wtk(i,j,k)=tk(i,j,k)*w(1,i,j,k)
47                           endif

53            9              if(tk(i,j,k).gt.tkm) then
54                           tkm=tk(i,j,k)
55                           ikm=i
56                           jkm=j
57                           kkm=k
58                           endif
59           34              if(tom(i,j,k).gt.tem) then
60                           tem=tom(i,j,k)
61                           iem=i
62                           jem=j
63                           kem=k
64                           endif
65
66                      600 continue
```

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

② Original 코드에서 I=2,il, j=2,jl, k=2,kl 과 같이 동일한 크기의 각 do-loop를 하나의 loop로 합쳐서 파이프라이닝이 되도록 하면서 do-loop 수행에 의한 overhead를 줄였다.

8. turbkomzone2 루틴

turbkomzone2 루틴은 turbkom 루틴과 거의 같은 식으로 do-loop 순서수정으로 cache miss rate를 줄여 최적화 하였다. 실제 turbkomzone2 루틴은 turbkom 루틴과 변수이름 정도만 차이날뿐 거의 비슷하면서 do-loop의 크기가 작은 루틴이다.

9. turhs 루틴

A. 반복 계산되는 visc array를 한번만 계산

<Original 코드>

```
98                           do 100 k=2,kl
99                           kp=k+1
```

```
100                        km=k-1
101                        do 100 j=2,jl
102                        jp=j+1
103                        jm=j-1
104                   !
105          1             do   i = 1,i2
106         29             visc(i) = (t(i,j,k)/tfree)**vex/res
107                        enddo

127        288             vise=((visc(i)+emu(i,j,k)*sigma_tko)*fac1
128                       *      +(visc(ip)+emu(ip,j,k)*sigma_tko)*fac2)*vol2/2.
129         18             visk=((visc(i)+emu(i,j,k)*sigmastar_tko)*fac1
130                       *      +(visc(ip)+emu(ip,j,k)*sigmastar_tko)*fac2)*vol2/2.

320                        do 200 k=2,kl
321                        kp=k+1
322                        km=k-1
323                        do 200 i=2,il
324                        ip=i+1
325                        im=i-1
326                   !
327                        do j = 1,j2
328        320             visc(j) = (t(i,j,k)/tfree)**vex/res
329                        enddo

352        413             vise=((visc(j)+emu(i,j,k)*sigma_tko)*fac1
353                       *      +(visc(jp)+emu(i,jp,k)*sigma_tko)*fac2)*vol2/2.
354        304             visk=((visc(j)+emu(i,j,k)*sigmastar_tko)*fac1
355                       *      +(visc(jp)+emu(i,jp,k)*sigmastar_tko)*fac2)*vol2/2.

537                        do 300 i=2,il
538                        ip=i+1
539                        im=i-1
540          2             do 300 j=2,jl
541                        jp=j+1
542                        jm=j-1
543                   !
544          1             do k = 1,k2
545       4107             visc(k) = (t(i,j,k)/tfree)**vex/res
546                        enddo

569       4178             vise=((visc(k)+emu(i,j,k)*sigma_tko)*fac1
570                       *      +(visc(kp)+emu(i,j,kp)*sigma_tko)*fac2)*vol2/2.
571        970             visk=((visc(k)+emu(i,j,k)*sigmastar_tko)*fac1
572                       *      +(visc(kp)+emu(i,j,kp)*sigmastar_tko)*fac2)*vol2/2.
```

<최적화 코드>

```
102                        do   k = 1,k2
103                        do   j = 1,j2
104                        do   i = 1,i2
105        315             visc(i,j,k) = (t(i,j,k)/tfree)**vex/res
106                        enddo
107                        enddo
108                        enddo

135        424             vise=((visc(i,j,k)+emu(i,j,k)*sigma_tko)*fac1
```

```
136                              *       +(visc(ip,j,k)+emu(ip,j,k)*sigma_tko)*fac2)*vol2/2.
137           13                  visk=((visc(i,j,k)+emu(i,j,k)*sigmastar_tko)*fac1
138                              *       +(visc(ip,j,k)+emu(ip,j,k)*sigmastar_tko)*fac2)*vol2/2.

356          548                  vise=((visc(i,j,k)+emu(i,j,k)*sigma_tko)*fac1
357                              *       +(visc(i,jp,k)+emu(i,jp,k)*sigma_tko)*fac2)*vol2/2.
358           66                  visk=((visc(i,j,k)+emu(i,j,k)*sigmastar_tko)*fac1
359                              *       +(visc(i,jp,k)+emu(i,jp,k)*sigmastar_tko)*fac2)*vol2/2.

586          171                  vise=((visc(i,j,k)+emu(i,j,k)*sigma_tko)*fac1
587                              *       +(visc(i,j,kp)+emu(i,j,kp)*sigma_tko)*fac2)*vol2/2.
588           77                  visk=((visc(i,j,k)+emu(i,j,k)*sigmastar_tko)*fac1
589                              *       +(visc(i,j,kp)+emu(i,j,kp)*sigmastar_tko)*fac2)*vol2/2.
```

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

B. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
320                               do 200 k=2,kl
321                               kp=k+1
322                               km=k-1
323                               do 200 i=2,il
324                               ip=i+1
325                               im=i-1

405            2                  do 25 j=1,j2
406                         !
407          930                  uu=ur(i,j,k)*etx(i,j,k)+vr(i,j,k)*ety(i,j,k)
408                              *    +wr(i,j,k)*etz(i,j,k)
409         2016                    cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))*sqrt(etx(i,j,k)**2+
410                              *       ety(i,j,k)**2+etz(i,j,k)**2)

434           13               25 continue

533            6              200 continue


537                               do 300 i=2,il
538                               ip=i+1
539                               im=i-1
540            2                  do 300 j=2,jl
541                               jp=j+1
542                               jm=j-1

548            7                  do 30 k=1,kl
549                               kp=k+1
550                         !
551           76                  vol2=vol(i,j,k)+vol(i,j,kp)
552            1                  fac1=vol(i,j,kp)/vol2
553                               fac2=vol(i,j,k)/vol2
```

```
554     1862        six=fac1*xix(i,j,k)+fac2*xix(i,j,kp)
555       37        siy=fac1*xiy(i,j,k)+fac2*xiy(i,j,kp)
556       47        siz=fac1*xiz(i,j,k)+fac2*xiz(i,j,kp)
557     5425        sjx=fac1*etx(i,j,k)+fac2*etx(i,j,kp)
558        9        sjy=fac1*ety(i,j,k)+fac2*ety(i,j,kp)
559     1886        sjz=fac1*etz(i,j,k)+fac2*etz(i,j,kp)
560       75        skx=fac1*ztx(i,j,k)+fac2*ztx(i,j,kp)
561       56        sky=fac1*zty(i,j,k)+fac2*zty(i,j,kp)
562       53        skz=fac1*ztz(i,j,k)+fac2*ztz(i,j,kp)

620              30 continue

622                 do 35 k=1,k2
623     6654          uu=ur(i,j,k)*ztx(i,j,k)+vr(i,j,k)*zty(i,j,k)
624               *     +wr(i,j,k)*ztz(i,j,k)
625     9390          cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))*sqrt(ztx(i,j,k)**2+
626               *         zty(i,j,k)**2+ztz(i,j,k)**2)

643                 tc(1,k)=vol(i,j,k)*w(1,i,j,k)*tk(i,j,k)*ctp
644       27        tc(2,k)=vol(i,j,k)*w(1,i,j,k)*tom(i,j,k)*ctp
645       42        tcm(1,k)=vol(i,j,k)*w(1,i,j,k)*tk(i,j,k)*ctm
646       63        tcm(2,k)=vol(i,j,k)*w(1,i,j,k)*tom(i,j,k)*ctm

649      180     35 continue

650                   do 31 k=2,kl
651                   l=k-1
652       21          if((k.eq.2) .or. (k.eq.kl)) then
653                   do ir=1,2
654       58            tw(ir,i,j,k)=tw(ir,i,j,k)+(tc(ir,k)-tc(ir,k-1))
655                   enddo
656                   else
657                     do ir=1,2

688     4424          tw(ir,i,j,k)=tw(ir,i,j,k)+(1.+0.5*p1)*(tc(ir,k)-tc(ir,l))
689               * -0.5*p2*(tc(ir,l)-tc(ir,l-1))
690                   enddo
691                   endif
692        5      31  continue

748             300 continue
```

<최적화 코드>

```
328                 do 200 k=2,kl
329                 kp=k+1
330                 km=k-1

410                 do 25 j=1,j2
411                 do i=2,il
412            !
413      471         uu=ur(i,j,k)*etx(i,j,k)+vr(i,j,k)*ety(i,j,k)
414               *     +wr(i,j,k)*etz(i,j,k)
415      154          cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))*sqrt(etx(i,j,k)**2+
416               *         ety(i,j,k)**2+etz(i,j,k)**2)

440       21         enddo
```

```
441                        25 continue

546                       200 continue

559                            do 30 k=1,kl
560                            kp=k+1
561              5             do i=2,il
562                            ip=i+1
563                            im=i-1
564                            do j=2,jl
565                            jp=j+1
566                            jm=j-1
567                     !
568             14             vol2=vol(i,j,k)+vol(i,j,kp)
569                            fac1=vol(i,j,kp)/vol2
570                            fac2=vol(i,j,k)/vol2
571            544             six=fac1*xix(i,j,k)+fac2*xix(i,j,kp)
572                            siy=fac1*xiy(i,j,k)+fac2*xiy(i,j,kp)
573             86             siz=fac1*xiz(i,j,k)+fac2*xiz(i,j,kp)
574            408             sjx=fac1*etx(i,j,k)+fac2*etx(i,j,kp)
575            814             sjy=fac1*ety(i,j,k)+fac2*ety(i,j,kp)
576            656             sjz=fac1*etz(i,j,k)+fac2*etz(i,j,kp)
577                            skx=fac1*ztx(i,j,k)+fac2*ztx(i,j,kp)
578            900             sky=fac1*zty(i,j,k)+fac2*zty(i,j,kp)
579             11             skz=fac1*ztz(i,j,k)+fac2*ztz(i,j,kp)

636                            enddo
637             10             enddo
638
639                        30 continue

641                            do 35 k=1,k2
642                            do j=2,jl
643                            do i=2,il
644            528               uu=ur(i,j,k)*ztx(i,j,k)+vr(i,j,k)*zty(i,j,k)
645                          *    +wr(i,j,k)*ztz(i,j,k)
646            465               cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))*sqrt(ztx(i,j,k)**2+
647                          *        zty(i,j,k)**2+ztz(i,j,k)**2)

664            169             tck(1,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tk(i,j,k)*ctp
665            155             tck(2,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tom(i,j,k)*ctp
666             73             tcmk(1,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tk(i,j,k)*ctm
667              2             tcmk(2,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tom(i,j,k)*ctm

670                            enddo
671              1             enddo
672                        35 continue

673                              do 31 k=2,kl
674                              l=k-1
675                            do j=2,jl
676                            do i=2,il
677                              if((k.eq.2) .or. (k.eq.kl)) then
678                              do ir=1,2
679             13                 tw(ir,i,j,k)=tw(ir,i,j,k)+(tck(ir,i,j,k)-tck(ir,i,j,k-1))
680                              enddo
681                              else
682                                do ir=1,2

713            164               tw(ir,i,j,k)=tw(ir,i,j,k)
```

```
714                         * +(1.+0.5*p1)*(tck(ir,i,j,k)-tck(ir,i,j,l))
715                         * -0.5*p2*(tck(ir,i,j,l)-tck(ir,i,j,l-1))
716                           enddo
717                           endif
718                          enddo
719                          enddo
720                      31   continue
```

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

② Loop 순서를 변경하기 위해 original 코드의 2차원 tc, tcm array 대신 Serial 최적화 코드에서는 4차원 tck, tcmk array로 선언하여 사용하였다.

③ Serial 최적화 코드에서 559번째 줄 근처의 k->i->j 순서로 진행되는 do-loop를 k->j->i 순서로 수정할 경우 좀더 성능이 나아질 수 있다.(실제 MPI 코드에서는 k->j->I 순서로 수정되었다.)

10. turhszone2 루틴, turhszone3 루틴

turhszone2 루틴은 turhs 루틴과 변수이름 정도만 차이날뿐 거의 비슷한 루틴이기 때문에 turhs 루틴을 최적화 한것과 비슷하게 수정하였으며, 상대적으로 do-loop의 크기가 작은 turhszone3 루틴은 아래에서 visc array만을 수정했다.

11. boundturhs 루틴

A. do-loop 순서 변경으로 Cache miss rate를 줄임

<Original 코드>

```
 4                    do 100 j=2,jl
 5                    do 100 k=2,kl
 6        12          wtk(i2,j,k)=wtk(il,j,k)
 7        26          wtom(i2,j,k)=wtom(il,j,k)
 8        18          tk(i2,j,k)=tk(il,j,k)
 9        16          tom(i2,j,k)=tom(il,j,k)
10        15          emu(i2,j,k)=emu(il,j,k)
11                100 continue
12
13                    do 300 i=1,i2
14                    do 300 j=2,jl
15         1          wtk(i,j,k2)=wtk(i,j,kl)
16         2          wtom(i,j,k2)=wtom(i,j,kl)
17         2          tk(i,j,k2)=tk(i,j,kl)
18         1          tom(i,j,k2)=tom(i,j,kl)
19                    emu(i,j,k2)=emu(i,j,kl)
20         4          wtk(i,j,1)=wtk(i,j,2)
21         3          wtom(i,j,1)=wtom(i,j,2)
```

```
22      4        tk(i,j,1)=tk(i,j,2)
23      7        tom(i,j,1)=tom(i,j,2)
24      7        emu(i,j,1)=emu(i,j,2)
25           300 continue
26
27               do 200 i=1,i2
28               do 200 k=1,k2
29     14        tk(i,j2,k)=tkfree
30      2        tom(i,j2,k)=tomfree
31     21        emu(i,j2,k)=emufree
32     10        wtk(i,j2,k)=tk(i,j2,k)*w(1,i,j2,k)
33               wtom(i,j2,k)=tom(i,j2,k)*w(1,i,j2,k)
34               wtk(i,1,k)=wtk(i,2,k)
35               wtom(i,1,k)=wtom(i,2,k)
36     12        tk(i,1,k)=tk(i,2,k)
37      4        tom(i,1,k)=tom(i,2,k)
38               emu(i,1,k)=emu(i,2,k)
39           200 continue
```

<최적화 코드>

```
 4               do 100 k=2,kl
 5               do 100 j=2,jl
 6               wtk(i2,j,k)=wtk(il,j,k)
 7               wtom(i2,j,k)=wtom(il,j,k)
 8     21        tk(i2,j,k)=tk(il,j,k)
 9     26        tom(i2,j,k)=tom(il,j,k)
10     10        emu(i2,j,k)=emu(il,j,k)
11      1    100 continue
12
13               do 300 j=2,jl
14               do 300 i=1,i2
15      2        wtk(i,j,k2)=wtk(i,j,kl)
16      1        wtom(i,j,k2)=wtom(i,j,kl)
17               tk(i,j,k2)=tk(i,j,kl)
18               tom(i,j,k2)=tom(i,j,kl)
19               emu(i,j,k2)=emu(i,j,kl)
20      2        wtk(i,j,1)=wtk(i,j,2)
21      2        wtom(i,j,1)=wtom(i,j,2)
22      3        tk(i,j,1)=tk(i,j,2)
23               tom(i,j,1)=tom(i,j,2)
24      1        emu(i,j,1)=emu(i,j,2)
25           300 continue
26
27               do 200 k=1,k2
28               do 200 i=1,i2
29      1        tk(i,j2,k)=tkfree
30               tom(i,j2,k)=tomfree
31               emu(i,j2,k)=emufree
32      9        wtk(i,j2,k)=tk(i,j2,k)*w(1,i,j2,k)
33               wtom(i,j2,k)=tom(i,j2,k)*w(1,i,j2,k)
34      6        wtk(i,1,k)=wtk(i,2,k)
35               wtom(i,1,k)=wtom(i,2,k)
36               tk(i,1,k)=tk(i,2,k)
37      4        tom(i,1,k)=tom(i,2,k)
38      1        emu(i,1,k)=emu(i,2,k)
39           200 continue
```

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

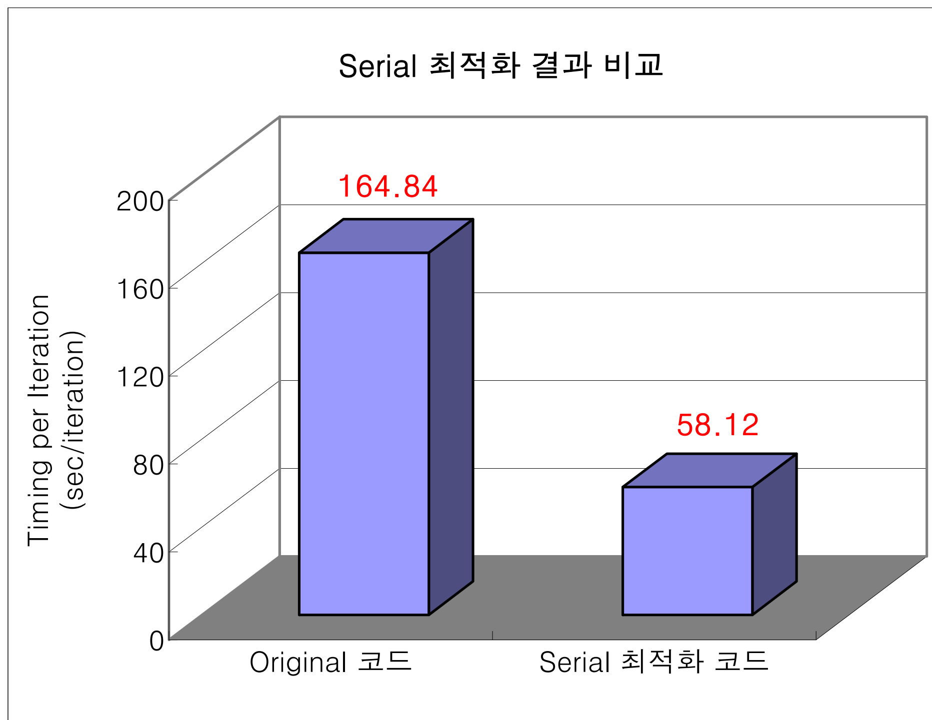
7. Serial 최적화 결과

전반적으로 Original 코드와 Serial 최적화 코드를 비교 분석해 봤을 때 크게 수정된 부분은 없으며 연속적인 메모리 access를 위하여 do-loop의 순서를 변경하여 높은 성능향상효과를 보았다.

이렇게 최적화하여 수행한 결과를 비교해 보면 다음과 같다.

참고로 아래 결과는 동일한 최적화 컴파일 옵션(-O3 -qarch=pwr4) 및 library(-lmass)를 사용했을 때의 결과이다.

	p690 1.7GHz 1CPU
Original 코드	164.84 sec/iteration
Serial 최적화 코드	58.12 sec/iteration
Speedup	2.84



8. MPI 병렬화 코드 비교 및 분석

MPI 병렬화를 위해서 다음의 코드들을 수정하였다.

1. param_mpi.inc

A. MPI 병렬 코드를 위한 기본적인 변수들을 선언

<MPI 병렬화 코드>

```
      parameter(maxmpi=256,maxmpi1=maxmpi-1)
      common/mpi_variables/nproc,id,nprocj,nprock,idj,idk,
     & jse(2,0:maxmpi1),kse(2,0:maxmpi1),
     & jsez2(2,0:maxmpi1),ksez2(2,0:maxmpi1),
     & jsez3(2,0:maxmpi1),ksez3(2,0:maxmpi1)
      data id0/0/
```

① 여러가지 MPI subroutine들을 위한 기본적인 변수들 및 실제 병렬화되는 부분의 영역을 나타내는 변수들을 선언해 두었다.

2. decomp_2d 루틴

A. 병렬로 계산되어질 **array**들을 효율적으로 분할하는 루틴

<MPI 병렬화 코드>

```
1           subroutine decomp_2d
2           include 'vcom_cav_rk.inc'
3           INCLUDE 'param_mpi.inc'
4           integer tab(maxmpi)
5     c
6           n=nproc
7           nn=0
8           do i=1,n
9              if(mod(n,i) .eq. 0)   then
10                nn=nn+1
11                tab(nn)=i
12             endif
13          enddo
14    c
15          i0=1
16          n1=sqrt(float(n))
17          if(n1**2.ne.n) n1=n1+1
18          do i=1,nn
19             if(n1.le.tab(i)) then
20                i0=i
21                goto 100
22             endif
23          enddo
24    100   continue
```

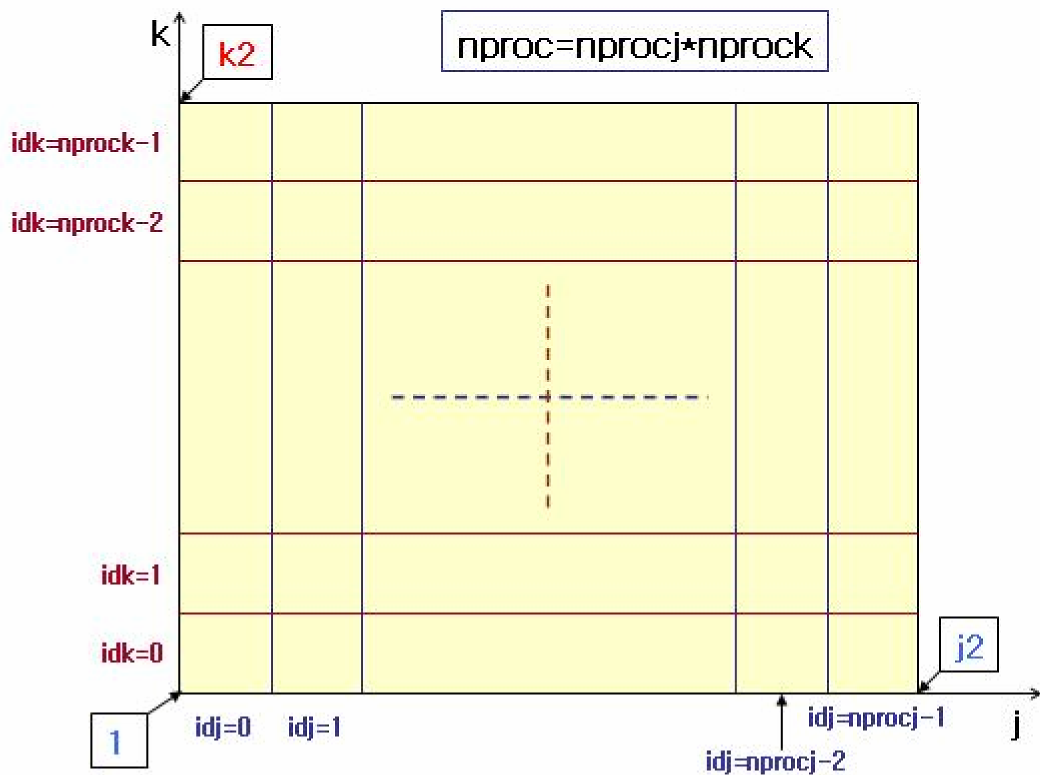
```
25          do i=i0,nn
26          do j=i0,1,-1
27             ii=tab(i)*tab(j)
28             if(ii .eq. n) then
29                nprock=tab(i)
30                nprocj=tab(j)
31                goto 101
32             elseif(ii .lt. n) then
33                goto 102
34             endif
35          enddo
36    102    continue
37          enddo
38    101    continue
39          idk=id/nprocj
40          idj=mod(id,nprocj)

46          call mpicut(kse,1,k2,nprock)
47          call mpicut(jse,1,j2,nprocj)
48          call mpicut(ksez2,1,k2z2,nprock)
49          call mpicut(jsez2,1,j2z2,nprocj)
50          call mpicut(ksez3,1,k2z3,nprock)
51          call mpicut(jsez3,1,j2z3,nprocj)
```

```
55          subroutine mpicut(ks,kstart,kend,nthds)
56          integer ks(2,*)
57    c
58          nn=kend-kstart+1
59          nblk=nn/nthds
60          ndelta=mod(nn,nthds)
61          if(ndelta.gt.0) then
62             k=1
63             ks(1,1)=1
64             ks(2,1)=ks(1,1)+nblk
65          do k=2,ndelta
66             ks(1,k)=ks(1,k-1)+nblk+1
67             ks(2,k)=ks(1,k)+nblk
68          enddo
69             k=ndelta+1
70             ks(1,k)=ks(1,k-1)+nblk+1
71             ks(2,k)=ks(1,k)+nblk-1
72          do k=ndelta+2,nthds
73             ks(1,k)=ks(1,k-1)+nblk
74             ks(2,k)=ks(1,k)+nblk-1
75          enddo
76          else
77             ks(1,1)=1
78             ks(2,1)=ks(1,1)+nblk-1
79          do k=2,nthds
80             ks(1,k)=ks(1,k-1)+nblk
81             ks(2,k)=ks(1,k)+nblk-1
82          enddo
83          endif
84          do k=1,nthds
85             ks(1,k)=ks(1,k)+kstart-1
86             ks(2,k)=ks(2,k)+kstart-1
87          enddo
```

① 이 루틴은 병렬로 나누어 계산하게 될 kse, jse, ksez2, jsez2, ksez3, jsez3 array 들을 MPI task 수에 따라 효율적으로 나누어 주는 역할을 한다.

② 즉 MPI task수가 정해 지면 최대한 비슷한 갯수로 j방향과 k방향으로 영역 수를 계산하고 해당 영역의 index를 정한다. 그런 후 mpicut 루틴을 call하여 최대한 load balancing이 되도록 jse, jsez2, jsez3 array에 대해서는 j방향 영역 갯수로 나누어 주고, kse, ksez2, ksez3 array에 대해서는 k방향 영역 개수로 나누어 준다.

③ 전체적인 영역분할 구조는 다음 그림과 같다.



④ 임의의 영역에서 해당 rank는 다음과 같다.

rank = idj + nprocj * idk

⑤ 다음은 mpi task 수를 12로 하여 decomp_2d 루틴을 수행했을 때 나오는 결과 예이다.

```
0: nproc = 12
0: id= 0 >> nprocj= 3 nprock= 4 idj= 0 idk= 0
1: id= 1 >> nprocj= 3 nprock= 4 idj= 1 idk= 0
2: id= 2 >> nprocj= 3 nprock= 4 idj= 2 idk= 0
3: id= 3 >> nprocj= 3 nprock= 4 idj= 0 idk= 1
4: id= 4 >> nprocj= 3 nprock= 4 idj= 1 idk= 1
5: id= 5 >> nprocj= 3 nprock= 4 idj= 2 idk= 1
6: id= 6 >> nprocj= 3 nprock= 4 idj= 0 idk= 2
```

```
  7: id= 7 >> nprocj= 3 nprock= 4 idj= 1 idk= 2
  8: id= 8 >> nprocj= 3 nprock= 4 idj= 2 idk= 2
  9: id= 9 >> nprocj= 3 nprock= 4 idj= 0 idk= 3
 10: id= 10 >> nprocj= 3 nprock= 4 idj= 1 idk= 3
 11: id= 11 >> nprocj= 3 nprock= 4 idj= 2 idk= 3

  0: ===============================
  0:      kse(1, 0) =    1      kse(2, 0) =   31
  0:      kse(1, 1) =   32      kse(2, 1) =   62
  0:      kse(1, 2) =   63      kse(2, 2) =   92
  0:      kse(1, 3) =   93      kse(2, 3) = 122
  0:   ksez2(1, 0) =    1   ksez2(2, 0) =   16
  0:   ksez2(1, 1) =   17   ksez2(2, 1) =   32
  0:   ksez2(1, 2) =   33   ksez2(2, 2) =   47
  0:   ksez2(1, 3) =   48   ksez2(2, 3) =   62
  0:   ksez3(1, 0) =    1   ksez3(2, 0) =   16
  0:   ksez3(1, 1) =   17   ksez3(2, 1) =   32
  0:   ksez3(1, 2) =   33   ksez3(2, 2) =   47
  0:   ksez3(1, 3) =   48   ksez3(2, 3) =   62
  0: ===============================
  0:      jse(1, 0) =    1      jse(2, 0) =   28
  0:      jse(1, 1) =   29      jse(2, 1) =   55
  0:      jse(1, 2) =   56      jse(2, 2) =   82
  0:   jsez2(1, 0) =    1   jsez2(2, 0) =   21
  0:   jsez2(1, 1) =   22   jsez2(2, 1) =   42
  0:   jsez2(1, 2) =   43   jsez2(2, 2) =   62
  0:   jsez3(1, 0) =    1   jsez3(2, 0) =   21
  0:   jsez3(1, 1) =   22   jsez3(2, 1) =   42
  0:   jsez3(1, 2) =   43   jsez3(2, 2) =   62
  0: ===============================
```

3. mpi_com_pm2 루틴

A. k 방향으로 이웃하는 rank와 경계 부근의 data 교환

<MPI 병렬화 코드>

```
179                        nj=je-js+1
180                        if(idk.ne.nprock-1) THEN
181                           idest1=idj+nprocj*(idk+1)
182                           l1=1
183                           do k=ke-1,ke
184                           do j=js,je
185                           do i=1,i2
186                           do ir=1,5
187            20                BUFFSJ1(l1)=w(ir,i,j,k)
188            27                l1=l1+1
189                           enddo
190                           enddo
191                           enddo
192                           enddo
193                        else
194                           idest1=MPI_PROC_NULL
195                        endif
196                        if(idk.ne.0) THEN
197                           idest2=idj+nprocj*(idk-1)
```

```
198                        l1=1
199                        do k=ks,ks+1
200                        do j=js,je
201                        do i=1,i2
202                        do ir=1,5
203        24                    BUFFSJ2(l1)=w(ir,i,j,k)
204        21                    l1=l1+1
205                        enddo
206                        enddo
207                        enddo
208                        enddo
209                     else
210                        idest2=MPI_PROC_NULL
211                     endif
212
213                     len=5*i2*nj*2
214                     itag=1
215                     CALL MPI_ISEND(BUFFSJ1,len,MPI_REAL8,idest1,itag,
216                    & MPI_COMM_WORLD,isend1,ierr)
217                     CALL MPI_ISEND(BUFFSJ2,len,MPI_REAL8,idest2,itag,
218                    & MPI_COMM_WORLD,isend2,ierr)
219
220                     CALL MPI_IRECV(BUFFRJ1,len,MPI_REAL8,idest2,itag,
221                    & MPI_COMM_WORLD,irecv1,ierr)
222                     CALL MPI_IRECV(BUFFRJ2,len,MPI_REAL8,idest1,itag,
223                    & MPI_COMM_WORLD,irecv2,ierr)
224
225                     CALL MPI_WAIT(isend1,istatus,ierr)
226                     CALL MPI_WAIT(isend2,istatus,ierr)
227                     CALL MPI_WAIT(irecv1,istatus,ierr)
228                     CALL MPI_WAIT(irecv2,istatus,ierr)
229
230                     if(idk.ne.0) THEN
231                        l1=1
232                        do k=ks-2,ks-1
233                        do j=js,je
234                        do i=1,i2
235                        do ir=1,5
236        40                    w(ir,i,j,k)=BUFFRJ1(l1)
237         8                    l1=l1+1
238                        enddo
239                        enddo
240         1              enddo
241                        enddo
242                     endif
243                     if(idk.ne.nprock-1) THEN
244                        l1=1
245                        do k=ke+1,ke+2
246                        do j=js,je
247                        do i=1,i2
248                        do ir=1,5
249        21                    w(ir,i,j,k)=BUFFRJ2(l1)
250        12                    l1=l1+1
251                        enddo
252                        enddo
253                        enddo
254                        enddo
255                     endif
```

① 각 rank에서 알고 있는 w array에 대해 먼저 k 방향으로 다음 rank(idj+nprocj*(idk+1))와 접하는 부분의 data를 BUFFSJ1에 저장한 후 다음 rank로 보내주면 다음 rank에서는 BUFFRJ1으로 받아 이미 알고 있는 영역의 w array에 추가적으로 저장한다.

② 또한 k 방향으로 이전 rank(idj+nprocj*(idk-1))와 접하는 부분의 data를 BUFFSJ2에 저장한 후 이전 rank로 보내주면 이전 rank에서는 BUFFRJ2로 받아 이미 알고 있는 영역의 w array에 추가적으로 저장한다.

③ 두개의 통신이 각각 전 rank에 걸쳐 네트워크 상에서 중첩되는 부분 없이 한방향으로 이루어지므로 통신 효율을 높일 수 있다. 즉 전체적으로 단 2번의 통신만으로 모든 통신이 이루어진다.

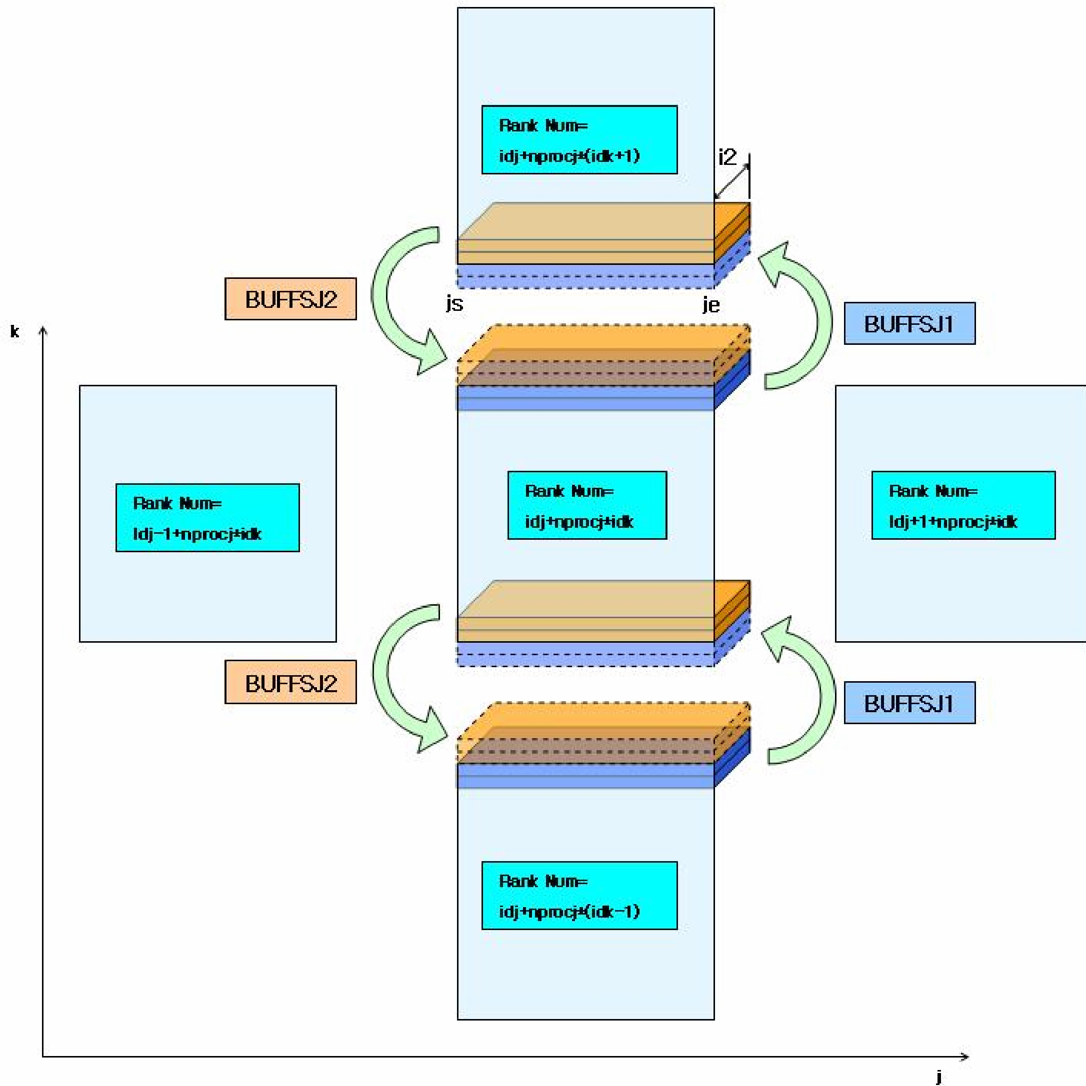


그림 I.2 K 방향으로의 데이터 상호 교환도

B. j 방향으로 이웃하는 rank와 경계 부근의 data 교환

<MPI 병렬화 코드>

```
257                      ks1=max(1,ks-2)
258                      ke1=min(k2,ke+2)
259                      nk=ke1-ks1+1
260                      if(idj.ne.nprocj-1) THEN
261                        idest1=idj+1+nprocj*idk
262                        l1=1
263                        do k=ks1,ke1
264                        do j=je-1,je
265                        do i=1,i2
266                        do ir=1,5
267          59                BUFFSK1(l1)=w(ir,i,j,k)
268          26                l1=l1+1
269                        enddo
270                        enddo
271                        enddo
272                        enddo
273                      else
274                        idest1=MPI_PROC_NULL
275                      endif
276                      if(idj.ne.0) THEN
277                        idest2=idj-1+nprocj*idk
278                        l1=1
279                        do k=ks1,ke1
280                        do j=js,js+1
281                        do i=1,i2
282                        do ir=1,5
283          61                BUFFSK2(l1)=w(ir,i,j,k)
284          23                l1=l1+1
285                        enddo
286                        enddo
287                        enddo
288                        enddo
289                      else
290                        idest2=MPI_PROC_NULL
291                      endif
292           1          len=5*i2*2*nk
293                      itag=1
294                      CALL MPI_ISEND(BUFFSK1,len,MPI_REAL8,idest1,itag,
295                     & MPI_COMM_WORLD,isend1,ierr)
296                      CALL MPI_ISEND(BUFFSK2,len,MPI_REAL8,idest2,itag,
297                     & MPI_COMM_WORLD,isend2,ierr)
298
299                      CALL MPI_IRECV(BUFFRK1,len,MPI_REAL8,idest2,itag,
300                     & MPI_COMM_WORLD,irecv1,ierr)
301                      CALL MPI_IRECV(BUFFRK2,len,MPI_REAL8,idest1,itag,
302                     & MPI_COMM_WORLD,irecv2,ierr)
303
304                      CALL MPI_WAIT(isend1,istatus,ierr)
305                      CALL MPI_WAIT(isend2,istatus,ierr)
306                      CALL MPI_WAIT(irecv1,istatus,ierr)
307                      CALL MPI_WAIT(irecv2,istatus,ierr)
308
```

```
309                          if(idj.ne.0) THEN
310                             I1=1
311                             do k=ks1,ke1
312                             do j=js-2,js-1
313                             do i=1,i2
314                             do ir=1,5
315            23                  w(ir,i,j,k)=BUFFRK1(I1)
316            49                  I1=I1+1
317                             enddo
318             6               enddo
319                             enddo
320                             enddo
321                          endif
322                          if(idj.ne.nprocj-1) THEN
323                             I1=1
324                             do k=ks1,ke1
325                             do j=je+1,je+2
326                             do i=1,i2
327                             do ir=1,5
328            36                  w(ir,i,j,k)=BUFFRK2(I1)
329            25                  I1=I1+1
330                             enddo
331            11               enddo
332                             enddo
333                             enddo
334
335                          endif
```

① 앞에서 설명된 k 방향으로의 data 교환과 마찬가지로 j방향으로 data 교환을 한다. 이때 k 방향으로 data 교환후 알게 된 이웃하는 rank와의 경계부분 data도 같이 전송하기 위해 k방향 길이를 ks1부터 ke1으로 늘려 주게 된다.

② 각 rank에서 알고 있는 w array에 대해 j 방향으로 다음 rank(idj+1+nprocj*idk)와 접하는 부분의 data를 BUFFSK1에 저장한 후 다음 rank로 보내주면 다음 rank에서는 BUFFRK1으로 받아 이미 알고 있는 영역의 w array에 추가적으로 저장한다.

③ 또한 j 방향으로 이전 rank(idj-1+nprocj*idk)와 접하는 부분의 data를 BUFFSK2에 저장한 후 이전 rank로 보내주면 이전 rank에서는 BUFFRK2로 받아 이미 알고 있는 영역의 w array에 추가적으로 저장한다.

④ 두개의 통신이 각각 전 rank에 걸쳐 네트워크 상에서 중첩되는 부분 없이 한방향으로 이루어지므로 통신 효율을 높일 수 있다. 즉 전체적으로 단 2번의 통신만으로 모든 통신이 이루어진다.

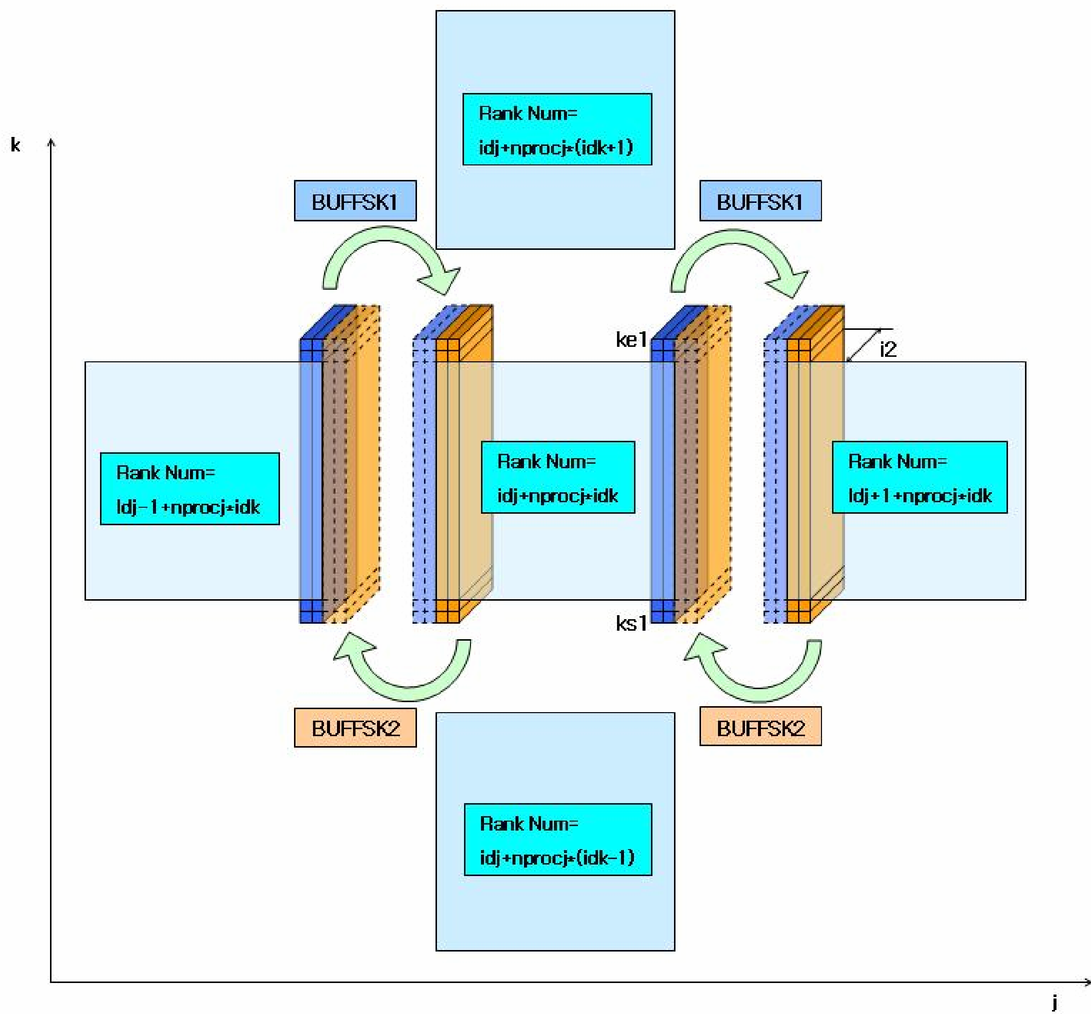


그림 I.3 J방향으로의 데이터 상호 교환도

4. mpi_com_pm2 루틴

<MPI 병렬화 코드>

```
353                       nj=je-js+1
  354                       if(idk.ne.nprock-1) THEN
  355                          idest1=idj+nprocj*(idk+1)
  356                          l1=1
  357                          do k=ke-1,ke
  358                          do j=js,je
  359                          do i=1,i2
  360                          do ir=1,2
  361           1                 BUFFSJ1(l1)=w(ir,i,j,k)
  362           6                 l1=l1+1
  363                          enddo
  364           5              enddo
  365                          enddo
  366                          enddo
```

```
367                       else
368                          idest1=MPI_PROC_NULL
369                       endif
370                       if(idk.ne.0) THEN
371                          idest2=idj+nprocj*(idk-1)
372                          l1=1
373                          do k=ks,ks+1
374                          do j=js,je
375                          do i=1,i2
376                          do ir=1,2
377           5                 BUFFSJ2(l1)=w(ir,i,j,k)
378          17                 l1=l1+1
379                          enddo
380           6              enddo
381                          enddo
382                          enddo
383                       else
384                          idest2=MPI_PROC_NULL
385                       endif
386
387                       len=2*i2*nj*2
388                       itag=1
389                       CALL MPI_ISEND(BUFFSJ1,len,MPI_REAL8,idest1,itag,
390                      & MPI_COMM_WORLD,isend1,ierr)
391                       CALL MPI_ISEND(BUFFSJ2,len,MPI_REAL8,idest2,itag,
392                      & MPI_COMM_WORLD,isend2,ierr)
393
394                       CALL MPI_IRECV(BUFFRJ1,len,MPI_REAL8,idest2,itag,
395                      & MPI_COMM_WORLD,irecv1,ierr)
396                       CALL MPI_IRECV(BUFFRJ2,len,MPI_REAL8,idest1,itag,
397                      & MPI_COMM_WORLD,irecv2,ierr)
398
399                       CALL MPI_WAIT(isend1,istatus,ierr)
400                       CALL MPI_WAIT(isend2,istatus,ierr)
401                       CALL MPI_WAIT(irecv1,istatus,ierr)
402                       CALL MPI_WAIT(irecv2,istatus,ierr)
403                       if(idk.ne.0) THEN
404                          l1=1
405                          do k=ks-2,ks-1
406                          do j=js,je
407                          do i=1,i2
408                          do ir=1,2
409          11                 w(ir,i,j,k)=BUFFRJ1(l1)
410                             l1=l1+1
411                          enddo
412          14              enddo
413                          enddo
414                          enddo
415                       endif
416                       if(idk.ne.nprock-1) THEN
417                          l1=1
418                          do k=ke+1,ke+2
419                          do j=js,je
420                          do i=1,i2
421                          do ir=1,2
422           8                 w(ir,i,j,k)=BUFFRJ2(l1)
423                             l1=l1+1
424                          enddo
425           6              enddo
426           1              enddo
```

```
427                                     enddo
428                                 endif
```

① 앞의 mpi_com_pm2 루틴과 거의 동일하며 단지 차이가 있는 부분은은 mpi_com_pm2 루틴에서는 data 교환이 이루어지는 array의 첫번째 index 크기가 5이기 때문에 w array를 BUFF에 저장할 때 do ir=1,5로 do loop를 수행했는데, mpi_com_pm2_2 루틴에서는 data 교환이 이루어지는 array의 첫번째 index 크기가 2이기 때문에 do=1,2로 do loop를 수행한다는 것이다.

5. mpi_com_output 루틴

A. 여러 array에 대해 각 rank에서 알고 있는 영역의 값을 0 rank로 전달

<MPI 병렬화 코드>

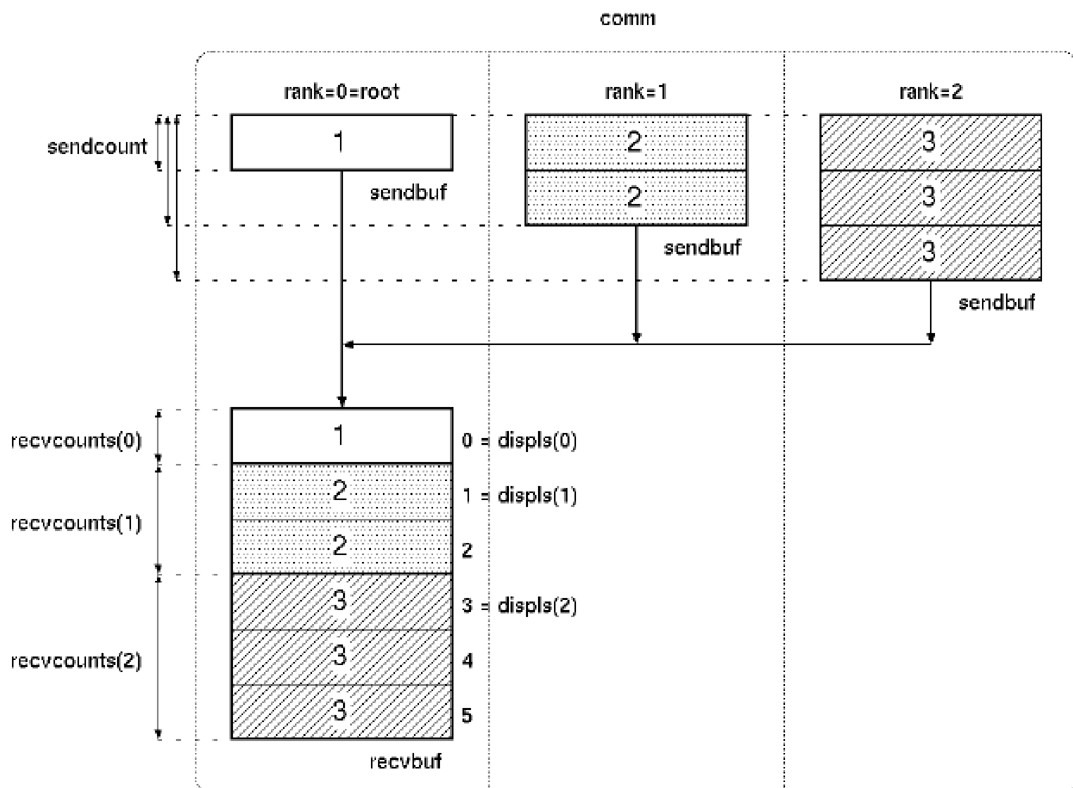
```
531                                 if(id.eq.0) then
532                                 DO kk=0,nprock-1
533                                     ks=kse(1,kk)
534                                     ke=kse(2,kk)
535                                     nk=ke-ks+1
536                                 DO jj=0,nprocj-1
537                                     ip=jj+nprocj*kk
538                                     js=jse(1,jj)
539                                     je=jse(2,jj)
540                                     nj=je-js+1
541                                     len=13*i2*nj*nk
542                                     ilrecv(ip)=len
543                                 enddo
544                                 enddo
545                                     iadrecv(0)=0
546                                 do ip=1,nproc-1
547                                     iadrecv(ip)=iadrecv(ip-1)+ilrecv(ip-1)
548                                 enddo
549                                 endif

552                                 ks=kse(1,idk)
553                                 ke=kse(2,idk)
554                                 nk=ke-ks+1
555                                 js=jse(1,idj)
556                                 je=jse(2,idj)
557                                 nj=je-js+1
558                                 ilsend=13*i2*nj*nk
559                                     l1=1
560                                     do k=ks,ke
561                                     do j=js,je
562                                     do i=1,i2
563                                     do ir=1,5
564                66                       BUFFS(l1)=w(ir,i,j,k)
565                                         l1=l1+1
566                                     enddo
567                 7                       BUFFS(l1)=ur(i,j,k)
568                                         l1=l1+1
```

```
569        4              BUFFS(I1)=vr(i,j,k)
570                       I1=I1+1
571        9              BUFFS(I1)=wr(i,j,k)
572                       I1=I1+1
573       10              BUFFS(I1)=p(i,j,k)
574                       I1=I1+1
575       13              BUFFS(I1)=wtk(i,j,k)
576        1              I1=I1+1
577       12              BUFFS(I1)=wtom(i,j,k)
578                       I1=I1+1
579        8              BUFFS(I1)=tk(i,j,k)
580                       I1=I1+1
581        8              BUFFS(I1)=tom(i,j,k)
582        1              I1=I1+1
583                    enddo
584                    enddo
585                    enddo
586                 CALL MPI_GATHERV(BUFFS,ilsend,        mpi_real8,
587                &                    BUFFR,ilrecv,iadrecv,mpi_real8,
588                &                    0,MPI_COMM_WORLD,ierr)
589                 if(id.eq.0) then
590                    I1=1
591                 DO kk=0,nprock-1
592                 ks=kse(1,kk)
593                 ke=kse(2,kk)
594                 DO jj=0,nprocj-1
595                 js=jse(1,jj)
596                 je=jse(2,jj)
597
598                    do k=ks,ke
599                    do j=js,je
600                    do i=1,i2
601                    do ir=1,5
602       17              w(ir,i,j,k)=BUFFR(I1)
603        3              I1=I1+1
604                    enddo
605        2              ur(i,j,k)=BUFFR(I1)
606                       I1=I1+1
607        3              vr(i,j,k)=BUFFR(I1)
608                       I1=I1+1
609                       wr(i,j,k)=BUFFR(I1)
610                       I1=I1+1
611                       p(i,j,k)=BUFFR(I1)
612                       I1=I1+1
613        3              wtk(i,j,k)=BUFFR(I1)
614                       I1=I1+1
615        2              wtom(i,j,k)=BUFFR(I1)
616                       I1=I1+1
617                       tk(i,j,k)=BUFFR(I1)
618                       I1=I1+1
619                       tom(i,j,k)=BUFFR(I1)
620        2              I1=I1+1
621                    enddo
622                    enddo
623                    enddo
624
625                 enddo
626                 enddo
627                 endif
```

① 먼저 0 rank에서 각 rank로부터 전송받을 data 크기를 나타내는 ilrecv array 및 data를 저장할 위치를 나타내는 iadrecv array를 계산한다.

② 전 rank에서 0 rank로 보낼 data 크기를 나타내는 ilsend를 계산하고 또한 전송할 data array들을 모두 BUFFS array에 저장한 후 MPI_GATHERV 루틴을 이용하여 0 rank의 BUFFR array에 모두 저장한다.

③ 0 rank에서는 BUFFR array로부터 전체 data를 추출하여 저장한다.



< Initializing Process >

1. datain 루틴

A. input data reading 과정

<Serial 최적화 코드>

```
 5      OPEN(15,FILE='double_cav.d')

17      read(iread,*) istart,ifinal
20      READ   (IREAD,*) RM, res, iturb
23      read(iread,*) delt
```

```
40      if(number_zone.eq.1) then
41      read(iread,500)
42      read(iread,*)   inputfom
43      write(*,*) inputfom
44      read(iread,500)
45      read(iread,*) xgx1,xgx2
46      read(iread,500)
47      read(iread,*) igx1,igx2
48      read(iread,500)
49      read(iread,*) pgx1
50      read(iread,500)
51      read(iread,*) qgx1
52      read(iread,500)
53      read(iread,*) ygy1,ygy2
54      read(iread,500)
55      read(iread,*) jgy1,jgy2
56      read(iread,500)
57      read(iread,*) pgy1
58      read(iread,500)
59      read(iread,*) qgy1
60      read(iread,500)
61      read(iread,*) zgz1,zgz2
62      read(iread,500)
63      read(iread,*) kgz1,kgz2
64      read(iread,500)
65      read(iread,*) pgz1
66      read(iread,500)
67      read(iread,*) qgz1
71      read(iread,500)
72      read(iread,*) iouttest,jouttest,kouttest
```

<MPI 병렬화 코드>

```
 7      real buff(39)
 8      integer ibuff(2,39)
 9      equivalence(ibuff,buff)
```

```
11          if(id.eq.id0) then
12          OPEN(15,FILE='double_cav.d')
13          endif

16          if(id.eq.id0) then
26          read(iread,*) istart,ifinal
29          READ   (IREAD,*) RM, res, iturb
32          read(iread,*) delt
33          endif

34          if(nproc.gt.1) then
35          if(id.eq.id0) then
36            buff(1)=RM
37            buff(2)=res
38            buff(3)=delt
39            ibuff(1,4)=istart
40            ibuff(2,4)=ifinal
41            ibuff(1,5)=iturb
42          endif
43          isrce=0
44          len=5
45          CALL MPI_BCAST(BUFF,len,MPI_REAL8,isrce,MPI_COMM_WORLD,ierr)
46          if(id.ne.id0) then
47            RM=buff(1)
48            res=buff(2)
49            delt=buff(3)
50            istart=ibuff(1,4)
51            ifinal=ibuff(2,4)
52            iturb=ibuff(1,5)
53          endif
54
55          endif
```

```
90          if(number_zone.eq.1) then
91          if(id.eq.id0) then
92          read(iread,500)
93          read(iread,*)   inputfom
94          write(*,*) inputfom
95          read(iread,500)
96          read(iread,*) xgx1,xgx2
97          read(iread,500)
98          read(iread,*) igx1,igx2
99          read(iread,500)
100         read(iread,*) pgx1
101         read(iread,500)
102         read(iread,*) qgx1
103         read(iread,500)
104         read(iread,*) ygy1,ygy2
105         read(iread,500)
106         read(iread,*) jgy1,jgy2
107         read(iread,500)
108         read(iread,*) pgy1
109         read(iread,500)
110         read(iread,*) qgy1
111         read(iread,500)
112         read(iread,*) zgz1,zgz2
113         read(iread,500)
```

```
114                        read(iread,*) kgz1,kgz2
115                        read(iread,500)
116                        read(iread,*) pgz1
117                        read(iread,500)
118                        read(iread,*) qgz1
119                        read(iread,500)
120                        read(iread,*) iouttest,jouttest,kouttest
121                        endif
122                        if(nproc.gt.1) then
123                        if(id.eq.id0) then
124                          buff(1)=xgx1
125                          buff(2)=xgx2
126                          buff(3)=pgx1
127                          buff(4)=qgx1
128                          buff(5)=ygy1
129                          buff(6)=ygy2
130                          buff(7)=pgy1
131                          buff(8)=qgy1
132                          buff(9)=zgz1
133                          buff(10)=zgz2
134                          buff(11)=pgz1
135                          buff(12)=qgz1
136                          ibuff(1,13)=igx1
137                          ibuff(2,13)=igx2
138                          ibuff(1,14)=jgy1
139                          ibuff(2,14)=jgy2
140                          ibuff(1,15)=kgz1
141                          ibuff(2,15)=kgz2
142                          ibuff(1,16)=iouttest
143                          ibuff(2,16)=kouttest
144                          ibuff(2,17)=inputfom
145                        endif
146                        isrce=0
147                        len=17
148                        CALL MPI_BCAST(BUFF,len,MPI_REAL8,isrce,MPI_COMM_WORLD,ierr)
149                        if(id.ne.id0) then
150                          xgx1=buff(1)
151                          xgx2=buff(2)
152                          pgx1=buff(3)
153                          qgx1=buff(4)
154                          ygy1=buff(5)
155                          ygy2=buff(6)
156                          pgy1=buff(7)
157                          qgy1=buff(8)
158                          zgz1=buff(9)
159                          zgz2=buff(10)
160                          pgz1=buff(11)
161                          qgz1=buff(12)
162                          igx1=ibuff(1,13)
163                          igx2=ibuff(2,13)
164                          jgy1=ibuff(1,14)
165                          jgy2=ibuff(2,14)
166                          kgz1=ibuff(1,15)
167                          kgz2=ibuff(2,15)
168                          iouttest=ibuff(1,16)
169                          kouttest=ibuff(2,16)
170                          inputfom=ibuff(1,17)
171                        endif
172
173                        endif
```

① Serial 코드와 달리 MPI 코드에서는 0번 rank에서 input data를 읽은 후 그 data를 전체 프로세스로 broadcating하여 전달한다.

② 이때 input data가 real 변수도 있고, integer 변수도 있기 때문에 equivalence를 사용하여 하나로 공유되게 하여 한번의 MPI_BCAST 함수 call로 해당 변수 모두를 broadcating할 수 있도록 하였다.

2. initco 루틴

A. restart 시 저장된 data를 reading하는 과정

<Serial 최적화 코드>

```
 88            WRITE(*,*) 'ITERATION CONTINUE...'
 89            OPEN( 8,FILE='flow_flat_plate.fom')
 90            I=1
 91            k=1
 92            DO 279 J=1,J2
 93            READ(8,666) W(1,I,J,K),W(2,I,J,K),W(3,I,J,K),W(4,I,J,K),
 94           *            W(5,I,J,K)   ,wtk(i,j,k),wtom(i,j,k)
 95             do ii=2,i2
 96             do kk=2,k2
 97            W(1,Ii,J,Kk)=W(1,1,J,K)
 98            W(2,Ii,J,Kk)=W(2,1,J,K)
 99            W(3,Ii,J,Kk)=W(3,1,J,K)
100            W(4,Ii,J,Kk)=W(4,1,J,K)
101            W(5,Ii,J,Kk)=W(5,1,J,K)
102            wtk(ii,j,kk)=wtk(1,j,k)
103            wtom(ii,j,kk)=wtom(1,j,k)
104            enddo
105            enddo
106        279 CONTINUE
107            close(8)
```

<MPI 병렬화 코드>

```
 93            if(id.eq.id0) then
 94            WRITE(*,*) 'ITERATION CONTINUE...'
 95            OPEN( 8,FILE='flow_flat_plate.fom')
 96            I=1
 97            k=1
 98            DO 279 J=1,J2
 99            READ(8,666) W(1,I,J,K),W(2,I,J,K),W(3,I,J,K),W(4,I,J,K),
100           *            W(5,I,J,K)   ,wtk(i,j,k),wtom(i,j,k)
101             do ii=2,i2
102             do kk=2,k2
103            W(1,Ii,J,Kk)=W(1,1,J,K)
104            W(2,Ii,J,Kk)=W(2,1,J,K)
105            W(3,Ii,J,Kk)=W(3,1,J,K)
106            W(4,Ii,J,Kk)=W(4,1,J,K)
107            W(5,Ii,J,Kk)=W(5,1,J,K)
```

```
108                         wtk(ii,j,kk)=wtk(1,j,k)
109                         wtom(ii,j,kk)=wtom(1,j,k)
110                         enddo
111                         enddo
112                     279 CONTINUE
113                         close(8)
114                         endif
115                         if(nproc.gt.1) then
116                         isrce=id0
117                         len=5*imx*jmx*kmx
118                         CALL MPI_BCAST(W,len,MPI_REAL8,isrce,MPI_COMM_WORLD,ierr)
119                         len=imx*jmx*kmx
120                         CALL MPI_BCAST(wtk,len,MPI_REAL8,isrce,MPI_COMM_WORLD,ierr)
121                         CALL MPI_BCAST(wtom,len,MPI_REAL8,isrce,MPI_COMM_WORLD,ierr)
122                         endif
```

① Serial 코드와 달리 MPI 코드에서는 0번 rank에서 restart하는데 필요한 data를 읽은 후 그 data를 전체 프로세스로 broadcating하여 전달한다.

② 최종적으로 ur, vr, wr, p, t, tk, tom, emu, w, wtk, wtom array 들이 계산되며, 이 array들은 모든 rank가 동일한 전체 영역의 값을 가지게 된다.

3. initcozone2 루틴

A. restart 시 저장된 data를 reading하는 과정

<Serial 최적화 코드>

```
196                         WRITE(*,*) 'ITERATION CONTINUE...'
197                         OPEN( 8,FILE='flowz2.fom')
198                         DO 299 K=1,K2z2
199                         DO 299 J=1,J2z2
200                         DO 298 I=1,i2z2
201                         READ(8,666) Wz2(1,I,J,K),Wz2(2,I,J,K),Wz2(3,I,J,K),Wz2(4,I,J,K),
202                        *          Wz2(5,I,J,K)   ,wtkz2(i,j,k),wtomz2(i,j,k)
203                     298 continue
204                     299 CONTINUE
205                         close(8)
206                     666 format(1x,7e13.5)
```

<MPI 병렬화 코드>

```
226                         if(id.eq.id0) then
227                         WRITE(*,*) 'ITERATION CONTINUE...'
228                         OPEN( 8,FILE='flowz2.fom')
229                         DO 299 K=1,K2z2
230                         DO 299 J=1,J2z2
231                         DO 298 I=1,i2z2
232                         READ(8,666) Wz2(1,I,J,K),Wz2(2,I,J,K),Wz2(3,I,J,K),Wz2(4,I,J,K),
233                        *          Wz2(5,I,J,K)   ,wtkz2(i,j,k),wtomz2(i,j,k)
234                     298 continue
```

```
235          299 CONTINUE
236                close(8)
237                endif
238                isrce=id0
239                len=5*imxz2*jmxz2*kmxz2
240                CALL MPI_BCAST(Wz2,len,MPI_REAL8,isrce,MPI_COMM_WORLD,ierr)
241                len=imxz2*jmxz2*kmxz2
242                CALL MPI_BCAST(wtkz2,len,MPI_REAL8,isrce,MPI_COMM_WORLD,ierr)
243             CALL MPI_BCAST(wtomz2,len,MPI_REAL8,isrce,MPI_COMM_WORLD,ierr)
244
245          666 format(1x,7e13.5)
```

① initco 루틴과 같이 MPI 코드에서는 0번 rank에서 restart하는데 필요한 data를 읽은 후 그 data를 전체 프로세스로 broadcating하여 전달한다.

② 최종적으로 urz2, vrz2, wrz2, pz2, tz2, tkz2, tomz2, emuz2, wz2, wtkz2, wtomz2 array 들이 계산되며, 이 array들은 모든 rank가 동일한 전체 영역의 값을 가지게 된다.

4. initcozone3 루틴

initcozone2 루틴과 변수 이름 정도만 차이 날 뿐 거의 똑 같은 루틴이기 때문에 동일한 형식으로 수정하여 mpi 병렬화 하였다. 다음 변수들은 initcozone3 루틴에서 계산되는 변수들이다. 이 array들은 Initcozone2와 마찬가지로 모든 rank가 동일한 전체 영역의 값을 가지게 된다.

① Urz3, vrz3, wrz3, pz3, tz3, tkz3, tomz3, emuz3, wz3, wtkz3, wtomz3

5. metric 루틴

A. VOL array 계산 과정

<Serial 최적화 코드>

```
20                  DO 10 K=2,KL
21                  N        = K   -1
22                  DO 10 J=2,JL
23                  M        = J   -1
24                  DO 10 I=2,IL
25                  L        = I   -1
26         1        XP       = .125*(X(1,I,J,K) +X(1,I,M,K) +X(1,I,M,N) +X(1,I,J,N)
27                 .               +X(1,L,J,K) +X(1,L,M,K) +X(1,L,M,N) +X(1,L,J,N))
28         4        YP       = .125*(X(2,I,J,K) +X(2,I,M,K) +X(2,I,M,N) +X(2,I,J,N)
29                 .               +X(2,L,J,K) +X(2,L,M,K) +X(2,L,M,N) +X(2,L,J,N))
30         1        ZP       = .125*(X(3,I,J,K) +X(3,I,M,K) +X(3,I,M,N) +X(3,I,J,N)
31                 .               +X(3,L,J,K) +X(3,L,M,K) +X(3,L,M,N) +X(3,L,J,N))
32                  XA       = X(1,I,J,K)
```

```
 33                YA          = X(2,I,J,K)
 34                ZA          = X(3,I,J,K)
 35                XB          = X(1,I,M,K)
 36                YB          = X(2,I,M,K)
 37                ZB          = X(3,I,M,K)
 38                XC          = X(1,I,M,N)
 39                YC          = X(2,I,M,N)
 40                ZC          = X(3,I,M,N)
 41                XD          = X(1,I,J,N)
 42                YD          = X(2,I,J,N)
 43                ZD          = X(3,I,J,N)
 44                XAX=XA+XB+XC+XD
 45                YAY=YA+YB+YC+YD
 46                ZAZ=ZA+ZB+ZC+ZD

178                VOLP11      = VOLP1(XP,XAX,YAC,ZBD,ZAC,YBD)
179                VOLP22      = VOLP2(YP,YAY,ZAC,XBD,XAC,ZBD)
180                VOLP33      = VOLP3(ZP,ZAZ,XAC,YBD,YAC,XBD)
181                VP6         = VOLPYM(VOLP11,VOLP22,VOLP33)
182      9         VOL(I,J,K)      = (VP1   +VP2   +VP3   +VP4   +VP5   +VP6)/6.

187             10 CONTINUE

189                DO 12 K=2,KL
190                DO 12 J=2,JL
191                VOL(1,J,K)      = VOL(2,J,K)
192      1         VOL(i2,J,K) = VOL(IL,J,K)
193             12 CONTINUE
194                do 19 j = 2,jl
195                do 19 i = 1,i2
196                vol(i,j,1) = vol(i,j,2)
197                vol(i,j,k2) = vol(i,j,kl)
198             19 continue
199                DO 14 K=1,K2
200                DO 14 I=1,I2
201                VOL(I,1,K)      = VOL(I,2,K)
202                VOL(I,j2,K) = VOL(I,JL,K)
203             14 CONTINUE
```

<MPI 병렬화 코드>

```
 27                do 10 k=max(kse(1,idk)-2,2),min(kse(2,idk)+2,kl)
 28                N           = K   -1
 30                do 10 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
 31                M           = J   -1
 32                DO 10 I=2,IL
 33                L           = I   -1
 34      6         XP          = .125*(X(1,I,J,K) +X(1,I,M,K) +X(1,I,M,N) +X(1,I,J,N)
 35               .                     +X(1,L,J,K) +X(1,L,M,K) +X(1,L,M,N) +X(1,L,J,N))
 36      2         YP          = .125*(X(2,I,J,K) +X(2,I,M,K) +X(2,I,M,N) +X(2,I,J,N)
 37               .                     +X(2,L,J,K) +X(2,L,M,K) +X(2,L,M,N) +X(2,L,J,N))
 38                ZP          = .125*(X(3,I,J,K) +X(3,I,M,K) +X(3,I,M,N) +X(3,I,J,N)
 39               .                     +X(3,L,J,K) +X(3,L,M,K) +X(3,L,M,N) +X(3,L,J,N))
 40                XA          = X(1,I,J,K)
 41                YA          = X(2,I,J,K)
 42                ZA          = X(3,I,J,K)
 43                XB          = X(1,I,M,K)
```

```
44                YB          = X(2,I,M,K)
45                ZB          = X(3,I,M,K)
46                XC          = X(1,I,M,N)
47                YC          = X(2,I,M,N)
48                ZC          = X(3,I,M,N)
49                XD          = X(1,I,J,N)
50                YD          = X(2,I,J,N)
51                ZD          = X(3,I,J,N)
52                XAX=XA+XB+XC+XD
53                YAY=YA+YB+YC+YD
54                ZAZ=ZA+ZB+ZC+ZD

186               VOLP11      = VOLP1(XP,XAX,YAC,ZBD,ZAC,YBD)
187               VOLP22      = VOLP2(YP,YAY,ZAC,XBD,XAC,ZBD)
188       2       VOLP33      = VOLP3(ZP,ZAZ,XAC,YBD,YAC,XBD)
189               VP6         = VOLPYM(VOLP11,VOLP22,VOLP33)
190      10       VOL(I,J,K)      = (VP1   +VP2   +VP3   +VP4   +VP5   +VP6)/6.

197       1    10 CONTINUE

200               do 12 k=max(kse(1,idk)-2,2),min(kse(2,idk)+2,kl)
202               do 12 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
203               VOL(1,J,K)      = VOL(2,J,K)
204               VOL(i2,J,K) = VOL(IL,J,K)
205            12 CONTINUE
206
207               if(idk.eq.0) then
209               do 19 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
210               do 19 i = 1,i2
211               vol(i,j,1) = vol(i,j,2)
212            19 continue
213               endif
214               if(idk.eq.nprock-1) then
216               do j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
217               do i = 1,i2
218               vol(i,j,k2) = vol(i,j,kl)
219               enddo
220               enddo
221               endif
222
223               if(idj.eq.0) then
225               do 14 k=max(kse(1,idk)-2,1),min(kse(2,idk)+2,k2)
226               DO 14 I=1,I2
227               VOL(I,1,K)      = VOL(I,2,K)
228            14 CONTINUE
229               endif
230               if(idj.eq.nprocj-1) then
232               do k=max(kse(1,idk)-2,1),min(kse(2,idk)+2,k2)
233               DO I=1,I2
234       1       VOL(I,j2,K) = VOL(I,JL,K)
235               enddo
236               enddo
237               endif
```

① VOL array를 구할 때 각각의 MPI rank가 해당 영역만 계산을 하도록 지정해 주었다. 계산되는 vol array에 대해 나중에 경계부분의 값을 을 access하기 때문에

각 rank의 구간이 경계부분을 서로 중첩하면서 포함하여 계산하도록 do-loop를 수행하고 있다. 또한 j=1, j2 및 k=1,k2에 대해 계산할 때 idj, idk 변수를 이용하여 이 영역을 포함하는 rank만 수행할 수 있도록 하였다.

B. xix, etx, ztx, xix, etx, ztx array 계산 과정

<Serial 최적화 코드>

```
207                 do 200 i=2,il
208                 l=i-1
209                 do 200 j=2,jl
210                 m=j-1
211                 do 200 k=2,kl
212                 n=k-1
213     12          SX        = (X(2,I,J,N)   -X(2,I,M,K))*(X(3,I,J,K)   -X(3,I,M,N))
214                .          -(X(3,I,J,N)   -X(3,I,M,K))*(X(2,I,J,K)   -X(2,I,M,N))
215                .          +(X(2,L,J,N)   -X(2,L,M,K))*(X(3,L,J,K)   -X(3,L,M,N))
216                .          -(X(3,L,J,N)   -X(3,L,M,K))*(X(2,L,J,K)   -X(2,L,M,N))

225      6          xix(i,j,k)=sx/vol(i,j,k)/4.

260      1      200 continue

263                 DO 22 K=2,KL
264                 DO 22 J=2,JL
265                 xix(1,j,k)=xix(2,j,k)
268                 etx(1,j,k)=-etx(2,j,k)
271                 ztx(1,j,k)=-ztx(2,j,k)
274                 xix(i2,j,k)=xix(il,j,k)
277                 etx(i2,j,k)=-etx(il,j,k)
280                 ztx(i2,j,k)=-ztx(il,j,k)
283              22 CONTINUE

285                 do 29 j = 2,jl
286                 do 29 i = 1,I2
287                 xix(i,j,1)=xix(i,j,2)
290                 etx(i,j,1)=etx(i,j,2)
293                 ztx(i,j,1)=-ztx(i,j,2)
296                 xix(i,j,k2)=xix(i,j,kl)
299                 etx(i,j,k2)=etx(i,j,kl)
302                 ztx(i,j,k2)=-ztx(i,j,kl)
305              29 continue
306
```

<MPI 병렬화 코드>

```
243                 do 200 k=max(kse(1,idk)-2,2),min(kse(2,idk)+2,kl)
244                 n=k-1
246                 do 200 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
247                 m=j-1
248                 do 200 i=2,il
249                 l=i-1
250      7          SX        = (X(2,I,J,N)   -X(2,I,M,K))*(X(3,I,J,K)   -X(3,I,M,N))
251                .          -(X(3,I,J,N)   -X(3,I,M,K))*(X(2,I,J,K)   -X(2,I,M,N))
```

```
252                      .              +(X(2,L,J,N)   -X(2,L,M,K))*(X(3,L,J,K)   -X(3,L,M,N))
253                      .              -(X(3,L,J,N)   -X(3,L,M,K))*(X(2,L,J,K)   -X(2,L,M,N))

262          10            xix(i,j,k)=sx/vol(i,j,k)/4.

298                    200 continue

302                        do 22 k=max(kse(1,idk)-2,2),min(kse(2,idk)+2,kl)
304                        do 22 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
305                        xix(1,j,k)=xix(2,j,k)
308           1            etx(1,j,k)=-etx(2,j,k)
311           1            ztx(1,j,k)=-ztx(2,j,k)
314                        xix(i2,j,k)=xix(il,j,k)
317                        etx(i2,j,k)=-etx(il,j,k)
320                        ztx(i2,j,k)=-ztx(il,j,k)
323                     22 CONTINUE

325                        if(idk.eq.0) then
327                        do 29 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
328                        do 29 i = 1,I2
329                        xix(i,j,1)=xix(i,j,2)
332                        etx(i,j,1)=etx(i,j,2)
335                        ztx(i,j,1)=-ztx(i,j,2)
338                     29 continue
339                        endif
340                        if(idk.eq.nprock-1) then
342                        do j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
343                        do i = 1,I2
344                        xix(i,j,k2)=xix(i,j,kl)
347                        etx(i,j,k2)=etx(i,j,kl)
350           1            ztx(i,j,k2)=-ztx(i,j,kl)
353                        enddo
354                        enddo
355                        endif
```

① 위의 VOL array를 구할 때와 마찬가지로 xix array를 구할 때에도 각각의 MPI rank가 해당 영역만 계산을 하도록 지정해 주었다. 나중에 경계 부분의 값을 access하기 때문에 각 rank의 구간이 경계부분을 서로 중첩하여 포함하여 계산하도록 do-loop를 수행하고 있다. 또한 k=1,k2에 대해 계산할 때 idk 변수를 이용하여 이 영역을 포함하는 rank만 수행할 수 있도록 하였다.

6. metriczone2, metriczone3 루틴

metric 루틴과 변수 이름 정도만 차이날 뿐 거의 같은 루틴이기 때문에 metric 루틴과 비슷하게 수정하여 병렬화 하였다.

< Main Process >

1. double_cav 루틴

A. main 코드에 mpi 기본 루틴 삽입

<Serial 최적화 코드>

```
26        OPEN( 7,FILE='gri3d.dat',form='UNFORMATTED')
27        OPEN(15,FILE='ns3d.d')
28        OPEN(38,FILE='hist.dat')
29        open(27,file='ns3d.out')

31        call datain
32        call gridin
```

<MPI 병렬화 코드>

```
22        call mpi_init(ierr)
23        call mpi_comm_rank(mpi_comm_world,id, ierr)
24        call mpi_comm_size(mpi_comm_world,nproc, ierr)

27        if(id.eq.id0) then
28        OPEN( 7,FILE='gri3d.dat',form='UNFORMATTED')
29        OPEN(15,FILE='ns3d.d')
30        OPEN(38,FILE='hist.dat')
31        open(27,file='ns3d.out')
32        endif

34        call datain
36        call decomp_2d

76        call gridin

374       call mpi_finalize(ierr)
```

① mian 코드인 double_cav 루틴에 기본적인 mpi 루틴을 삽입하였다.

② 영역을 나누어 계산되어질 array들을 효율적으로 분할하는 decomp_2d 루틴이 사용되었다.

B. Main iteration 구간

<Serial 최적화 코드>

```
53        DO 2000 ITR=ISTART,IFINAL

56        DO 301 K=1,K2
57        DO 301 J=1,J2
```

```
 58                               DO 301 I=1,I2
 59          9                      Wn(1,I,J,K)=W(1,I,J,K)
 60         60                      Wn(2,I,J,K)=W(2,I,J,K)
 61          2                      Wn(3,I,J,K)=W(3,I,J,K)
 62                                 Wn(4,I,J,K)=W(4,I,J,K)
 63        100                      Wn(5,I,J,K)=W(5,I,J,K)
 64                           301 CONTINUE

 87                               if(iturb.eq.2) then
 88                               DO 311 K=1,K2
 89                               DO 311 J=1,J2
 90                               DO 311 I=1,I2
 91         25                      wtkn(I,J,K)=wtk(I,J,K)
 92         42                      wtomn(I,J,K)=wtom(i,J,K)
 93                           311 CONTINUE
 94
 95                               if(number_zone.ge.2) then
 96                               DO 312 K=1,K2z2
 97                               DO 312 J=1,J2z2
 98                               DO 312 I=1,I2z2
 99          2                      wtknz2(I,J,K)=wtkz2(I,J,K)
100          3                      wtomnz2(I,J,K)=wtomz2(I,J,K)
101          1                312 CONTINUE
102                               DO 313 K=1,K2z3
103                               DO 313 J=1,J2z3
104                               DO 313 I=1,I2z3
105          1                      wtknz3(I,J,K)=wtkz3(I,J,K)
106          6                      wtomnz3(I,J,K)=wtomz3(I,J,K)
107                           313 CONTINUE
108                               endif
109                               endif

175                               if(iturb.ge.2) then
177                               do 2011 itrunge=1,4
178                               CALL turbkom(itr,itrunge)
180                               if(number_zone.ge.2) then
181                               CALL turbkomzone2(itr,itrunge)
182                               call turbkomzone3(itr,itrunge)
183                               endif
185                               call boundturb
186                               if(number_zone.ge.2) then
187                               call boundturbzone2
188                               call boundturbzone3
189                               endif
191                          2011 continue
192                               endif

194                               if((mod(itr,100).eq.0)) then
195                               call outputtest(iouttest,jouttest,kouttest)
196                               OPEN( 8,FILE='flow.fom')
197                               DO 399 K=1,K2
198                               DO 399 J=1,J2
199                               DO 399 I=1,I2
200                               WRITE(8,667) W(1,I,J,K),W(2,I,J,K),W(3,I,J,K),W(4,I,J,K),
201                              .                W(5,I,J,K),wtk(i,j,k),wtom(i,j,k)
202                           399 CONTINUE
203                               close(8)

227                               endif
```

```
230                     write(6,*) 'Timing Step',tt0
231                2000 CONTINUE
```

<MPI 병렬화 코드>

```
100                     DO 2000 ITR=ISTART,IFINAL

105                     do 301 k=kse(1,idk),kse(2,idk)
106                     do 301 j=jse(1,idj),jse(2,idj)
107                     DO 301 I=1,I2
108       105            Wn(1,I,J,K)=W(1,I,J,K)
109        12            Wn(2,I,J,K)=W(2,I,J,K)
110        88            Wn(3,I,J,K)=W(3,I,J,K)
111        54            Wn(4,I,J,K)=W(4,I,J,K)
112        26            Wn(5,I,J,K)=W(5,I,J,K)
113         2       301 CONTINUE

140                     if(iturb.eq.2) then
143                     do 311 k=max(kse(1,idk)-2,1),min(kse(2,idk)+2,k2)
144                     do 311 j=max(jse(1,idj)-2,1),min(jse(2,idj)+2,j2)
145                     DO 311 I=1,I2
146       105            wtkn(I,J,K)=wtk(I,J,K)
147        26            wtomn(I,J,K)=wtom(i,J,K)
148         2       311 CONTINUE
149
150                     if(number_zone.ge.2) then
153                     do 312 k=max(ksez2(1,idk)-2,2),min(ksez2(2,idk)+2,klz2)
154                     do 312 j=max(jsez2(1,idj)-2,2),min(jsez2(2,idj)+2,jlz2)
155                     DO 312 I=1,I2z2
156        13            wtknz2(I,J,K)=wtkz2(I,J,K)
157         8            wtomnz2(I,J,K)=wtomz2(I,J,K)
158                 312 CONTINUE
161                     do 313 k=max(ksez3(1,idk)-2,2),min(ksez3(2,idk)+2,klz3)
162                     do 313 j=max(jsez3(1,idj)-2,2),min(jsez3(2,idj)+2,jlz3)
163                     DO 313 I=1,I2z3
164         4            wtknz3(I,J,K)=wtkz3(I,J,K)
165         7            wtomnz3(I,J,K)=wtomz3(I,J,K)
166                 313 CONTINUE
167                     endif
168                     endif

262                     if(iturb.ge.2) then
264                     do 2011 itrunge=1,4
265                     CALL turbkom(itr,itrunge)
267                     if(number_zone.ge.2) then
268                     CALL turbkomzone2(itr,itrunge)
269                     call turbkomzone3(itr,itrunge)
270                     endif
272                     call boundturb
273                     if(number_zone.ge.2) then
274                     call boundturbzone2
275                     call boundturbzone3
276                     endif
278                2011 continue
279                     endif

281                     if((mod(itr,100).eq.0)) then
```

```
282                              call outputtest(iouttest,jouttest,kouttest)
283                              if(id.eq.id0) then
284                              OPEN( 8,FILE='flow.fom')
285                              DO 399 K=1,K2
286                              DO 399 J=1,J2
287                              DO 399 I=1,I2
288                              WRITE(8,667) W(1,I,J,K),W(2,I,J,K),W(3,I,J,K),W(4,I,J,K),
289                             .             W(5,I,J,K),wtk(i,j,k),wtom(i,j,k)
290                          399 CONTINUE
291                              close(8)
292                              endif

320                              endif

323                              if(id.eq.id0) then
324                              write(6,*) 'Timing Step',tt0
325                              endif
326                         2000 CONTINUE
```

① Wn array를 계산할 때 각 rank의 해당부분만 계산하도록 do loop를 수행하였다.

② Wtkn, wtomn, wtknz2, wtomnz2, wtknz3, wtomnz3 를 계산할 때 뒤에서 각 rank의 경계부분에 있는 값을 access하기 때문에 각 rank에 대한 계산 영역을 경계부분도 포함하도록 잡아서 do loop를 수행하고 있다.

③ 수행 결과를 write하는 부분에서는 rank=0일때만 수행되도록 하고 있다.

C. Core Iteration 구간

<Serial 최적화 코드>

```
111                              do 2001 itrunge=1,4
112                              CALL NASTOK(itr,itrunge)
114                              rtrms=alog10(rtrms)
115                              WRITE (*,666)     itr,itrunge,rtrms

119                              do 100 k=2,kl
120                              do 100 j=2,jl
121                              do 100 i=2,il
122          142                  ur(i,j,k)=w(2,i,j,k)/w(1,i,j,k)
123          353                  vr(i,j,k)=w(3,i,j,k)/w(1,i,j,k)
124           23                  wr(i,j,k)=w(4,i,j,k)/w(1,i,j,k)
125          110                  p(i,j,k)=(w(5,i,j,k)-0.5*w(1,i,j,k)*(ur(i,j,k)**2
126                             *    +vr(i,j,k)**2+wr(i,j,k)**2))*(gamm)
127          264                  t(i,j,k)=p(i,j,k)/w(1,i,j,k)
128                          100 continue
129
130                              if(number_zone.ge.2) then

134                              CALL NASTOKzone2(itr,itrunge)
135                              do 101 k=2,klz2
136                              do 101 j=2,jlz2
137                              do 101 i=2,ilz2
138            6                 urz2(i,j,k)=wz2(2,i,j,k)/wz2(1,i,j,k)
139           48                 vrz2(i,j,k)=wz2(3,i,j,k)/wz2(1,i,j,k)
```

```
140          4          wrz2(i,j,k)=wz2(4,i,j,k)/wz2(1,i,j,k)
141          3          pz2(i,j,k)=(wz2(5,i,j,k)-0.5*wz2(1,i,j,k)*(urz2(i,j,k)**2
142                    *    +vrz2(i,j,k)**2+wrz2(i,j,k)**2))*(gamm)
143         30          tz2(i,j,k)=pz2(i,j,k)/wz2(1,i,j,k)
144                 101 continue

148                     CALL NASTOKzone3(itr,itrunge)
149                     do 102 k=2,klz3
150                     do 102 j=2,jlz3
151                     do 102 i=2,ilz3
152          4          urz3(i,j,k)=wz3(2,i,j,k)/wz3(1,i,j,k)
153         19          vrz3(i,j,k)=wz3(3,i,j,k)/wz3(1,i,j,k)
154          1          wrz3(i,j,k)=wz3(4,i,j,k)/wz3(1,i,j,k)
155          6          pz3(i,j,k)=(wz3(5,i,j,k)-0.5*wz3(1,i,j,k)*(urz3(i,j,k)**2
156                    *    +vrz3(i,j,k)**2+wrz3(i,j,k)**2))*(gamm)
157         15          tz3(i,j,k)=pz3(i,j,k)/wz3(1,i,j,k)
158                 102 continue
159                     endif
160
161                     call bound
162                     if(number_zone.ge.2) then
163                     call boundzone2
164                     call boundzone3
165                     endif

168                2001 continue
```

<MPI 병렬화 코드>

```
170                     do 2001 itrunge=1,4
171                     CALL NASTOK(itr,itrunge)
172                     ks=kse(1,idk)
173                     ke=kse(2,idk)
174                     js=jse(1,idj)
175                     je=jse(2,idj)
176                     call mpi_com_pm2(w,imx,jmx,kmx,i2,k2,
177                    & nprocj,nprock,idj,idk,js,je,ks,ke)
179                     rtrms=alog10(rtrms)
180                     if(id.eq.id0) then
181                     WRITE (*,666)      itr,itrunge,rtrms
182                     endif

188                     do 100 k=max(kse(1,idk)-2,2),min(kse(2,idk)+2,kl)
189                     do 100 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
190                     do 100 i=2,il
191        136          ur(i,j,k)=w(2,i,j,k)/w(1,i,j,k)
192        210          vr(i,j,k)=w(3,i,j,k)/w(1,i,j,k)
193        118          wr(i,j,k)=w(4,i,j,k)/w(1,i,j,k)
194        577          p(i,j,k)=(w(5,i,j,k)-0.5*w(1,i,j,k)*(ur(i,j,k)**2
195                    *    +vr(i,j,k)**2+wr(i,j,k)**2))*(gamm)
196        106          t(i,j,k)=p(i,j,k)/w(1,i,j,k)
197          2      100 continue

200                     if(number_zone.ge.2) then

204                     CALL NASTOKzone2(itr,itrunge)
205                     ksz2=ksez2(1,idk)
```

```
206                          kez2=ksez2(2,idk)
207                          jsz2=jsez2(1,idj)
208                          jez2=jsez2(2,idj)
209                          call mpi_com_pm2(wz2,imxz2,jmxz2,kmxz2,i2z2,k2z2,
210                         & nprocj,nprock,idj,idk,jsz2,jez2,ksz2,kez2)

213                          do 101 k=max(ksez2(1,idk)-2,2),min(ksez2(2,idk)+2,klz2)
214                          do 101 j=max(jsez2(1,idj)-2,2),min(jsez2(2,idj)+2,jlz2)
215                          do 101 i=2,ilz2
216          7               urz2(i,j,k)=wz2(2,i,j,k)/wz2(1,i,j,k)
217         17               vrz2(i,j,k)=wz2(3,i,j,k)/wz2(1,i,j,k)
218          7               wrz2(i,j,k)=wz2(4,i,j,k)/wz2(1,i,j,k)
219         66               pz2(i,j,k)=(wz2(5,i,j,k)-0.5*wz2(1,i,j,k)*(urz2(i,j,k)**2
220                         *    +vrz2(i,j,k)**2+wrz2(i,j,k)**2))*(gamm)
221         15               tz2(i,j,k)=pz2(i,j,k)/wz2(1,i,j,k)
222          4          101 continue

226                          CALL NASTOKzone3(itr,itrunge)
227                          ksz3=ksez3(1,idk)
228                          kez3=ksez3(2,idk)
229                          jsz3=jsez3(1,idj)
230                          jez3=jsez3(2,idj)
231                          call mpi_com_pm2(wz3,imxz3,jmxz3,kmxz3,i2z3,k2z3,
232                         & nprocj,nprock,idj,idk,jsz3,jez3,ksz3,kez3)

235                          do 102 k=max(ksez3(1,idk)-2,2),min(ksez3(2,idk)+2,klz3)
236                          do 102 j=max(jsez3(1,idj)-2,2),min(jsez3(2,idj)+2,jlz3)
237                          do 102 i=2,ilz3
238          7               urz3(i,j,k)=wz3(2,i,j,k)/wz3(1,i,j,k)
239         15               vrz3(i,j,k)=wz3(3,i,j,k)/wz3(1,i,j,k)
240          2               wrz3(i,j,k)=wz3(4,i,j,k)/wz3(1,i,j,k)
241         22               pz3(i,j,k)=(wz3(5,i,j,k)-0.5*wz3(1,i,j,k)*(urz3(i,j,k)**2
242                         *    +vrz3(i,j,k)**2+wrz3(i,j,k)**2))*(gamm)
243          8               tz3(i,j,k)=pz3(i,j,k)/wz3(1,i,j,k)
244                     102 continue
245
246                          endif
247
248                          call bound
249                          if(number_zone.ge.2) then
250                          call boundzone2
251                          call boundzone3
252                          endif

255                    2001 continue
```

① Nastok 루틴을 지나면 각 rank에 해당하는 영역 내의 w array 값만 알게 되는데, 다음에 오는 계산과정에서 경계 부근의 w array를 access하기 때문에 mpi_comm_pm2 루틴을 call하여 경계 부근의 data를 미리 전달해 준다.

② Nastokzone2, nastokzone3 루틴 후의 계산과정도 동일하게 진행된다.

2. nastok 루틴

A. dw array 계산 부분 – part 1

<Serial 최적화 코드>

```
 21                 do 100 k=2,kl
 22                 do 100 j=2,jl
 23        2        do 10 i=1,i2
 24
 25      178         pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
 26      279         pi=sqrt(xix(i,j,k)**2+xiy(i,j,k)**2+xiz(i,j,k)**2)
 27                 cx=xix(i,j,k)/pi
 28      477         cy=xiy(i,j,k)/pi
 29       30         cz=xiz(i,j,k)/pi
 30      118         the=cx*ur(i,j,k)+cy*vr(i,j,k)+cz*wr(i,j,k)
 31       42         cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))
 32       98         alpha=w(1,i,j,k)/(cc*sqrt(2.))
 33      151         beta=1./(w(1,i,j,k)*cc*sqrt(2.))


265       80           dw(ir,i,j,k)=dw(ir,i,j,k)+(
266                * (1.+0.5*p1)*(fm(ir,l)-fm(ir,i)) -0.5*p2*(fm(ir,l+1)-fm(ir,l)))

272        5      100 continue
```

<MPI 병렬화 코드>

```
 30                 do 100 k=max(kse(1,idk),2),min(kse(2,idk),kl)
 32                 do 100 j=max(jse(1,idj),2),min(jse(2,idj),jl)
 33        4        do 10 i=1,i2
 34
 35      403         pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
 36      189         pi=sqrt(xix(i,j,k)**2+xiy(i,j,k)**2+xiz(i,j,k)**2)
 37                 cx=xix(i,j,k)/pi
 38                 cy=xiy(i,j,k)/pi
 39                 cz=xiz(i,j,k)/pi
 40                 the=cx*ur(i,j,k)+cy*vr(i,j,k)+cz*wr(i,j,k)
 41       79         cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))
 42      207         alpha=w(1,i,j,k)/(cc*sqrt(2.))
 43       81         beta=1./(w(1,i,j,k)*cc*sqrt(2.))

275      520           dw(ir,i,j,k)=dw(ir,i,j,k)+(
276                * (1.+0.5*p1)*(fm(ir,l)-fm(ir,i)) -0.5*p2*(fm(ir,l+1)-fm(ir,l)))

282               100 continue
```

B. dw array 계산 부분 – part 2

<Serial 최적화 코드>

```
276                 do 200 k=2,kl
277                   do 20 j=1,j2
278        1          do i=2,il
279
280       51             pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
```

```
281          304               pi=sqrt(etx(i,j,k)**2+ety(i,j,k)**2+etz(i,j,k)**2)
282                            ex=etx(i,j,k)/pi
283          380               ey=ety(i,j,k)/pi
284           39               ez=etz(i,j,k)/pi
285          117               the=ex*ur(i,j,k)+ey*vr(i,j,k)+ez*wr(i,j,k)
286           48               cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))
287           25               alpha=w(1,i,j,k)/(cc*sqrt(2.))
288          435               beta=1./(w(1,i,j,k)*cc*sqrt(2.))

442           87              gj(ir,i,j)=fprc*vol(i,j,k)
443          297              gmj(ir,i,j)=fmrc*vol(i,j,k)

446                      20 continue

449                           do 21 j=2,jl
450                           m=j-1

478          446              dw(ir,i,j,k)=dw(ir,i,j,k)+(1.+0.5*p1)*(gj(ir,i,j)-gj(ir,i,m))
479                         *        -0.5*p2*(gj(ir,i,m)-gj(ir,i,m-1))

483                      21 continue

484                           do 23 j=2,jl
485                           m=j+1

513          413              dw(ir,i,j,k)=dw(ir,i,j,k)+
514                         * (1.+0.5*p1)*(gmj(ir,i,m)-gmj(ir,i,j))
515                         * -0.5*p2*(gmj(ir,i,m+1)-gmj(ir,i,m))

519                      23 continue

521                     200 continue
```

<MPI 병렬화 코드>

```
286                          do 200 k=max(kse(1,idk),2),min(kse(2,idk),kl)
288                          do 20 j=max(jse(1,idj)-2,1),min(jse(2,idj)+2,j2)
289                           do i=2,il
290
291          475               pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
292          165               pi=sqrt(etx(i,j,k)**2+ety(i,j,k)**2+etz(i,j,k)**2)
293                            ex=etx(i,j,k)/pi
294                            ey=ety(i,j,k)/pi
295           62               ez=etz(i,j,k)/pi
296          206               the=ex*ur(i,j,k)+ey*vr(i,j,k)+ez*wr(i,j,k)
297          103               cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))
298                            alpha=w(1,i,j,k)/(cc*sqrt(2.))
299                            beta=1./(w(1,i,j,k)*cc*sqrt(2.))

453          463              gj(ir,i,j)=fprc*vol(i,j,k)
454          509              gmj(ir,i,j)=fmrc*vol(i,j,k)

456            1          enddo
457            1      20 continue

461                          do 21 j=max(jse(1,idj),2),min(jse(2,idj),jl)
462                           m=j-1
```

```
490      596          dw(ir,i,j,k)=dw(ir,i,j,k)+(1.+0.5*p1)*(gj(ir,i,j)-gj(ir,i,m))
491                 *       -0.5*p2*(gj(ir,i,m)-gj(ir,i,m-1))

495        1       21 continue

497                     do 23 j=max(jse(1,idj),2),min(jse(2,idj),jl)
498                       m=j+1

526      669              dw(ir,i,j,k)=dw(ir,i,j,k)+
527                 * (1.+0.5*p1)*(gmj(ir,i,m)-gmj(ir,i,j))
528                 * -0.5*p2*(gmj(ir,i,m+1)-gmj(ir,i,m))

532                23 continue

534               200 continue
```

C. dw array 계산 부분 – part 3

<Serial 최적화 코드>

```
522                       do 30 k=1,k2
523                       do j=2,jl
524                       do i=2,il
525      118                pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
526      334                pi=sqrt(ztx(i,j,k)**2+zty(i,j,k)**2+ztz(i,j,k)**2)
527                         zx=ztx(i,j,k)/pi
528      433                zy=zty(i,j,k)/pi
529       34                zz=ztz(i,j,k)/pi
530      122                the=zx*ur(i,j,k)+zy*vr(i,j,k)+zz*wr(i,j,k)
531       40                cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))
532                         alpha=w(1,i,j,k)/(cc*sqrt(2.))
533      467                beta=1./(w(1,i,j,k)*cc*sqrt(2.))

686       88              hk(ir,i,j,k)=gprc*vol(i,j,k)
687      356              hmk(ir,i,j,k)=gmrc*vol(i,j,k)

690       28            enddo
691        1            enddo
692                30 continue

693                       do 31 k=2,kl
694                       n=k-1
695                       do j=2,jl

723      362              dw(ir,i,j,k)=dw(ir,i,j,k)+(1+0.5*p1)*(hk(ir,i,j,k)-hk(ir,i,j,n))
724                 *       -0.5*p2*(hk(ir,i,j,n)-hk(ir,i,j,n-1))

728                     enddo
729                31 continue

731                       do 33 k=2,kl
732                       n=k+1
733                       do j=2,jl

761      372              dw(ir,i,j,k)=dw(ir,i,j,k)+
762                 *   (1.+0.5*p1)*(hmk(ir,i,j,n)-hmk(ir,i,j,k))
```

```
763                              *   -0.5*p2*(hmk(ir,i,j,n+1)-hmk(ir,i,j,n))

767                               enddo
768                            33 continue
```

<MPI 병렬화 코드>

```
536                                do 30 k=max(kse(1,idk)-2,1),min(kse(2,idk)+2,k2)
538                                do j=max(jse(1,idj),2),min(jse(2,idj),jl)
539            2                   do i=2,il
540          557                     pi2=(gamma-1)*(ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)/2.
541          166                     pi=sqrt(ztx(i,j,k)**2+zty(i,j,k)**2+ztz(i,j,k)**2)
542                                  zx=ztx(i,j,k)/pi
543                                  zy=zty(i,j,k)/pi
544                                  zz=ztz(i,j,k)/pi
545           65                     the=zx*ur(i,j,k)+zy*vr(i,j,k)+zz*wr(i,j,k)
546          205                     cc=sqrt(gamma*p(i,j,k)/w(1,i,j,k))
547                                  alpha=w(1,i,j,k)/(cc*sqrt(2.))
548          140                     beta=1./(w(1,i,j,k)*cc*sqrt(2.))

701          547                  hk(ir,i,j,k)=gprc*vol(i,j,k)
702          676                  hmk(ir,i,j,k)=gmrc*vol(i,j,k)

705                               enddo
706                               enddo
707            1               30 continue

709                                do 31 k=max(kse(1,idk),2),min(kse(2,idk),kl)
710                                n=k-1
712                                do j=max(jse(1,idj),2),min(jse(2,idj),jl)

740          565                  dw(ir,i,j,k)=dw(ir,i,j,k)+(1+0.5*p1)*(hk(ir,i,j,k)-hk(ir,i,j,n))
741                              *        -0.5*p2*(hk(ir,i,j,n)-hk(ir,i,j,n-1))

745                               enddo
746                            31 continue

749                                do 33 k=max(kse(1,idk),2),min(kse(2,idk),kl)
750                                n=k+1
752                                do j=max(jse(1,idj),2),min(jse(2,idj),jl)

780          728                  dw(ir,i,j,k)=dw(ir,i,j,k)+
781                              *   (1.+0.5*p1)*(hmk(ir,i,j,n)-hmk(ir,i,j,k))
782                              *   -0.5*p2*(hmk(ir,i,j,n+1)-hmk(ir,i,j,n))

786                               enddo
787                            33 continue
```

① dw array를 구할 때 각각의 MPI rank가 해당 영역만 계산을 하도록 지정해 주었다.

D. W array 계산 부분

<Serial 최적화 코드>

```
786                    DO 60 K=2,KL
787                    DO 60 J=2,JL
788                    DO 60 I=2,IL
789       205          W(1,I,J,K)   = Wn(1,I,J,K)-rungefactor/vol(i,j,k)*DW(1,I,J,K)
790       125          W(2,I,J,K)   = Wn(2,I,J,K)-rungefactor/vol(i,j,k)*DW(2,I,J,K)
791       195          W(3,I,J,K)   = Wn(3,I,J,K)-rungefactor/vol(i,j,k)*DW(3,I,J,K)
792         1          W(4,I,J,K)   = Wn(4,I,J,K)-rungefactor/vol(i,j,k)*DW(4,I,J,K)
793       245          W(5,I,J,K)   = Wn(5,I,J,K)-rungefactor/vol(i,j,k)*DW(5,I,J,K)
794                 60 CONTINUE
```

<MPI 병렬화 코드>

```
806                    do 60 k=max(kse(1,idk),2),min(kse(2,idk),kl)
808                    do 60 j=max(jse(1,idj),2),min(jse(2,idj),jl)
809                    DO 60 I=2,IL
810       127          W(1,I,J,K)   = Wn(1,I,J,K)-rungefactor/vol(i,j,k)*DW(1,I,J,K)
811        99          W(2,I,J,K)   = Wn(2,I,J,K)-rungefactor/vol(i,j,k)*DW(2,I,J,K)
812        72          W(3,I,J,K)   = Wn(3,I,J,K)-rungefactor/vol(i,j,k)*DW(3,I,J,K)
813        84          W(4,I,J,K)   = Wn(4,I,J,K)-rungefactor/vol(i,j,k)*DW(4,I,J,K)
814       261          W(5,I,J,K)   = Wn(5,I,J,K)-rungefactor/vol(i,j,k)*DW(5,I,J,K)
815                 60 CONTINUE
```

① 각 MPI rank에 대해 해당 영역의 값만 알고 있는 Wn, vol, DW array를 이용하여 해당 영역에 대해 W를 계산하게 된다.

② Nastok 루틴 전체적으로 보면 각 MPI rank에 대해 경계부분의 data 값을 이용하여 해당 영역의 W array를 계산하게 된다. 전 nastok 루틴 계산 후 다음 iteration의 nastok 루틴을 계산하기 위해서는 경계부분의 data 값을 통신을 통해 전달받아야 한다는 것이다.

E. RTRMS, RTMAX, IRT, JRT, KRT 계산 부분

<Serial 최적화 코드>

```
797                    RTRMS       = 0.
798                    RTMAX       = 0.
799                    NSUP        = 0
800                    DO 90 K=2,KL
801                    DO 90 J=2,JL
802                    DO 90 I=2,IL
803         1          RT          = dw(1,i,J,K)
804                    RTRMS       = RTRMS   +RT**2
805       271          IF (ABS(RT).LE.ABS(RTMAX)) GO TO 90
806                    RTMAX       = RT
807                    IRT         = I
808                    JRT         = J
809                    KRT         = K
810                 90 CONTINUE
```

```
811                    RTRMS      = SQRT(RTRMS/FLOAT((IL   -1)*(JL   -1)*(KL   -1)))
```

<MPI 병렬화 코드>

```
818                    RTRMS      = 0.
819                    RTMAX      = 0.
820                    NSUP       = 0
822                    do 90 k=max(kse(1,idk),2),min(kse(2,idk),kl)
824                    do 90 j=max(jse(1,idj),2),min(jse(2,idj),jl)
825                    DO 90 I=2,IL
826                    RT         = dw(1,i,J,K)
827         1          RTRMS      = RTRMS   +RT**2
828       132          IF (ABS(RT).LE.ABS(RTMAX)) GO TO 90
829                    RTMAX      = RT
830                    IRT        = I
831                    JRT        = J
832                    KRT        = K
833        54       90 CONTINUE
834              c
835                    CALL MPI_ALLREDUCE(RTRMS,RTRMSS,1,MPI_REAL8,
836                   & MPI_SUM,MPI_COMM_WORLD,ierr)
837                    RTRMS=RTRMSS
838
839                    CALL MPI_ALLGATHER(RTMAX,1,MPI_REAL8,buffproc,1,MPI_REAL8,
840                   & MPI_COMM_WORLD,ierr)
841                    kk0=id0
842                    RTMAX0=buffproc(kk0)
843                    DO 91 kk=1,nproc-1
844                    IF (ABS(buffproc(kk)).LE.ABS(RTMAX0)) GO TO 91
845                    kk0=kk
846                    RTMAX0=buffproc(kk)
847              91    CONTINUE
848                    if(id.eq.kk0) then
849                    ibuff(1)=IRT
850                    ibuff(2)=JRT
851                    ibuff(3)=KRT
852                    ENDIF
853                    isrce=kk0
854                    len=3
855                 CALL MPI_BCAST(IBUFF,len,MPI_INTEGER4,isrce,MPI_COMM_WORLD,ierr)
856                    IRT=ibuff(1)
857                    JRT=ibuff(2)
858                    KRT=ibuff(3)
859              c
860                    RTRMS      = SQRT(RTRMS/FLOAT((IL   -1)*(JL   -1)*(KL   -1)))
```

① 각 rank에 대해 해당 영역의 값만 알고 있는 dw array를 이용하여 RT, RTRMS를 계산하게 된다.

② 각 rank에 대한 RTRMS를 구한 다음 전 rank에서 MPI_ALLREDUCE 루틴의 MPI_SUM operator를 이용하여 이에 대한 합계를 구한 후 RTRMS에 저장한다.

③ RT의 최대값을 구하기 위해서 먼저 각 rank에서 RT의 최대값을 구한 후 이를 MPI_ALLGATHER 루틴을 이용하여 buffproc array로 저장한 다음 다시 buffproc

의 최대값을 찾아 RTMAX0를 구한다. 또한 이때의 buffproc array index를 이용하여 해당 rank를 찾아 내고 그에 해당하는 IRT, JRT, KRT 값을 ibuff arrary에 저장한 후 MPI_BCAST 루틴을 이용하여 전체 rank에 전달해 준다.

④ 최종적으로 전 rank에 대해 RTMAX0, IRT, JRT, KRT가 계산되게 된다.

3. nastokzone2, nastokzone3 루틴

nastok 루틴과 변수 이름 정도만 차이 날 뿐 거의 똑 같은 루틴이기 때문에 동일한 형식으로 수정하여 mpi 병렬화 하였다. 다음 변수들은 nastokzone2, nastokzone3oundturb 루틴에서 계산되는 변수들이다.

① nastokzone2 : dwz2, wz2, RTRMSz2

② nastokzone3 : dwz3, wz3, RTRMSz3

4. vsflux 루틴

A. visc array 계산 부분

<Serial 최적화 코드>

```
23                     do k = 1,k2
24                     do j = 1,j2
25                     do i = 1,i2
26        357          visc(i,j,k) = (t(i,j,k)/tfree)**vex/res
27                     enddo
28                     enddo
29                     enddo
```

<MPI 병렬화 코드>

```
34                     do k=max(kse(1,idk)-1,1),min(kse(2,idk)+1,k2)
36                     do j=max(jse(1,idj)-1,1),min(jse(2,idj)+1,j2)
37                     do i = 1,i2
38        139          visc(i,j,k) = (t(i,j,k)/tfree)**vex/res
39         67          enddo
40          1          enddo
41                     enddo
```

① visc array를 계산할 때 뒤에서 각 rank의 경계부분에 있는 값을 access하기 때문에 각 rank에 대한 계산 영역을 경계부분도 포함하도록 잡아서 do loop를 수행하고 있다.

B. dw array 계산 부분 – part 1

<Serial 최적화 코드>

```
 30                    do 100 k=2,kl
 31                    kp=k+1
 32                    km=k-1

 34          6         do 100 j=2,jl
 35                    jp=j+1
 36                    jm=j-1

 39                    do 10 i=1,il

195         44      10 continue
196                    do 11 i=2,il
197                    do ir=1,4
198        407          dw(ir+1,i,j,k)=dw(ir+1,i,j,k)
199                   *  -(fv(ir,i)-fv(ir,i-1))
200                    enddo
201                 11 continue
202         12     100 continue
```

<MPI 병렬화 코드>

```
 44                    do 100 k=max(kse(1,idk),2),min(kse(2,idk),kl)
 45                    kp=k+1
 46                    km=k-1

 49                    do 100 j=max(jse(1,idj),2),min(jse(2,idj),jl)
 50                    jp=j+1
 51                    jm=j-1

 54         58         do 10 i=1,il

210                 10 continue
211                    do 11 i=2,il
212                    do ir=1,4
213        380          dw(ir+1,i,j,k)=dw(ir+1,i,j,k)
214                   *  -(fv(ir,i)-fv(ir,i-1))
215                    enddo
216                 11 continue
217         10     100 continue
```

C. dw array 계산 부분 – part 2

<Serial 최적화 코드>

```
206                    do 200 k=2,kl
207                    kp=k+1
208                    km=k-1
```

```
216           3           do 20 j=1,jl
217                       jp=j+1
218                       do i=2,il

355          58             vco=(visc(i,j,k)*fac1+visc(i,jp,k)*fac2)/pr
356                      *    +(emu(i,j,k)*fac1+emu(i,jp,k)*fac2)/prt

358          64           dn =((visc(i,j,k) +emu(i,j,k))*fac1
359                      *    +(visc(i,jp,k)+emu(i,jp,k))*fac2)

362         492           fvj(1,i,j)=dn*(d1*ui+d5*vi+d6*wi+b1*uj+b5*vj+b7*wj
363                      *       +f1*uk+f7*vk+f9*wk) -2./3.*r1*tk1*sjx*vol22
364         416           fvj(2,i,j)=dn*(d7*ui+d2*vi+d8*wi+b5*uj+b2*vj+b6*wj
365                      *         +f5*uk+f2*vk+f10*wk) -2./3.*r1*tk1*sjy*vol22

376           1           enddo
377           3        20 continue

378                       do 21 j=2,jl
379                       do i=2,il
380                       do ir=1,4
381         465           dw(ir+1,i,j,k)=dw(ir+1,i,j,k)-(fvj(ir,i,j)-fvj(ir,i,j-1))
382                       enddo
383                       enddo
384                    21 continue

385                   200 continue
```

<MPI 병렬화 코드>

```
222                       do 200 k=max(kse(1,idk),2),min(kse(2,idk),kl)
223                       kp=k+1
224                       km=k-1

227                       do 20 j=max(jse(1,idj)-1,1),min(jse(2,idj),jl)
228                       jp=j+1
229          12           do i=2,il

366         158             vco=(visc(i,j,k)*fac1+visc(i,jp,k)*fac2)/pr
367                      *    +(emu(i,j,k)*fac1+emu(i,jp,k)*fac2)/prt

369          40           dn =((visc(i,j,k) +emu(i,j,k))*fac1
370                      *    +(visc(i,jp,k)+emu(i,jp,k))*fac2)
371                      *    *vol2/2.

373         460           fvj(1,i,j)=dn*(d1*ui+d5*vi+d6*wi+b1*uj+b5*vj+b7*wj
374                      *       +f1*uk+f7*vk+f9*wk) -2./3.*r1*tk1*sjx*vol22
375         254           fvj(2,i,j)=dn*(d7*ui+d2*vi+d8*wi+b5*uj+b2*vj+b6*wj
376                      *         +f5*uk+f2*vk+f10*wk) -2./3.*r1*tk1*sjy*vol22

387                       enddo
388           3        20 continue

390                       do 21 j=max(jse(1,idj),2),min(jse(2,idj),jl)
391                       do i=2,il
392                       do ir=1,4
393         457           dw(ir+1,i,j,k)=dw(ir+1,i,j,k)-(fvj(ir,i,j)-fvj(ir,i,j-1))
```

```
394                        enddo
395                        enddo
396          1          21 continue

397                    200 continue
```

D. dw array 계산 부분 – part 3

<Serial 최적화 코드>

```
398                        do 30 k=1,kl
399                        kp=k+1
400          5             do j=2,jl
401                        jp=j+1
402                        jm=j-1
403                        do i=2,il

540        180             vco=(visc(i,j,k)*fac1+visc(i,j,kp)*fac2)/pr
541                      *    +(emu(i,j,k)*fac1+emu(i,j,kp)*fac2)/prt

543         51             dn =((visc(i,j,k) +emu(i,j,k))*fac1+(visc(i,j,kp)
544                      *    +emu(i,j,kp))*fac2)*vol2/2.

546        435             fvk(1,i,j,k)=dn*(e1*ui+e5*vi+e6*wi+f1*uj+f5*vj+f6*wj
547                      *       +c1*uk+c5*vk+c7*wk) -2./3.*r1*tk1*skx*vol22
548        273             fvk(2,i,j,k)=dn*(e7*ui+e2*vi+e8*wi+f7*uj+f2*vj+f8*wj
549                      *         +c5*uk+c2*vk+c6*wk) -2./3.*r1*tk1*sky*vol22

559                        enddo
560          3             enddo
561                     30 continue

562                        do 31 k=2,kl
563                        do j=2,jl
564                        do i=2,il
565                        do ir=1,4
566        616             dw(ir+1,i,j,k)=dw(ir+1,i,j,k)-(fvk(ir,i,j,k)-fvk(ir,i,j,k-1))
567                        enddo
568                        enddo
569                        enddo
570                     31 continue
```

<MPI 병렬화 코드>

```
403                        do 30 k=max(kse(1,idk)-1,1),min(kse(2,idk),kl)
404                        kp=k+1
406                        do j=max(jse(1,idj),2),min(jse(2,idj),jl)
407                        jp=j+1
408                        jm=j-1
409         18             do i=2,il
410                        ip=i+1

546        452             vco=(visc(i,j,k)*fac1+visc(i,j,kp)*fac2)/pr
547                      *    +(emu(i,j,k)*fac1+emu(i,j,kp)*fac2)/prt
```

```
549      108        dn =((visc(i,j,k) +emu(i,j,k))*fac1+(visc(i,j,kp)
550                *    +emu(i,j,kp))*fac2)*vol2/2.

552      171        fvk(1,i,j,k)=dn*(e1*ui+e5*vi+e6*wi+f1*uj+f5*vj+f6*wj
553                *       +c1*uk+c5*vk+c7*wk) -2./3.*r1*tk1*skx*vol22
554      441        fvk(2,i,j,k)=dn*(e7*ui+e2*vi+e8*wi+f7*uj+f2*vj+f8*wj
555                *         +c5*uk+c2*vk+c6*wk) -2./3.*r1*tk1*sky*vol22

565                 enddo
566        7        enddo
567              30 continue

569                 do 31 k=max(kse(1,idk),2),min(kse(2,idk),kl)
571        1        do j=max(jse(1,idj),2),min(jse(2,idj),jl)
572                 do i=2,il
573                 do ir=1,4
574      552        dw(ir+1,i,j,k)=dw(ir+1,i,j,k)-(fvk(ir,i,j,k)-fvk(ir,i,j,k-1))
575                 enddo
576                 enddo
577                 enddo
578              31 continue
```

① dw array를 업데이트할 때 각각의 MPI rank가 해당 영역만 계산을 하도록 지정해 주었다.

② dw array를 계산하기 위해서 각각 fvk(*,*,j-1) 및 fvk(*,*,*,k-1)의 값을 알아야 하기 때문에 그 위의 do-loop에서 max(jse(1,idj)-1,1) 및 max(kse(1,idk)-1,1) 와 같이 구간을 확장 했으며 이 계산 및 jp=j+1, kp=k+1일때의 계산을 하기 위해서 앞의 visc array를 계산할 때 역시 동일하게 do-loop의 구간을 확장했다.

5. vsfluxz2, vsfluxz3 루틴

vsflux 루틴과 변수 이름 정도만 차이 날 뿐 거의 똑 같은 루틴이기 때문에 동일한 형식으로 수정하여 mpi 병렬화 하였다. 다음 변수들은 vsfluxz2, vsfluxz3 루틴에서 계산되는 변수들이다.

① vsfluxz2 : dwz2

② vsfluxz3 : dwz3

6. bound 루틴

A. w, ur, vr, wr, p, t array 계산 과정

<Serial 최적화 코드>

```
3                   do 100 k=2,kl
```

```
 4                       do 100 j=2,jl
 5                       do ir=1,5
 6          78            w(ir,i2,j,k)=w(ir,il,j,k)
 7                       enddo
 8                       ur(i2,j,k)=ur(il,j,k)
 9                       vr(i2,j,k)=vr(il,j,k)
10                       wr(i2,j,k)=wr(il,j,k)
11          14            p(i2,j,k)=p(il,j,k)
12          15            t(i2,j,k)=p(i2,j,k)/w(1,i2,j,k)
13                   100 continue

15                       do 300 j=2,jl
16                       do 300 i=1,i2
17                       do ir=1,5
18           2            w(ir,i,j,k2)=w(ir,i,j,kl)
19           5            w(ir,i,j,1)=w(ir,i,j,2)
20                       enddo
21           2            ur(i,j,k2)=ur(i,j,kl)
22           1            vr(i,j,k2)=vr(i,j,kl)
23           3            wr(i,j,k2)=wr(i,j,kl)
24                       t(i,j,k2)=t(i,j,kl)
25                       p(i,j,k2)=p(i,j,kl)
26                       ur(i,j,1)=ur(i,j,2)
27           5            vr(i,j,1)=vr(i,j,2)
28                       wr(i,j,1)=wr(i,j,2)
29           2            t(i,j,1)=t(i,j,2)
30           4            p(i,j,1)=p(i,j,2)
31                   300 continue

34                       do 200 k=1,k2
35                       do 200 i=1,i2
37                       do ir=1,5
38           9            w(ir,i,j2,k)=w(ir,i,jl,k)
39          11            w(ir,i,1,k)=w(ir,i,2,k)
40                       enddo
41           3            ur(i,j2,k)=ur(i,jl,k)
42           1            vr(i,j2,k)=vr(i,jl,k)
43           2            wr(i,j2,k)=wr(i,jl,k)
44                       t(i,j2,k)=t(i,jl,k)
45                       p(i,j2,k)=p(i,jl,k)
46           6            ur(i,1,k)=ur(i,2,k)
47           1            vr(i,1,k)=vr(i,2,k)
48           9            wr(i,1,k)=wr(i,2,k)
49           2            t(i,1,k)=t(i,2,k)
50                       p(i,1,k)=p(i,2,k)
51                   200 continue
```

<MPI 병렬화 코드>

```
19                       do 100 k=max(kse(1,idk)-2,2),min(kse(2,idk)+2,kl)
20           1            do 100 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
21                       do ir=1,5
22          49            w(ir,i2,j,k)=w(ir,il,j,k)
23                       enddo
24          32            ur(i2,j,k)=ur(il,j,k)
25          22            vr(i2,j,k)=vr(il,j,k)
26          13            wr(i2,j,k)=wr(il,j,k)
```

```
27          13               p(i2,j,k)=p(il,j,k)
28          11               t(i2,j,k)=p(i2,j,k)/w(1,i2,j,k)
29           2          100 continue

30                           if(idk.eq.0) then
32                           do 300 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
33                           do 300 i=1,i2
34                           do ir=1,5
35           7               w(ir,i,j,1)=w(ir,i,j,2)
36                           enddo
37           4               ur(i,j,1)=ur(i,j,2)
38           2               vr(i,j,1)=vr(i,j,2)
39                           wr(i,j,1)=wr(i,j,2)
40           2               t(i,j,1)=t(i,j,2)
41           1               p(i,j,1)=p(i,j,2)
42                      300 continue
43                           endif

44                           if(idk.eq.nprock-1) then
46                           do 309 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
47                           do 309 i=1,i2
48                           do ir=1,5
49           9               w(ir,i,j,k2)=w(ir,i,j,kl)
50                           enddo
51           3               ur(i,j,k2)=ur(i,j,kl)
52                           vr(i,j,k2)=vr(i,j,kl)
53           1               wr(i,j,k2)=wr(i,j,kl)
54           1               t(i,j,k2)=t(i,j,kl)
55                           p(i,j,k2)=p(i,j,kl)
56                      309 continue
57                           endif

60                           if(idj.eq.0) then
62                           do 200 k=max(kse(1,idk)-2,1),min(kse(2,idk)+2,k2)
63                           do 200 i=1,i2
65                           do ir=1,5
66          19               w(ir,i,1,k)=w(ir,i,2,k)
67                           enddo
68          10               ur(i,1,k)=ur(i,2,k)
69                           vr(i,1,k)=vr(i,2,k)
70           2               wr(i,1,k)=wr(i,2,k)
71                           t(i,1,k)=t(i,2,k)
72                           p(i,1,k)=p(i,2,k)
73                      200 continue
74                           endif
```

① w, ur, vr, wr, t, p array를 구할 때 각각의 MPI rank가 해당 영역만 계산을 하도록 지정해 주었다. 나중에 경계 부분의 값을 access하기 때문에 각 rank의 구간이 경계부분을 서로 중첩하여 포함하도록 do-loop를 수행하고 있다. 또한 k=1,k2 및 j=1 대해 계산할 때 idk 변수 및 idj 변수를 이용하여 이 영역을 포함하는 rank만 수행할 수 있도록 하였다.

B. Zone2에서 w, ur, vr, wr, p, t array 계산

<Serial 최적화 코드>

```
76              if(number_zone.ge.2) then
77              j=1
78              do 301 k=kszone2+1,kezone2
79              do 301 i=iszone2+1,iezone2
80              iz2=i-iszone2+1
81              kz2=k-kszone2+1
82     1        ur(i,j,k)=urz2(iz2,jlz2,kz2) ! need correction
83              vr(i,j,k)=vrz2(iz2,jlz2,kz2)
84              wr(i,j,k)=wrz2(iz2,jlz2,kz2)
85              p(i,j,k)=pz2(iz2,jlz2,kz2)
86              w(1,i,1,k)=wz2(1,iz2,jlz2,kz2)
87              w(2,i,1,k)=ur(i,j,k)*w(1,i,j,k)
88              w(3,i,1,k)=vr(i,j,k)*w(1,i,j,k)
89     1        w(4,i,1,k)=wr(i,j,k)*w(1,i,j,k)
90              w(5,i,1,k)=p(i,j,k)/gamm+0.5*w(1,i,j,k)*
91             *          (ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)
92              t(i,1,k)=p(i,1,k)/w(1,i,1,k)
93          301 continue
```

<MPI 병렬화 코드>

```
121             nr=5
122             if(number_zone.ge.2) then
125               call cast10(urz2,vrz2,wrz2,pz2,wz2,
126            &                 urz20,vrz20,wrz20,pz20,wz20,
127            & imxz2,jmxz2,kmxz2,i2z2,nproc,id,nprocj,nprock,idj,idk,
128            & jsez2,ksez2,jlz2,nprocj-1,nr)
129
130             j=1
131             if(idj.eq.0) then
133               do 301 k=max(kse(1,idk)-2,kszone2+1),min(kse(2,idk)+2,kezone2)
134             do 301 i=iszone2+1,iezone2
135             iz2=i-iszone2+1
136               kz2=k-kszone2+1
138             ur(i,j,k)=urz20(iz2,kz2) ! need correction
140             vr(i,j,k)=vrz20(iz2,kz2)
142             wr(i,j,k)=wrz20(iz2,kz2)
144             p(i,j,k)=pz20(iz2,kz2)
146             w(1,i,1,k)=wz20(iz2,kz2)
147             w(2,i,1,k)=ur(i,j,k)*w(1,i,j,k)
148             w(3,i,1,k)=vr(i,j,k)*w(1,i,j,k)
149             w(4,i,1,k)=wr(i,j,k)*w(1,i,j,k)
150    1        w(5,i,1,k)=p(i,j,k)/gamm+0.5*w(1,i,j,k)*
151            *          (ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)
152             t(i,1,k)=p(i,1,k)/w(1,i,1,k)
153         301 continue
154             endif
```

```
91              subroutine cast10(ur,vr,wr,p,w,
92             &                 ur0,vr0,wr0,p0,w0,
93             & imx,jmx,kmx,i2,nproc,id,nprocj,nprock,idj,idk,
94             & jse,kse,j,jproc,nr)
```

```
111                        DO kk=0,nprock-1
112                        ks=kse(1,kk)
113          2             ke=kse(2,kk)
114                        nk=ke-ks+1
116                        jj=jproc
119                        nj=1
120                        idd=jj+nprocj*kk
121
122                        IF(id.eq.idd) THEN
125                          I1=1
126                          do k=ks,ke
128                          do i=1,i2
129          5                 BUFF(I1)=ur(i,j,k)
130                            I1=I1+1
131          6                 BUFF(I1)=vr(i,j,k)
132                            I1=I1+1
133                            BUFF(I1)=wr(i,j,k)
134                            I1=I1+1
135          1                 BUFF(I1)=p(i,j,k)
136                            I1=I1+1
137          5                 BUFF(I1)=w(1,i,j,k)
138                            I1=I1+1
139                          enddo
140                          enddo
141                        ENDIF

142                        isrce=idd
143                        len=5*i2*nj*nk
144                        CALL MPI_BCAST(BUFF,len,MPI_REAL8,isrce,MPI_COMM_WORLD,ierr)
145                          I1=1
146                          do k=ks,ke
147                          do i=1,i2
148         98                 ur0(i,k)=BUFF(I1)
149          5                 I1=I1+1
150        131                 vr0(i,k)=BUFF(I1)
151         35                 I1=I1+1
152         62                 wr0(i,k)=BUFF(I1)
153          1                 I1=I1+1
154         47                 p0(i,k)=BUFF(I1)
155                            I1=I1+1
156         23                 w0(i,k)=BUFF(I1)
157         15                 I1=I1+1
158                          enddo
159          4               enddo
161          1             ENDDO
```

① 각 rank에 대해 영역이 분할되어 해당 영역 내의 값만 알고 있는 ur, vr, wr, p, w array에 포함하고 있는 않는 j=jlz2일 때의 값을 access하게 되기 때문에 MPI 코드에서는 cast10이라는 subroutine을 call하여 ur, vr, wr, p, w array에 대해서 j=jlz2일때의 값을 알고 있는 각 rank들이 전 프로세스에 broadcasting을 한 후 해당 연산을 수행하게 된다.

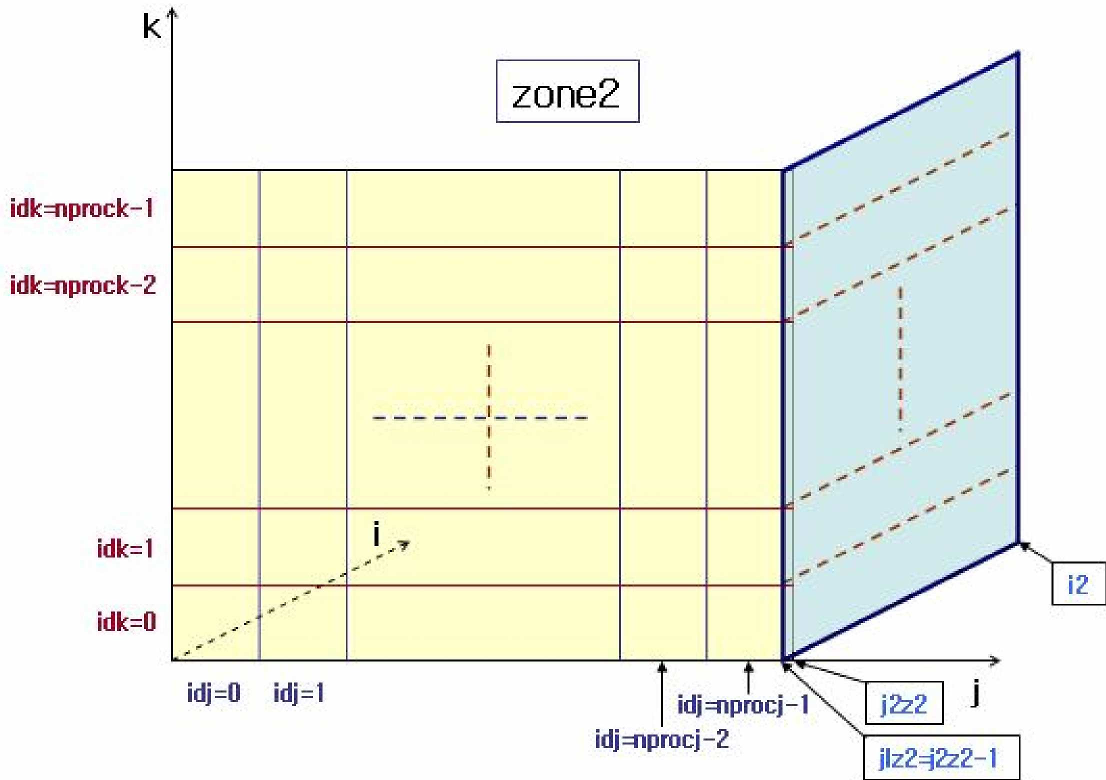


그림 I.4 Zone 2에서의 데이터 상호 교환도

C. Zone3에서 w, ur, vr, wr, p, t array 계산

<Serial 최적화 코드>

```
 95              do 302 k=kszone3+1,kezone3
 96              do 302 i=iszone3+1,iezone3
 97              iz3=i-iszone3+1
 98              kz3=k-kszone3+1
 99              ur(i,j,k)=urz3(iz3,jlz3,kz3) ! need correction
100              vr(i,j,k)=vrz3(iz3,jlz3,kz3)
101        1     wr(i,j,k)=wrz3(iz3,jlz3,kz3)
102              p(i,j,k)=pz3(iz3,jlz3,kz3)
103              w(1,i,1,k)=wz3(1,iz3,jlz3,kz3)
104              w(2,i,1,k)=ur(i,j,k)*w(1,i,j,k)
105              w(3,i,1,k)=vr(i,j,k)*w(1,i,j,k)
106        1     w(4,i,1,k)=wr(i,j,k)*w(1,i,j,k)
107              w(5,i,1,k)=p(i,j,k)/gamm+0.5*w(1,i,j,k)*
108             *        (ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)
109              t(i,1,k)=p(i,1,k)/w(1,i,1,k)
110          302 continue
```

<MPI 병렬화 코드>

```
158              call cast10(urz3,vrz3,wrz3,pz3,wz3,
```

```
159                              &                     urz30,vrz30,wrz30,pz30,wz30,
160                              & imxz3,jmxz3,kmxz3,i2z3,nproc,id,nprocj,nprock,idj,idk,
161                              & jsez3,ksez3,jlz3,nprocj-1,nr)
162                               if(idj.eq.0) then
164                                 do 302 k=max(kse(1,idk)-2,kszone3+1),min(kse(2,idk)+2,kezone3)
165                               do 302 i=iszone3+1,iezone3
166                               iz3=i-iszone3+1
167                                 kz3=k-kszone3+1
169                               ur(i,j,k)=urz30(iz3,kz3) ! need correction
171                               vr(i,j,k)=vrz30(iz3,kz3)
173                               wr(i,j,k)=wrz30(iz3,kz3)
175                               p(i,j,k)=pz30(iz3,kz3)
177                               w(1,i,1,k)=wz30(iz3,kz3)
178                               w(2,i,1,k)=ur(i,j,k)*w(1,i,j,k)
179                               w(3,i,1,k)=vr(i,j,k)*w(1,i,j,k)
180                               w(4,i,1,k)=wr(i,j,k)*w(1,i,j,k)
181                               w(5,i,1,k)=p(i,j,k)/gamm+0.5*w(1,i,j,k)*
182                              *          (ur(i,j,k)**2+vr(i,j,k)**2+wr(i,j,k)**2)
183                               t(i,1,k)=p(i,1,k)/w(1,i,1,k)
184                          302 continue
```

① Zone 2에서와 마찬가지로 각 rank에 대해 영역이 분할되어 해당 영역 내의 값만 알고 있는 ur, vr, wr, p, w array에 포함하고 있는 않는 j=jlz3일 때의 값을 access하게 되기 때문에 MPI 코드에서는 cast10 subroutine을 call하여 ur, vr, wr, p, w array에 대해서 j=jlz3일때의 값을 알고 있는 각 rank들이 전 프로세스에 broadcasting을 한 후 해당 연산을 수행하게 된다.

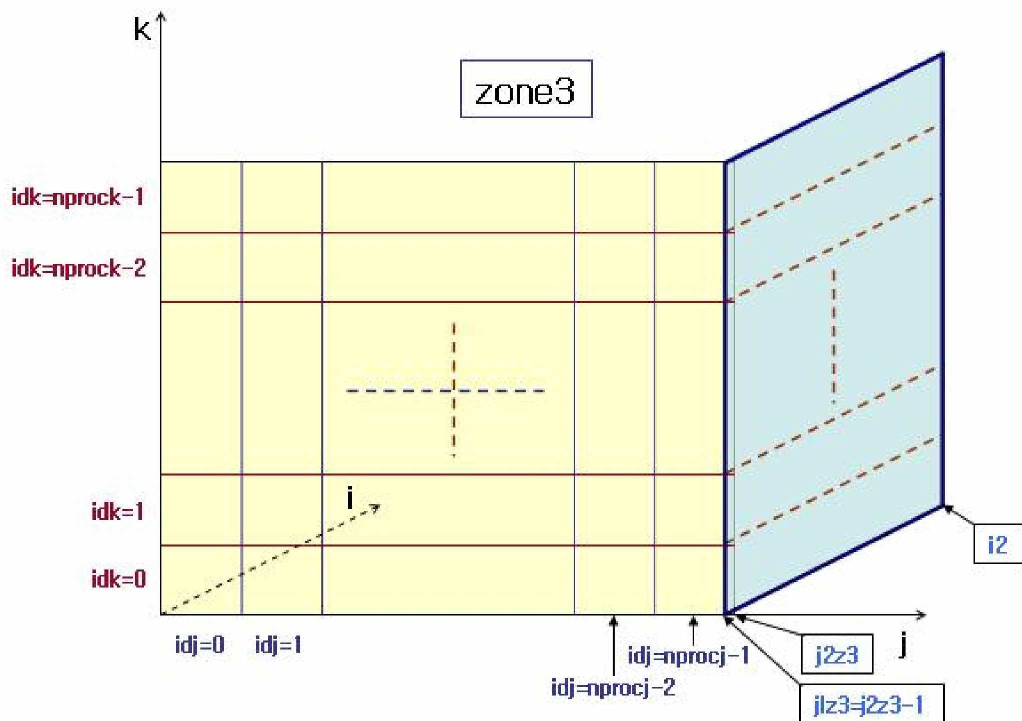


그림 I.5 Zone 3에서의 데이터 상호 교환도

7. boundturb, boundzone2, boundzone3, boundturbzone2, boundturbzone3 루틴

bound 루틴과 거의 비슷하게 구성되어 있기 때문에 동일한 형식으로 수정하여 mpi 병렬화 하였다. 다음 변수들은 boundturb, boundzone2, boundzone3, boundturbzone2, boundturbzone3 루틴에서 계산되는 변수들로 MPI 병렬화로 인해 각 rank에 대해서 해당 영역만 계산된다.

① boundturb : wtk, wtom, tk, tom, emu
② boundzone2 : wz2, urz2, vrz2, wrz2, pz2, tz2
③ boundzone3 : wz3, urz3, vrz3, wrz3, pz3, tz3
④ boundturbzone2 : wtkz2, twomz2, tkz2, tomz2, emuz2
⑤ boundturbzone3 : wrkz3, twomz3, tkz3, tomz3, emuz3

8. turhs 루틴

A. visc array 계산 부분

<Serial 최적화 코드>

```
102                     do   k = 1,k2
103                     do   j = 1,j2
104                     do   i = 1,i2
105        315           visc(i,j,k) = (t(i,j,k)/tfree)**vex/res
106                     enddo
107                     enddo
108                     enddo
```

<MPI 병렬화 코드>

```
158                     do k=max(kse(1,idk)-1,1),min(kse(2,idk)+1,k2)
160                     do j=max(jse(1,idj)-1,1),min(jse(2,idj)+1,j2)
161                     do   i = 1,i2
162        157           visc(i,j,k) = (t(i,j,k)/tfree)**vex/res
163         84           enddo
164          1           enddo
165                     enddo
```

① visc array를 계산할 때 뒤에서 각 rank의 경계부분에 있는 값을 access하기 때

문에 각 rank에 대한 계산 영역을 경계부분도 포함하도록 잡아서 do loop를 수행하고 있다.

B. tw array 계산 부분 – part 1

<Serial 최적화 코드>

```
109                     do 100 k=2,kl
110                     kp=k+1
111                     km=k-1
112           1         do 100 j=2,jl
113                     jp=j+1
114                     jm=j-1

117           1         do 10 i=1,il
118                     ip=i+1
119          20         vol2=vol(i,j,k)+vol(ip,j,k)

185         142      10 continue

316           1         do 14 i=2,il
317                     ip=i+1
318                     im=i-1
319                     do ir=1,2
320          52         tw(ir,i,j,k)=tw(ir,i,j,k)
321                    * -(fv(ir,i)-fv(ir,i-1))
322                     enddo
323                  14 continue

325           2     100 continue
```

<MPI 병렬화 코드>

```
167                     do 100 k=max(kse(1,idk),2),min(kse(2,idk),kl)
168                     kp=k+1
169                     km=k-1
171                     do 100 j=max(jse(1,idj),2),min(jse(2,idj),jl)
172                     jp=j+1
173                     jm=j-1

176          28         do 10 i=1,il
177                     ip=i+1
178          13         vol2=vol(i,j,k)+vol(ip,j,k)

244                  10 continue

375                     do 14 i=2,il
376                     ip=i+1
377                     im=i-1
378                     do ir=1,2
379          70         tw(ir,i,j,k)=tw(ir,i,j,k)
380                    * -(fv(ir,i)-fv(ir,i-1))
381                     enddo
382           1      14 continue
```

```
384          3        100 continue
```

C. tw array 계산 부분 – part 2

<Serial 최적화 코드>

```
328                     do 200 k=2,kl
329                     kp=k+1
330                     km=k-1
332          3          do 20 j=1,jl
333                     jp=j+1
334                     do i=2,il
335                     ip=i+1
336                     im=i-1
338         77          vol2=vol(i,j,k)+vol(i,jp,k)

356        548          vise=((visc(i,j,k)+emu(i,j,k)*sigma_tko)*fac1
357                    *      +(visc(i,jp,k)+emu(i,jp,k)*sigma_tko)*fac2)*vol2/2.
358         66          visk=((visc(i,j,k)+emu(i,j,k)*sigmastar_tko)*fac1
359                    *      +(visc(i,jp,k)+emu(i,jp,k)*sigmastar_tko)*fac2)*vol2/2.

404                     fvj(1,i,j)=visk*(b1*tki+b2*tkj+b3*tkk)
405        569          fvj(2,i,j)=vise*(b1*tei+b2*tej+b3*tek)
406         67          enddo

408          2        20 continue

535                     do 24 j=2,jl
536                     jp=j+1
537                     jm=j-1
538                     do i=2,il
539                     do ir=1,2
540        147          tw(ir,i,j,k)=tw(ir,i,j,k)
541                    * -(fvj(ir,i,j)-fvj(ir,i,j-1))
542                     enddo
543                     enddo
544                   24 continue

546                  200 continue
```

<MPI 병렬화 코드>

```
388          1          do 200 k=max(kse(1,idk),2),min(kse(2,idk),kl)
389                     kp=k+1
390                     km=k-1
393                     do 20 j=max(jse(1,idj)-1,1),min(jse(2,idj),jl)
395                     jp=j+1
396         11          do i=2,il
397                     ip=i+1
398                     im=i-1
400                     vol2=vol(i,j,k)+vol(i,jp,k)

418        246          vise=((visc(i,j,k)+emu(i,j,k)*sigma_tko)*fac1
419                    *      +(visc(i,jp,k)+emu(i,jp,k)*sigma_tko)*fac2)*vol2/2.
```

```
420        46          visk=((visc(i,j,k)+emu(i,j,k)*sigmastar_tko)*fac1
421                   *      +(visc(i,jp,k)+emu(i,jp,k)*sigmastar_tko)*fac2)*vol2/2.

466       430          fvj(1,i,j)=visk*(b1*tki+b2*tkj+b3*tkk)
467        46          fvj(2,i,j)=vise*(b1*tei+b2*tej+b3*tek)
468                    enddo

470                 20 continue

601                    do 24 j=max(jse(1,idj),2),min(jse(2,idj),jl)
602                    jp=j+1
603                    jm=j-1
604                    do i=2,il
605                    do ir=1,2
606       110          tw(ir,i,j,k)=tw(ir,i,j,k)
607                   * -(fvj(ir,i,j)-fvj(ir,i,j-1))
608                    enddo
609                    enddo
610         1       24 continue

612                200 continue
```

① tw array를 계산하기 위해서 각각 fvk(*,*,j-1)의 값을 알아야 하기 때문에 그 위의 do-loop에서 max(jse(1,idj)-1,1)와 같이 구간을 확장 했으며 이 계산 및 jp=j+1일때의 계산을 하기 위해서 앞의 visc array를 계산할 때 역시 동일하게 do-loop의 구간을 확장했다.

D. tw array 계산 부분 – part 3

<Serial 최적화 코드>

```
559                    do 30 k=1,kl
560                    kp=k+1
561         5          do i=2,il
562                    ip=i+1
563                    im=i-1
564                    do j=2,jl
565                    jp=j+1
566                    jm=j-1

568        14          vol2=vol(i,j,k)+vol(i,j,kp)

586       171          vise=((visc(i,j,k)+emu(i,j,k)*sigma_tko)*fac1
587                   *      +(visc(i,j,kp)+emu(i,j,kp)*sigma_tko)*fac2)*vol2/2.
588        77          visk=((visc(i,j,k)+emu(i,j,k)*sigmastar_tko)*fac1
589                   *      +(visc(i,j,kp)+emu(i,j,kp)*sigmastar_tko)*fac2)*vol2/2.

634       769          fvk(1,i,j,k)=visk*(c1*tki+c2*tkj+c3*tkk)
635                    fvk(2,i,j,k)=vise*(c1*tei+c2*tej+c3*tek)
636                    enddo
637        10          enddo

639                 30 continue
```

```
641                    do 35 k=1,k2
642                    do j=2,jl
643                    do i=2,il
644        528           uu=ur(i,j,k)*ztx(i,j,k)+vr(i,j,k)*zty(i,j,k)
645                  *    +wr(i,j,k)*ztz(i,j,k)

664        169          tck(1,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tk(i,j,k)*ctp
665        155          tck(2,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tom(i,j,k)*ctp
666         73          tcmk(1,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tk(i,j,k)*ctm
667          2          tcmk(2,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tom(i,j,k)*ctm

670                    enddo
671          1         enddo
672                 35 continue
673                     do 31 k=2,kl
674                     l=k-1
675                    do j=2,jl
676                    do i=2,il
677                     if((k.eq.2) .or. (k.eq.kl)) then
678                     do ir=1,2
679         13            tw(ir,i,j,k)=tw(ir,i,j,k)+(tck(ir,i,j,k)-tck(ir,i,j,k-1))
680                     enddo
681                     else
682                       do ir=1,2

713        164           tw(ir,i,j,k)=tw(ir,i,j,k)
714                  * +(1.+0.5*p1)*(tck(ir,i,j,k)-tck(ir,i,j,l))
715                  * -0.5*p2*(tck(ir,i,j,l)-tck(ir,i,j,l-1))
716                     enddo
717                     endif
718                    enddo
719                    enddo
720                 31   continue

722                     do 33 k=2,kl
723                     l=k+1
724                    do j=2,jl
725                    do i=2,il
726          1          if((k.eq.2) .or. (k.eq.kl)) then
727                     do ir=1,2
728         14            tw(ir,i,j,k)=tw(ir,i,j,k)+(tcmk(ir,i,j,l)-tcmk(ir,i,j,k))
729                     enddo
730                     else
731                       do ir=1,2

763        149           tw(ir,i,j,k)=tw(ir,i,j,k)+
764                  * (1.+0.5*p1)*(tcmk(ir,i,j,l)-tcmk(ir,i,j,k))
765                  * -0.5*p2*(tcmk(ir,i,j,l+1)-tcmk(ir,i,j,l))
766                     enddo
767                    endif
768                    enddo
769                    enddo
770                 33 continue

772                    do 34 k=2,kl
773                    kp=k+1
774                    km=k-1
775                    do j=2,jl
776                    do i=2,il
777                    do ir=1,2
```

```
778         353          tw(ir,i,j,k)=tw(ir,i,j,k)
779                     * -(fvk(ir,i,j,k)-fvk(ir,i,j,k-1))
780                      enddo
781                      enddo
782                      enddo
783                   34 continue
```

<MPI 병렬화 코드>

```
618                      do 30 k=max(kse(1,idk)-1,1),min(kse(2,idk),kl)
619                      kp=k+1
621                      do j=max(jse(1,idj),2),min(jse(2,idj),jl)
622                      jp=j+1
623                      jm=j-1
624           5          do i=2,il
625                      ip=i+1
626                      im=i-1

646         391          vise=((visc(i,j,k)+emu(i,j,k)*sigma_tko)*fac1
647                     *       +(visc(i,j,kp)+emu(i,j,kp)*sigma_tko)*fac2)*vol2/2.
648         224          visk=((visc(i,j,k)+emu(i,j,k)*sigmastar_tko)*fac1
649                     *       +(visc(i,j,kp)+emu(i,j,kp)*sigmastar_tko)*fac2)*vol2/2.

694         530          fvk(1,i,j,k)=visk*(c1*tki+c2*tkj+c3*tkk)
695          14          fvk(2,i,j,k)=vise*(c1*tei+c2*tej+c3*tek)
696                      enddo
697           8          enddo
698
699                   30 continue

702           1          do 35 k=max(kse(1,idk)-2,1),min(kse(2,idk)+2,k2)
704                      do j=max(jse(1,idj),2),min(jse(2,idj),jl)
705                      do i=2,il
706         806            uu=ur(i,j,k)*ztx(i,j,k)+vr(i,j,k)*zty(i,j,k)
707                     *    +wr(i,j,k)*ztz(i,j,k)

726         383          tck(1,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tk(i,j,k)*ctp
727          78          tck(2,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tom(i,j,k)*ctp
728         398          tcmk(1,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tk(i,j,k)*ctm
729         109          tcmk(2,i,j,k)=vol(i,j,k)*w(1,i,j,k)*tom(i,j,k)*ctm

732                      enddo
733                      enddo
734                   35 continue

736                        do 31 k=max(kse(1,idk),2),min(kse(2,idk),kl)
737                        l=k-1
739                        do j=max(jse(1,idj),2),min(jse(2,idj),jl)
740                      do i=2,il
741                        if((k.eq.2) .or. (k.eq.kl)) then
742                        do ir=1,2
743          12              tw(ir,i,j,k)=tw(ir,i,j,k)+(tck(ir,i,j,k)-tck(ir,i,j,k-1))
744                        enddo
745                        else
746                          do ir=1,2

777         252           tw(ir,i,j,k)=tw(ir,i,j,k)
```

```
778                          * +(1.+0.5*p1)*(tck(ir,i,j,k)-tck(ir,i,j,l))
779                          * -0.5*p2*(tck(ir,i,j,l)-tck(ir,i,j,l-1))
780                            enddo
781                            endif
782                           enddo
783                           enddo
784                       31   continue

787                            do 33 k=max(kse(1,idk),2),min(kse(2,idk),kl)
788                            l=k+1
790                            do j=max(jse(1,idj),2),min(jse(2,idj),jl)
791                           do i=2,il
792          16                 if((k.eq.2) .or. (k.eq.kl)) then
793                            do ir=1,2
794           1                   tw(ir,i,j,k)=tw(ir,i,j,k)+(tcmk(ir,i,j,l)-tcmk(ir,i,j,k))
795                            enddo
796                            else
797                              do ir=1,2

829         137                 tw(ir,i,j,k)=tw(ir,i,j,k)+
830                          * (1.+0.5*p1)*(tcmk(ir,i,j,l)-tcmk(ir,i,j,k))
831                          * -0.5*p2*(tcmk(ir,i,j,l+1)-tcmk(ir,i,j,l))
832                            enddo
833                           endif
834                           enddo
835           1               enddo
836                       33 continue

839                           do 34 k=max(kse(1,idk),2),min(kse(2,idk),kl)
840                           kp=k+1
841                           km=k-1
843                           do j=max(jse(1,idj),2),min(jse(2,idj),jl)
844                           do i=2,il
845                           do ir=1,2
846         326                tw(ir,i,j,k)=tw(ir,i,j,k)
847                          * -(fvk(ir,i,j,k)-fvk(ir,i,j,k-1))
848                           enddo
849                           enddo
850           1               enddo
851                       34 continue
```

① tw array를 업데이트할 때 각각의 MPI rank가 해당 영역만 계산을 하도록 지정해 주었다.

② 즉 tw array를 업데이트할 때 fvk(*,*,*,k-1)의 값을 알아야 하기 때문에 위의 fvk를 계산하는 do-loop에서 max(kse(1,idk)-1,1)와 같이 구간을 확장 했으며 이 계산 및 kp=k+1일때의 계산을 하기 위해서 앞의 visc array를 계산할 때 역시 동일하게 do-loop의 구간을 확장했다.

③ 또한 l=k-1일 때의 tck(*,*,*,l-1)의 값 및 l=k+1일 때의 tcmk(*,*,*,l+1)의 값을 계산하기 위해서 max(kse(1,idk)-2,1),min(kse(2,idk)+2,k2) 와 같이 do-loop의 구간을 확장했다.

E. tw array 계산 부분 – part 4

<Serial 최적화 코드>

```
798                       do 400 k=2,kl
799                       kp=k+1
800                       km=k-1
801                       do 400 j=2,jl
802                       jp=j+1
803                       jm=j-1
804           2           do 400 i=2,il
805                       ip=i+1
806                       im=i-1

929          72           tw(1,i,j,k)=-tw(1,i,j,k)+stk*vol(i,j,k)
930          45           tw(2,i,j,k)=-tw(2,i,j,k)+ste*vol(i,j,k)
931         289       400 continue
```

<MPI 병렬화 코드>

```
867                       do 400 k=max(kse(1,idk),2),min(kse(2,idk),kl)
868                       kp=k+1
869                       km=k-1
871           1           do 400 j=max(jse(1,idj),2),min(jse(2,idj),jl)
872                       jp=j+1
873                       jm=j-1
874                       do 400 i=2,il
875                       ip=i+1
876                       im=i-1

999         196           tw(1,i,j,k)=-tw(1,i,j,k)+stk*vol(i,j,k)
1000        162           tw(2,i,j,k)=-tw(2,i,j,k)+ste*vol(i,j,k)
1001          6       400 continue
```

① tw array를 업데이트할 때 각각의 MPI rank가 해당 영역만 계산을 하도록 지정해 주었다.

F. dr1, dr2, tkm, ikm, jkm, kkm, tem, iem, jem, kem 계산 부분

<Serial 최적화 코드>

```
933                       dr1=1.e-15
934                         dr2=1.e-15
935                        tkm=0
936                        tem=0
937                       do 410 i=2,il
938           1              do 410 j=2,jl
939                            do 410 k=2,kl
940           5             dr1=dr1+tw(1,i,j,k)**2
941          15             dr2=dr2+tw(2,i,j,k)**2
942        2357        if(abs(tw(1,i,j,k)).gt.tkm) then
943                       tkm=abs(tw(1,i,j,k))
```

```
944                           ikm=i
945                           jkm=j
946                           kkm=k
947            1              endif
948                           if(abs(tw(2,i,j,k)).gt.tem) then
949                           tem=abs(tw(2,i,j,k))
950                           iem=i
951                           jem=j
952                           kem=k
953                           endif
954
955                       410 continue
956                           dr1=sqrt(dr1)/float((il-1)*(jl-1)*(kl-1))
957                             dr2=sqrt(dr2)/float((il-1)*(jl-1)*(kl-1))
958                             dr1=alog10(dr1)
959                             dr2=alog10(dr2)
960                             write(*,*) it,dr1,dr2,tkm,tem
```

<MPI 병렬화 코드>

```
1005                          dr(1)=0.
1006                            dr(2)=0.
1007                           tkm=0
1008                           tem=0
1010                          do 410 k=max(kse(1,idk),2),min(kse(2,idk),kl)
1012                          do 410 j=max(jse(1,idj),2),min(jse(2,idj),jl)
1013                          do 410 i=2,il
1014           3                dr(1)=dr(1)+tw(1,i,j,k)**2
1015           8                dr(2)=dr(2)+tw(2,i,j,k)**2
1016          24              if(abs(tw(1,i,j,k)).gt.tkm) then
1017                          tkm=abs(tw(1,i,j,k))
1018                          ikm=i
1019                          jkm=j
1020                          kkm=k
1021                          endif
1022          51              if(abs(tw(2,i,j,k)).gt.tem) then
1023                          tem=abs(tw(2,i,j,k))
1024                          iem=i
1025                          jem=j
1026                          kem=k
1027                          endif
1028
1029          58          410 continue
1030                    c
1031                          CALL MPI_ALLREDUCE(dr,drs,2,MPI_REAL8,
1032                         & MPI_SUM,MPI_COMM_WORLD,ierr)
1033                          dr(1)=drs(1)
1034                          dr(2)=drs(2)
1035                          tkme(1)=tkm
1036                          tkme(2)=tem
1037                          CALL MPI_ALLGATHER(tkme,2,MPI_REAL8,buffproc,2,MPI_REAL8,
1038                         & MPI_COMM_WORLD,ierr)
1039                          kk1=id0
1040                          kk2=id0
1041                          tkm=buffproc(1,kk1)
1042                          tem=buffproc(2,kk2)
1043                          DO kk=1,nproc-1
```

```
1044                              if(abs(buffproc(1,kk)).gt.tkm) then
1045                              tkm=abs(buffproc(1,kk))
1046                              kk1=kk
1047                              endif
1048                              if(abs(buffproc(2,kk)).gt.tem) then
1049                              tem=abs(buffproc(2,kk))
1050                              kk2=kk
1051                              endif
1052                              enddo
1053                              if(id.eq.kk1) then
1054                              ibuff(1)=ikm
1055                              ibuff(2)=jkm
1056                              ibuff(3)=kkm
1057                              ENDIF
1058                              isrce=kk1
1059                              len=3
1060                          CALL MPI_BCAST(IBUFF,len,MPI_INTEGER4,isrce,MPI_COMM_WORLD,ierr)
1061                              ikm=ibuff(1)
1062                              jkm=ibuff(2)
1063                              kkm=ibuff(3)
1064                              if(id.eq.kk2) then
1065                              ibuff(1)=iem
1066                              ibuff(2)=jem
1067                              ibuff(3)=kem
1068                              ENDIF
1069                              isrce=kk2
1070                              len=3
1071                          CALL MPI_BCAST(IBUFF,len,MPI_INTEGER4,isrce,MPI_COMM_WORLD,ierr)
1072                              iem=ibuff(1)
1073                              jem=ibuff(2)
1074                              kem=ibuff(3)
1075
1076                              dr(1)=dr(1)+1.e-15
1077                                  dr(2)=dr(2)+1.e-15
1078                              dr(1)=sqrt(dr(1))/float((il-1)*(jl-1)*(kl-1))
1079                                dr(2)=sqrt(dr(2))/float((il-1)*(jl-1)*(kl-1))
1080                                dr(1)=alog10(dr(1))
1081                                dr(2)=alog10(dr(2))
1082                              if(id.eq.id0) then
1083                                write(*,*) it,dr(1),dr(2),tkm,tem
1084                              endif
```

① 각 rank에 대해 해당 영역의 값만 알고 있는 tw array를 이용하여 dr array를 계산한다.

② 각 rank에 대한 dr array를 구한 다음 전 rank에서 MPI_ALLREDUCE 루틴의 MPI_SUM operator를 이용하여 이에 대한 합계를 구한 후 drs array에 저장한다.

③ tw(1,i,j,k) 및 tw(2,i,j,k) 의 최대값을 구하기 위해서 먼저 각 rank에서 tw(1,i,j,k) 및 tw(2,i,j,k)의 최대값 tkm 및 tem을 구한 후 이를 tkme array에 저장한다. 그런 후 tkme array를 MPI_ALLGATHER 루틴을 이용하여 buffproc array로 저장한 다음 먼저 buffproc(1,*)의 최대값을 찾아 tkm을 구하고 buffproc(2,*)의 최대값을 찾아 tem을 구한다. 또한 이때의 buffproc array index를 이용하여 해당 rank 즉 kk1, kk2를 찾아 내고 그에 해당하는 ikm, jkm,kkm 및 iem, jem, kem 값을 각각

ibuff arrary에 저장한 후 MPI_BCAST 루틴을 이용하여 전체 rank에 전달해 준다.

④ 최종적으로 전 rank에 대해 ikm, jkm, kkm 및 iem, jem, kem이 계산되게 된다.

9. turhszone2, turhszone3 루틴

turhs 루틴과 변수 이름 정도만 차이 날 뿐 거의 똑 같은 루틴이기 때문에 동일한 형식으로 수정하여 mpi 병렬화 하였다. 다음 변수들은 turhszone2, turhszone3 루틴에서 계산되는 변수들이다.

① turhszone2 : twz2

② turhszone3 : twz3

10. turbkom 루틴

A. wtk, wtom, tk, tom, emu array 계산 부분

<Serial 최적화 코드>

```
20                  do 600 k=2,kl
21        16        do 600 j=2,jl
22                  do 600 i=2,il
23       183        wtk(i,j,k)=wtkn(i,j,k)
24                 *          +rungefactor*tw(1,i,j,k)/vol(i,j,k)
25       315        wtom(i,j,k)=wtomn(i,j,k)
26                 *          +rungefactor*tw(2,i,j,k)/vol(i,j,k)

31        16        tk(i,j,k)=wtk(i,j,k)/w(1,i,j,k)
32       154        tom(i,j,k)=wtom(i,j,k)/w(1,i,j,k)
33       224        if(tom(i,j,k).lt.tomfree) then
34         4        tom(i,j,k)=tomfree
35                  wtom(i,j,k)=w(1,I,J,K)*tom(i,j,k)
36                  endif

42         8        emu(i,j,k)=tk(i,j,k)/tom(i,j,k)
43       354        if(emu(i,j,k).gt.emumax) then
44                  emu(i,j,k)=emumax
45                  tk(i,j,k)=emu(i,j,k)*tom(i,j,k)
46                  wtk(i,j,k)=tk(i,j,k)*w(1,i,j,k)
47                  endif

66              600 continue
```

<MPI 병렬화 코드>

```
23                  ks=kse(1,idk)
24                  ke=kse(2,idk)
25                  js=jse(1,idj)
```

```
26                      je=jse(2,idj)
27                      call mpi_com_pm2_2(tw,imx,jmx,kmx,i2,k2,
28                     & nprocj,nprock,idj,idk,js,je,ks,ke)

32                      do 600 k=max(kse(1,idk)-2,2),min(kse(2,idk)+2,kl)
33                      do 600 j=max(jse(1,idj)-2,2),min(jse(2,idj)+2,jl)
34                      do 600 i=2,il
35        178           wtk(i,j,k)=wtkn(i,j,k)
36                     *           +rungefactor*tw(1,i,j,k)/vol(i,j,k)
37        192           wtom(i,j,k)=wtomn(i,j,k)
38                     *           +rungefactor*tw(2,i,j,k)/vol(i,j,k)
39        317           tk(i,j,k)=wtk(i,j,k)/w(1,i,j,k)
40        303           tom(i,j,k)=wtom(i,j,k)/w(1,i,j,k)
41         61           if(tom(i,j,k).lt.tomfree) then
42          2           tom(i,j,k)=tomfree
43         30           wtom(i,j,k)=w(1,I,J,K)*tom(i,j,k)
44                      endif
45        127           emu(i,j,k)=tk(i,j,k)/tom(i,j,k)
46        141           if(emu(i,j,k).gt.emumax) then
47                      emu(i,j,k)=emumax
48          1           tk(i,j,k)=emu(i,j,k)*tom(i,j,k)
49        148           wtk(i,j,k)=tk(i,j,k)*w(1,i,j,k)
50                      endif
51                  600 continue
```

① turhs 루틴을 지나면 각 rank에 해당하는 영역 내의 tw array 값만 알게 되는데, 다음에 오는 계산과정에서 경계 부근의 tw array를 access하기 때문에 mpi_comm_pm2_2 루틴을 call하여 경계 부근의 data를 미리 전달해 준다.

② 각각의 array를 계산할 때에는 각 MPI rank가 해당 영역만 계산하도록 지정해 주었다.

B. tkm, ikm, jkm, kkm, tem, iem, jem, kem 계산 부분

<Serial 최적화 코드>

```
18                       tkm=0
19                       tem=0
20                      do 600 k=2,kl
21         16           do 600 j=2,jl
22                      do 600 i=2,il

53          9           if(tk(i,j,k).gt.tkm) then
54                      tkm=tk(i,j,k)
55                      ikm=i
56                      jkm=j
57                      kkm=k
58                      endif
59         34           if(tom(i,j,k).gt.tem) then
60                      tem=tom(i,j,k)
61                      iem=i
62                      jem=j
63                      kem=k
64                      endif
```

```
66                600 continue
```

<MPI 병렬화 코드>

```
74            tkme(1)=tkm
75            tkme(2)=tem
76            CALL MPI_ALLGATHER(tkme,2,MPI_REAL8,buffproc,2,MPI_REAL8,
77           & MPI_COMM_WORLD,ierr)
78            kk1=id0
79            kk2=id0
80            tkm=buffproc(1,kk1)
81            tem=buffproc(2,kk2)
82            DO kk=1,nproc-1
83            if(buffproc(1,kk).gt.tkm) then
84            tkm=buffproc(1,kk)
85            kk1=kk
86            endif
87            if(buffproc(2,kk).gt.tem) then
88            tem=buffproc(2,kk)
89            kk2=kk
90            endif
91            enddo
92            if(id.eq.kk1) then
93            ibuff(1)=ikm
94            ibuff(2)=jkm
95            ibuff(3)=kkm
96            ENDIF
97            isrce=kk1
98            len=3
99         CALL MPI_BCAST(IBUFF,len,MPI_INTEGER4,isrce,MPI_COMM_WORLD,ierr)
100           ikm=ibuff(1)
101           jkm=ibuff(2)
102           kkm=ibuff(3)
103           if(id.eq.kk2) then
104           ibuff(1)=iem
105           ibuff(2)=jem
106           ibuff(3)=kem
107           ENDIF
108           isrce=kk2
109           len=3
110        CALL MPI_BCAST(IBUFF,len,MPI_INTEGER4,isrce,MPI_COMM_WORLD,ierr)
111           iem=ibuff(1)
112           jem=ibuff(2)
113           kem=ibuff(3)
```

① turhs 루틴과 동일한 방법으로 각각의 변수들이 계산된다.

11. turbkomzone2, turbkomzone3 루틴

turbkom 루틴과 변수 이름 정도만 차이 날 뿐 거의 똑 같은 루틴이기 때문에 동일한 형식으로 수정하여 mpi 병렬화 하였다. 다음 변수들은 turbkomzone2, turbkomzone3 루틴

에서 계산되는 변수들이다.

① turbkomzone2 : wtkz2, wtomz2, tkz2, tomz2, emuz2

② turbkomzone3 : wtkz3, wtomz3, tkz3, tomz3, emuz3

12. outputtest 루틴

A. 0 rank로 data 전송 후 결과를 write하는 부분

<Serial 최적화 코드>

```
  4          tmp=1
  5          open(17, file='flowxy.dat')
  6          open(18, file='flowxz.dat')
  7          open(19, file='flowzy.dat')
  8
  9          write(17,11)
 10          write(17,*) 'variables= x, y, u, v, w, p, r, tk, tom'
 11          write(17,2144) il,jl
 12          k=kk
 13          do 30 j=1,jl
 14          do 30 i=1,il
 15          write(17,101) x(1,i,j,k),x(2,i,j,k),ur(i,j,k),vr(i,j,k),wr(i,j,k)
 16         *     ,p(i,j,k),w(1,i,j,k),tk(i,j,k),tom(i,j,k)
 17       30 continue
```

<MPI 병렬화 코드>

```
  7          call mpi_com_output(w,ur,vr,wr,p,wtk,wtom,tk,tom,
  8         & imx,jmx,kmx,i2,nproc,id,nprocj,nprock,idj,idk,
  9         & jse,kse,maxmpi1)

 12          if(id.eq.0) then
 13          tmp=1
 14          open(17, file='flowxy.dat')
 15          open(18, file='flowxz.dat')
 16          open(19, file='flowzy.dat')
 17          write(17,11)
 18          write(17,*) 'variables= x, y, u, v, w, p, r, tk, tom'
 19          write(17,2144) il,jl
 20          k=kk
 21          do 30 j=1,jl
 22          do 30 i=1,il
 23          write(17,101) x(1,i,j,k),x(2,i,j,k),ur(i,j,k),vr(i,j,k),wr(i,j,k)
 24         *     ,p(i,j,k),w(1,i,j,k),tk(i,j,k),tom(i,j,k)
 25       30 continue

123          endif
```

① w, ur, vr, p, wtk, wtom, tk, tom array에 대해 각각의 MPI rank가 알고 있는 영역

의 값을 mpi_com_output 루틴을 이용하여 0 rank로 보낸 다음 0 rank에서 이를 write하는 부분이다.

13. outputtestz2, outputtestz3 루틴

outputtest 루틴과 변수 이름만 차이날 뿐 거의 똑 같은 루틴이기 때문에 동일한 형식으로 수정하여 mpi 병렬화 하였다.

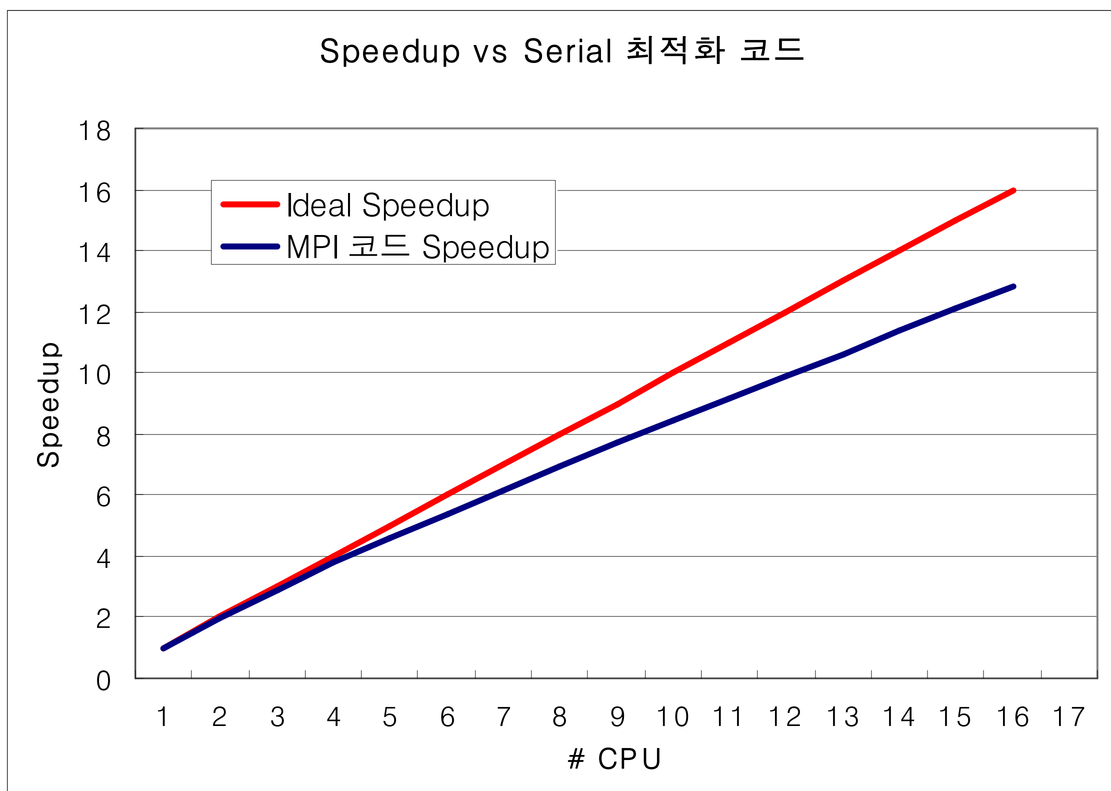
9. MPI 병렬화 결과

K 및 j 방향으로 모두 MPI 병렬화를 진행하다보니 array data를 다룰 때 조금 복잡하게 구현된 부분이 있었으며, speedup은 전반적으로 괜찮은 성능을 보인다.

IBM p690+ 1.7GHz 시스템에서 MPI 병렬화 코드를 수행한 결과는 다음과 같다.

# Tasks	Original 코드	Serial 최적화 코드	MPI 코드	Speedup vs Serial 최적화 코드	Speedup vs Original 코드
1	164.84	58.12	-	-	-
2	-	-	29.81	1.95	5.53
4	-	-	15.42	3.77	10.69
8	-	-	8.34	6.97	19.74
16	-	-	4.53	12.83	36.39

단위 : sec / Iteration



10. Summary

전반적으로 볼 때 serial 최적화의 내용은 사용자 코드가 do loop를 이용하여 다차원 배열을 access할 때 많은 cache miss를 일으키도록 작성되어 있는 부분을 cache miss를 일으키지 않도록 do-loop의 순서를 수정하여 최적화하였다.

또한 MPI 병렬화는 2차원으로 영역분할을 하면서 이 부분 조금 복잡하게 구현이 되었으며 추가적으로 몇 개의 subroutine이 더해졌다. 대체적으로 통신이 복잡하게 일어나는 알고리즘이 아니라서 각 rank에 대해 해당영역 및 경계부근에 대한 계산만 잘 수행되도록 변경하여 병렬화하였다. 통신량이 그렇게 많지 않아 scalability 역시 어느정도 잘 나오는 결과를 보였으며, 계산된 결과를 write하는 부분을 제외한다면 상대적으로 더 좋은 scalability 성능을 얻을 수 있다.

CHAPTER II. CTBL 코드

1. Code information

CFD Simulation 코드, DCTBL 코드

2. Makefile

Original 코드의 Makefile

```
FFLAGS= -p -O3 -qarch=pwr4 -qtune=pwr4 -qrealsize=8 -qdpc=e -qnolm   -qnosave
FFLAG = -p -O3 -qarch=pwr4 -qtune=pwr4 -qrealsize=8 -qdpc=e -qnolm   -qnosave
FC= xlf
LIB =
dctbl: $(OBJS)
          $(FC) $(FFLAG) $@.o -o $@ $(OBJS) $(LIB)   -bmaxdata:0x40000000 ₩
          -bmaxstack:0x10000000
```

최적화 코드의 Makefile

```
FFLAGS= -p -O3 -qnosave -qarch=pwr4 -qtune=pwr4 -qrealsize=8 -qdpc=e
FFLAG = -p -O3 -qnosave -qarch=pwr4 -qtune=pwr4 -qrealsize=8 -qdpc=e
FC= xlf
LIB= libfft.a
dctbl: $(OBJS)
          $(FC) $(FFLAG) $@.o -o $@ $(OBJS) $(LIB)   -bmaxdata:0x40000000 ₩
          -bmaxstack:0x10000000 -pg –g
```

OpenMP 병렬화 코드의 Makefile

```
FFLAGS= -O3 -qnosave -qarch=pwr3 -qtune=pwr3 -qrealsize=8 -qdpc=e ₩
           -qsmp=noauto -qreport=smplist   -qsource
FFLAG = -O3 -qnosave -qarch=pwr3 -qtune=pwr3 -qrealsize=8 -qdpc=e ₩
           -qsmp=noauto -qreport=smplist
FC= xlf_r
LIB = ../large/libfft.a
dctbl: $(OBJS)
          $(FC) $(FFLAG) $@.o -o $@ $(OBJS) $(LIB)   -bmaxdata:0x80000000 ₩
           -bmaxstack:0x10000000
```

–qnosave

: Fortran의 로컬변수는 static변수와 automatic 변수가 있다. static 변수는 프로그램이 끝날 때까지 data storages에 변수의 메모리 공간이 계속 유지 되며, Fortran77 컴파일러에서는 기본적으로 static 로컬변수를 사용한다. -qnosave옵션은 모든 로컬변수를 automatic 변수로 사용한다는 뜻으로, automatic 변수는 stack storage공간에 저장된다.

-qrealsize=8

: kind 타입이 지정되지 않은 단정도, 배정도의 실수 상수, 변수, 함수 등의 정밀도를 두 배로 증가 시킨다.

3. Flow Chart

프로그램의 전반적인 구조는 다음과 같이 fileopen, settup, gettup, end로 구성이 되어있다. Fileopen에서 input파일을 읽은 후 settup에서 mesh를 생성하고, getup에서 값을 계산하는 코드이다. Solver의 루틴은 Ax=b의 선형방정식을 풀게 되며, tomas-algorithm을 사용하여, banded tridiagonal matrix를 풀고 있다.

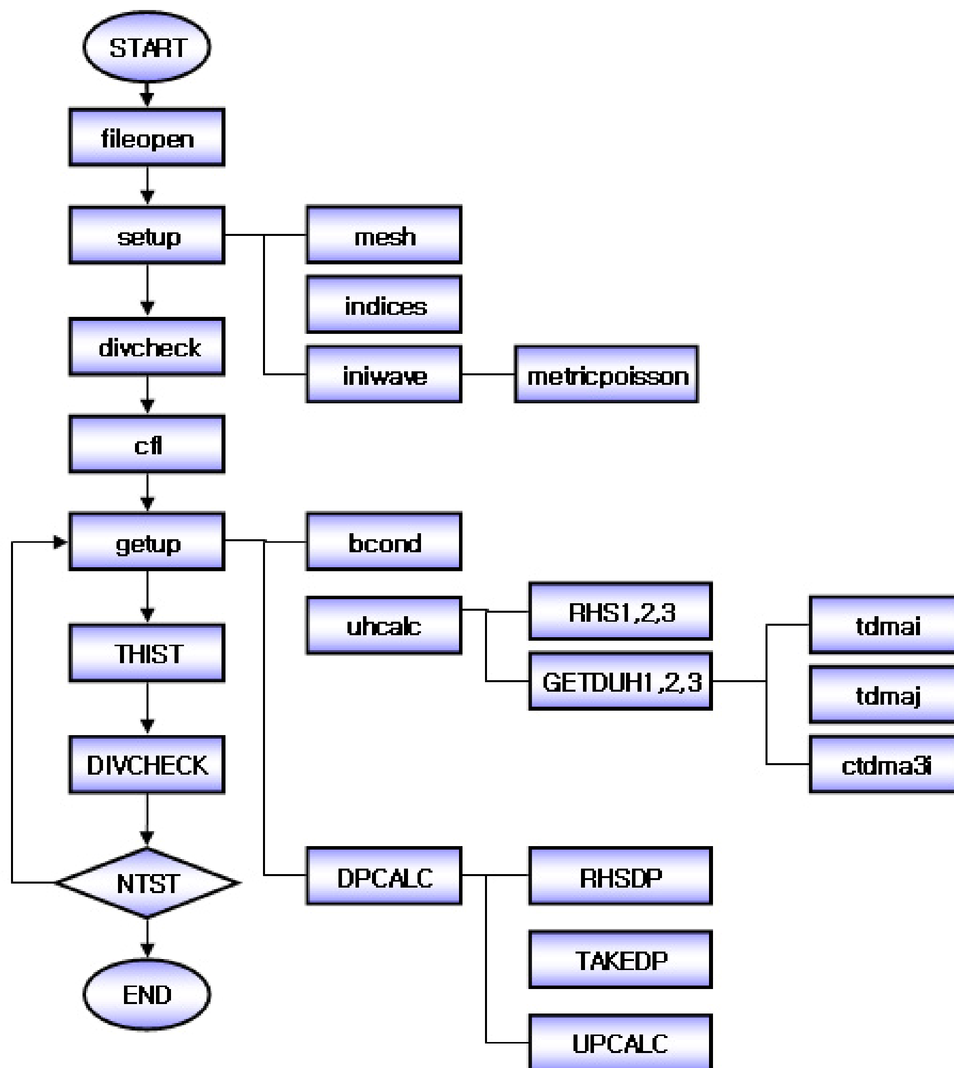


그림.II.1 DCTBL flow chart

4. Profiling 결과

다음은 Original코드, 최적화된 순차코드, OpenMP 병렬코드에 대한 각 프로파일링 결과이다. Original코드의 Tdma 서브루틴을 최적화된 순차코드에서는 tdmai, tdmaj, tdmai0로 분리하였다.

<Original 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 720.60 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
14.6	105.13	105.13	50	2102.60	2102.60	.upcalc [6]
13.1	199.88	94.75	50	1895.00	2137.22	.getduh3 [5]
10.8	277.92	78.04	50	1560.80	1800.72	.getduh2 [7]
10.3	352.50	74.58	50	1491.60	9920.70	.uhcalc [4]
10.0	424.70	72.20	50	1444.00	1685.36	.getduh1 [8]
6.8	473.52	48.82	50	976.40	976.40	.rhs3 [11]
6.6	521.20	47.68	50	953.60	953.60	.rhs1 [12]
6.1	564.99	43.79	50	875.80	875.80	.rhs2 [13]
5.1	601.63	36.64	50	732.80	989.50	.takedp [10]
4.1	631.41	29.78	50	595.60	13829.40	.getup [3]
3.1	653.96	22.55	7769600	0.00	0.00	.tdma [14]
2.6	672.34	18.38	2441600	0.01	0.01	.ctdma3 [15]
2.1	687.38	15.04	51	294.90	294.90	.divcheck [16]
1.5	698.15	10.77	50	215.40	215.40	.rhsdp [17]
0.9	704.90	6.75	101	66.83	66.83	.cfl [18]
0.4	707.56	2.66	5	532.00	532.00	.plane_up [21]
0.3	709.54	1.98	1228800	0.00	0.00	.rftbsub [22]
0.2	711.06	1.52	3276800	0.00	0.00	.cftmdl [25]
0.2	712.55	1.49				._xlqsub [26]
0.2	713.99	1.44	1228800	0.00	0.00	.rftfsub [27]
0.2	715.18	1.19				.__mcount [28]
0.1	716.17	0.99	2457600	0.00	0.00	.cft1st [29]
0.1	717.04	0.87	1	870.00	870.00	.iniup [30]
0.1	717.60	0.56	819200	0.00	0.00	.dctsub [31]
0.1	718.15	0.55	2457700	0.00	0.00	.bitrv2 [32]
0.1	718.54	0.39	748169	0.00	0.00	.load_locale [34]

<최적화 코드의 profiling 결과>

ngranularity: Each sample hit covers 4 bytes. Time: 109.79 seconds

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
15.1	16.63	16.63	50	332.60	474.30	.getduh3 [5]
10.0	27.62	10.99	50	219.80	219.80	.rhs2 [8]
10.0	38.58	10.96	50	219.20	219.20	.rhs3 [9]
9.6	49.15	10.57	9550	1.11	1.11	.ctdma3i [10]
8.5	58.53	9.38	50	187.60	187.60	.rhs1 [11]

5.7	64.74	6.21	19200	0.32	0.32	.tdmaj [15]
5.4	70.68	5.94	50	118.80	259.40	.getduh2 [6]
4.4	75.52	4.84	50	96.80	238.50	.getduh1 [7]
4.2	80.15	4.63	6551816	0.00	0.00	.cvtloop [17]
4.0	84.57	4.42	19200	0.23	0.23	.tdmai [18]
2.4	87.22	2.65	50	53.00	81.00	.takedp [19]
2.2	89.62	2.40	50	48.00	48.00	.upcalc [20]
1.5	91.25	1.63	50	32.60	1821.60	.getup [3]
1.5	92.86	1.61	101	15.94	15.94	.cfl [21]
1.4	94.39	1.53	50	30.60	30.60	.rhsdp.GL [22]
1.3	95.79	1.40	6450	0.22	0.22	.tdmai0 [23]
1.3	97.19	1.40	51	27.45	27.45	.divcheck [24]
1.2	98.52	1.33	50	26.60	1625.40	.uhcalc [4]
1.2	99.84	1.32	6792320	0.00	0.00	.__cvt_r [14]
1.0	100.92	1.08				.__mcount [25]
0.8	101.80	0.88				.dcosj3210f [26]
0.6	102.49	0.69				.memmove [27]
0.6	103.17	0.68				.dcosk3210f [28]
0.6	103.83	0.66				._xlfBeginIO [29]
0.4	104.27	0.44				.WriteUnit [30]
0.4	104.69	0.42				.dcosj1610l [31]
0.3	105.07	0.38				.LdNIR8ToDec [12]
0.3	105.43	0.36				.dcosk1610l [32]
0.3	105.71	0.28				.dcr1610f [33]
0.2	105.96	0.25				._xlfEndIO [34]
0.2	106.19	0.23				.drc1610f [35]
0.2	106.40	0.21				.drc1630f [36]
0.2	106.61	0.21				.drc831nl [37]
0.2	106.81	0.20	50	4.00	4.00	.bcond [38]
0.2	107.01	0.20				.IOPutByte [39]
0.2	107.21	0.20				.dcosk320f [40]
0.1	107.36	0.15	13244000	0.00	0.00	.pwr10 [42]
0.1	107.51	0.15				._xlfWriteLDReal [43]
0.1	107.65	0.14				.BeginIOWriteLd [44]
0.1	107.79	0.14				.PrepareUnit [45]
0.1	107.92	0.13				.dcr831pl [46]
0.1	108.04	0.12	6622000	0.00	0.00	.mf2x1 [47]
0.1	108.16	0.12				.IOTerminateRecord [48]
0.1	108.26	0.10	1	100.00	100.00	.iniup [49]
0.1	108.36	0.10				.GetMyLocalContext [50]
0.1	108.46	0.10				.IOWrite [51]
0.1	108.55	0.09				.dcosk322f [52]
0.1	108.63	0.08				.AllowIOAfterEOF [53]
0.1	108.71	0.08				.EndIOWriteLd [54]
0.1	108.79	0.08				.GetUnit [55]
0.1	108.87	0.08				.dcr811pl [56]
0.1	108.94	0.07				.FindUnit [57]
0.1	109.01	0.07				._ptrgl [58]
0.1	109.08	0.07				.dcr162f [59]
0.1	109.14	0.06	1	60.00	94280.00	.main [1]
0.1	109.20	0.06				._xlfEndIO.GL [60]

<OpenMP 코드의 profiling 결과>

```
ngranularity: Each sample hit covers 4 bytes. Time: 333.42 seconds

  %   cumulative   self              self     total
 time   seconds   seconds    calls  ms/call  ms/call  name
 10.5     35.08    35.08      799    43.90    43.90  .rhs3@OL@1 [2]
  9.9     68.13    33.05      796    41.52    41.52  .rhs2@OL@1 [3]
  9.6    100.20    32.07     9520     3.37     3.37  .ctdma3i [4]
  9.3    131.08    30.88      766    40.31    40.31  .rhs1@OL@1 [5]
  7.4    155.68    24.60                             .ThdCode [6]
  5.4    173.75    18.07                             ._xlsmpSyncRegionItem [7]
  5.0    190.34    16.59    19153     0.87     0.87  .tdmaj [12]
  4.0    203.78    13.44      799    16.82    21.50  .getduh3@OL@1 [8]
  3.4    214.97    11.19    19143     0.58     0.58  .tdmai [15]
  2.3    222.65     7.68      798     9.62     9.62  .getup@OL@1 [19]
  2.2    229.95     7.30      800     9.12    13.78  .getduh1@OL@1.GL [16]
  2.1    237.02     7.07      794     8.90    13.60  .getduh2@OL@1 [18]
  1.9    243.38     6.36      798     7.97    21.44  .getduh1@OL@3 [9]
  1.9    249.70     6.32      795     7.95    21.25  .getduh2@OL@3 [10]
  1.9    255.95     6.25      796     7.85    14.79  .getduh3@OL@3 [13]
  1.8    261.99     6.04      793     7.62    21.17  .getduh3@OL@2 [11]
  1.7    267.73     5.74      795     7.22    14.18  .getduh2@OL@2 [14]
  1.6    273.16     5.43      796     6.82    13.78  .getduh1@OL@2 [17]
  1.4    277.91     4.75      798     5.95     5.95  .uhcalc@OL@1 [22]
  1.3    282.37     4.46      789     5.65     5.65  .rhsdp@OL@1 [23]
  1.3    286.59     4.22      793     5.32     5.32  .getduh3@OL@5 [24]
  1.1    290.38     3.79     1609     2.36     2.36  .cfl@OL@1 [25]
  1.1    293.91     3.53      792     4.46     6.49  .takedp@OL@3 [20]
  1.0    297.28     3.37      793     4.25     6.19  .takedp@OL@4 [21]
  1.0    300.53     3.25      797     4.08     4.08  .takedp@OL@2 [26]
  1.0    303.72     3.19      791     4.03     4.03  .getduh3@OL@4 [27]
  0.9    306.87     3.15     6424     0.49     0.49  .tdmai0 [28]
  0.9    309.92     3.05       16   190.62   190.62  .iniup@OL@1 [29]
  0.8    312.49     2.57      794     3.24     3.24  .takedp@OL@5 [30]
  0.7    314.93     2.44      811     3.01     3.01  .divcheck@OL@1 [31]
  0.7    317.28     2.35                             .dcosj3211f [32]
  0.7    319.56     2.28      789     2.89     2.89  .upcalc@OL@3 [33]
  0.7    321.75     2.19      793     2.76     2.76  .upcalc@OL@1 [34]
  0.6    323.82     2.07      796     2.60     2.60  .upcalc@OL@2 [35]
  0.5    325.55     1.73      792     2.18     2.18  .upcalc@OL@4 [36]
  0.5    327.21     1.66                             .dcosk3211f [37]
  0.3    328.27     1.06                             .dcosj3210l [38]
  0.2    329.01     0.74                             .dcosk3210l [39]
  0.2    329.54     0.53                             .drc831nl [40]
  0.1    330.00     0.46                             .dcr1610f [41]
  0.1    330.42     0.42                             .drc1610f [42]
  0.1    330.72     0.30                             .allocate_memory [43]
  0.1    330.98     0.26                             .dcosk320f [44]
  0.1    331.19     0.21                             .drc1630f [45]
  0.1    331.37     0.18      784     0.23     0.23  .bcond@OL@3 [46]
  0.1    331.55     0.18                             .dcr811pl [47]
```

5. 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 순차코드 그리고 OpenMP 병렬코드에 대해 hpmcount를 실행시킨 결과이다.

<Original 코드의 hpmcount 결과>

```
Total execution time (wall clock time): 603.744974 seconds

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

 Total amount of time in user mode                : 600.000000 seconds
 Total amount of time in system mode              : 0.660000 seconds
 Maximum resident set size                        : 225328 Kbytes
 Average shared memory use in text segment        : 119987 Kbytes*sec
 Average unshared memory use in data segment      : 133708275 Kbytes*sec
 Number of page faults without I/O activity       : 56298
 Number of page faults with I/O activity          : 55
 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                      : 1
 Number of voluntary context switches             : 34
 Number of involuntary context switches           : 4081

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

  PM_FPU_FDIV (FPU executed FDIV instruction)                  :     900048273
  PM_FPU_FMA (FPU executed multiply-add instruction)           :   13680113464
  PM_FPU0_FIN (FPU0 produced a result)                         :   26567056515
  PM_FPU1_FIN (FPU1 produced a result)                         :   24500449179
  PM_CYC (Processor cycles)                                    :  569874701858
  PM_FPU_STF (FPU executed store instruction)                  :    7035759817
  PM_INST_CMPL (Instructions completed)                        :   90050778564
  PM_LSU_LDF (LSU executed Floating Point load instruction)    :   25517110841

  Utilization rate                          :        72.608 %
  Load and store operations                 :     32552.871 M
  MIPS                                      :       149.154
  Instructions per cycle                    :         0.158
  HW Float points instructions per Cycle    :         0.090
  Floating point instructions + FMAs        :     57711.859 M
  Float point instructions + FMA rate       :        95.590 Mflip/s
  FMA percentage                            :        47.408 %
  Computation intensity                     :         1.773
```

<최적화 코드의 hpmcount 결과>

```
Total execution time (wall clock time): 223.787592 seconds

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

 Total amount of time in user mode                 : 219.580000 seconds
 Total amount of time in system mode               : 1.880000 seconds
 Maximum resident set size                         : 194452 Kbytes
 Average shared memory use in text segment         : 139002 Kbytes*sec
 Average unshared memory use in data segment       : 42332373 Kbytes*sec
 Number of page faults without I/O activity        : 48472
 Number of page faults with I/O activity           : 15
 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              : 39
 Number of involuntary context switches            : 987

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

  PM_FPU_FDIV (FPU executed FDIV instruction)                   :      1055291740
  PM_FPU_FMA (FPU executed multiply-add instruction)            :     29190297957
  PM_FPU0_FIN (FPU0 produced a result)                          :     51460161638
  PM_FPU1_FIN (FPU1 produced a result)                          :     48933266798
  PM_CYC (Processor cycles)                                     :    208854611864
  PM_FPU_STF (FPU executed store instruction)                   :     14335032491
  PM_INST_CMPL (Instructions completed)                         :    166558328992
  PM_LSU_LDF (LSU executed Floating Point load instruction)     :     44129969563

  Utilization rate                          :          71.790 %
  Load and store operations                 :       58465.002 M
  MIPS                                      :         744.270
  Instructions per cycle                    :           0.797
  HW Float points instructions per Cycle    :           0.481
  Floating point instructions + FMAs        :      115248.694 M
  Float point instructions + FMA rate       :         514.991 Mflip/s
  FMA percentage                            :          50.656 %
  Computation intensity                     :           1.971
```

<OpenMP 코드의 hpmcount 결과>

```
Total execution time (wall clock time): 51.252509 seconds

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

 Total amount of time in user mode                : 315.770000 seconds
 Total amount of time in system mode              : 15.410000 seconds
 Maximum resident set size                        : 534332 Kbytes
 Average shared memory use in text segment        : 190291 Kbytes*sec
 Average unshared memory use in data segment      : 170175529 Kbytes*sec
 Number of page faults without I/O activity       : 133750
 Number of page faults with I/O activity          : 7
 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             : 50379
 Number of involuntary context switches           : 480

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

  PM_FPU_FDIV (FPU executed FDIV instruction)                 :      2101017772
  PM_FPU_FMA (FPU executed multiply-add instruction)          :     62617858391
  PM_FPU0_FIN (FPU0 produced a result)                        :    112006870947
  PM_FPU1_FIN (FPU1 produced a result)                        :    105470686725
  PM_CYC (Processor cycles)                                   :    562859809572
  PM_FPU_STF (FPU executed store instruction)                 :     30004727279
  PM_INST_CMPL (Instructions completed)                       :    434609087000
  PM_LSU_LDF (LSU executed Floating Point load instruction)    :    109727152458

  Utilization rate                         :          644.489 %
  Load and store operations                :       139731.880 M
  MIPS                                     :         8479.762
  Instructions per cycle                   :            0.772
  HW Float points instructions per Cycle   :            0.386
  Floating point instructions + FMAs       :       250090.689 M
  Float point instructions + FMA rate      :         4879.579 Mflip/s
  FMA percentage                           :           50.076 %
  Computation intensity                    :            1.790
```

Original 코드와 최적화된 순차코드의 hpmcount 결과에서 단위시간당 부동소수 연산회수를 나타내는 Float point instructions + FMA rate을 비교해 보면 각각 95.590 Mflip/s와 514.99 Mflip/s로, 이를 통해 최적화된 순차코드에서 단위시간당 더 많은 부동소수 연산을 수행해 코드의 효율 및 성능이 좋아졌음을 확인할 수 있다.

6. 순차코드 최적화 비교 및 분석

코드 전체에 걸쳐서, 비연속적인 메모리 접근을 유발하는 루프 nest에 대해 메모리 접근이 연속적이 되도록 루프의 순서를 바꿔주는 수정을 하였다. 코드에서 수행하는 계산의 핵심이 되는 선형 방정식을 풀이하는 과정에서는 1차원 배열을 2차원으로 변경하여 계산 수행을 위한 루틴 호출의 횟수를 줄이도록 하였다. 그리고 Fast Fourier Transform 라이브러리 사용을 통해 성능 향상을 시도하였다. 관련하여 수정된 내용들을 하나씩 살펴보도록 한다.

A. indices.f

< 최적화 코드 >

```
DO J=1,N2M
   FJMU(J)=J-JMU(J)
   FJMV(J)=J-JMV(J)
   FJPA(J)=JPA(J)-J
ENDDO
DO I=1,N1M
   FIMU(I)=I-IMU(I)
   FIMV(I)=I-IMV(I)
   FIPA(I)=IPA(I)-I
ENDDO
```

① 전체 코드에서 J-JMU(J), J-JMV(J), JPA(J)-J, I-IMU(I), I-IMV(I), IPA(I)-I의 계산이 자주 필요하다. 자주 반복되어야 하는 이와 같은 계산을 미리 배열에 넣어둠으로써 불필요한 반복 계산 회수를 줄일 수 있다. 위 부분은 Original 코드에는 없는 부분으로 최적화 코드에서는 위와 같은 인덱스 계산이 필요할 때 마다 배열 FJMU, FJMV, FJPA, FIMU, FIMV, FIPA를 불러 쓰게 된다.

B. iniup.f

< Original 코드 >

```
20                DO 10 NV=1,3
21                DO 10 I=0,N1
22                DO 10 J=0,N2
23                DO 10 K=1,N3M
24     62     10  U(NV,I,J,K)=0.0
25
26
27            C   IMPOSE U VELOCITY
28                DO 20 I=0,N1
29                DO 20 J=0,N2
30                DO 20 K=1,N3M
31     17     20  U(1,I,J,K)=1.0
32
```

```
33
34              C       IMPOSE ZERO-PRESSURE FLUCTUATIONS
35                      DO 60 I=1,N1M
36                      DO 60 J=1,N2M
37                      DO 60 K=1,N3M
38      8               P(I,J,K)=0.0
39                  60 CONTINUE
```

< 최적화 코드 >

```
20                      DO 10 K=1,N3M
21                      DO 10 J=0,N2
22                      DO 10 I=0,N1
23      4               U(I,J,K,1)=0.0
24      4               U(I,J,K,2)=0.0
25                      U(I,J,K,3)=0.0
26              10      CONTINUE
27
28
29              C       IMPOSE U VELOCITY
30                      DO 20 K=1,N3M
31                      DO 20 J=0,N2
32                      DO 20 I=0,N1
33      1       20      U(I,J,K,1)=1.0
34
35
36              C       IMPOSE ZERO-PRESSURE FLUCTUATIONS
37                      DO 60 K=1,N3M
38                      DO 60 J=1,N2M
39                      DO 60 I=1,N1M
40      2               P(I,J,K)=0.0
41                  60 CONTINUE
```

① Original 코드에서 배열 U(NV,I,J,K)에 대해 루프 실행을 NV, I, J, K순으로 하고 있어 불연속적인 메모리 접근이 발생한다. Fortran에서 메모리 접근을 연속적으로 하기 위해서는 가장 오른쪽 루프 인덱스부터 접근되도록 코드를 구성해야 한다. 최적화 코드에서는 배열 U와 P에 대해 K, J, I의 순으로 루프가 실행되도록 코드를 수정하고 있다.

② 루프 실행을 K, J, I의 순으로 변경하면서 배열 U(NV,I,J,K)를 U(I,J,K,NV)로 수정하였다. 이것은 뒷부분에서 배열 U에 접근할 때 접근 연속성을 높이기 위한 수정이다. 예를 들자면,

```
DO 60 K=1,N3M
   DO 60 J=1,N2M
      DO 60 I=1,N1M
         ... U(3,I,J,K) ...
60 CONTINUE
```

위와 같은 방식의 접근이 코드 뒷부분에서 자주 있게 되는데 이럴 경우 불연속 메모리 접근에 의해 많은 성능손실이 발생하게 된다. 이런 상황에서 배열에 대한 접근이 연속적이 되도록 인덱스 NV의 위치를 가장 오른쪽에 오도록 미리 배열구조를 수정한 것이다. 이때, 가장 바깥쪽에 위치해야 할 NV 루프를 풀어 버린 것은 차후 K루프에 대한 OpenMP 병렬화를 위한 조치이다.

C. divcheck.f

< Original 코드 >

```
18                 DO 20 I=1,N1M
19                 DO 20 J=1,N2M
20                 DO 20 K=1,N3M
21    37           KP=KPA(K)
22                 KM=KMA(K)
23   919           DIV=ABS((U(1,I+1,J,K)-U(1,I,J,K))*DX1
24                >      +(U(2,I,J+1,K)-U(2,I,J,K))/DY(J)
25                >      +(U(3,I,J,KP)-U(3,I,J,K))*DX3)
26                 IF (DIV.GT.DIVMAX) THEN
27                    IMAX = I
28                    JMAX = J
29                    KMAX = K
30                 ENDIF
31   655           DIVMAX = AMAX1(DIV,DIVMAX)
32              20 CONTINUE
```

< 최적화 코드 >

```
18                 DO 20 K=1,N3M
19     1           DO 20 J=1,N2M
20                 DO 20 I=1,N1M
21                 KP=KPA(K)
22                 KM=KMA(K)
23   101           DIV=ABS((U(I+1,J,K,1)-U(I,J,K,1))*DX1
24                >      +(U(I,J+1,K,2)-U(I,J,K,2))/DY(J)
25                >      +(U(I,J,KP,3)-U(I,J,K,3))*DX3)
26                 IF (DIV.GT.DIVMAX) THEN
27                    IMAX = I
28                    JMAX = J
29                    KMAX = K
30                 ENDIF
31    45           DIVMAX = AMAX1(DIV,DIVMAX)
32              20 CONTINUE
```

① 앞선 iniup.f에서 수행된 최적화와 마찬가지로 메모리 접근이 연속적으로 이루어 지도록 I,J,K의 루프 실행 순서를 K,J,I로 변경하였다.

D. cfl.f

< Original 코드 >

```
19                    DO 10 K=1,N3M
20                    KP=KPA(K)
21                    DO 10 I=1,N1M
22         2          IP=I+1
23                    DO 10 J=1,N2M
24                    JP=J+1
25       891          CFLL=ABS(U(1,I,J,K)+U(1,IP,J,K))*0.5*DX1
26                   >     +ABS(U(2,I,J,K)+U(2,I,JP,K))*0.5/DY(J)
27                   >     +ABS(U(3,I,J,K)+U(3,I,J,KP))*0.5*DX3
28        24          CFLM=AMAX1(CFLM,CFLL)
29         5     10   CONTINUE
```

< 최적화 코드 >

```
19                    DO 10 K=1,N3M
20                    KP=KPA(K)
21                    DO 10 J=1,N2M
22         1          JP=J+1
23                    DO 10 I=1,N1M
24                    IP=I+1
25       183          CFLL=ABS(U(I,J,K,1)+U(IP,J,K,1))*0.5*DX1
26                   >     +ABS(U(I,J,K,2)+U(I,JP,K,2))*0.5/DY(J)
27                   >     +ABS(U(I,J,K,3)+U(I,J,KP,3))*0.5*DX3
28         2          CFLM=AMAX1(CFLM,CFLL)
29         2     10   CONTINUE
```

① 메모리 접근이 연속적이 되도록 I 루프와 J 루프의 순서를 변경 하였다.

E. getup.f

< Original 코드 >

```
13               C    INITIALIZE THE INTERMEDIATE VELOCITY AND PRESSURE
14                    DO 10 NV=1,3
15                    DO 10 I=0,N1
16                    DO 10 J=0,N2
17                    DO 10 K=1,N3M
18      2714     10   UH(NV,I,J,K)=0.0
19                    DO 20 I=1,N1M
20                    DO 20 J=1,N2M
21                    DO 20 K=1,N3M
22       297     20   DP(I,J,K)=0.0
```

< 최적화 코드 >

```
14                     C      INITIALIZE THE INTERMEDIATE VELOCITY AND PRESSURE
15                            DO 10 K=1,N3M
16                            DO 10 J=0,N2
17                            DO 10 I=0,N1
18          12                UH(I,J,K,1)=0.0
19         169                UH(I,J,K,2)=0.0
20           2                UH(I,J,K,3)=0.0
21                    10      CONTINUE
22                            DO 20 K=1,N3M
23           1                DO 20 J=1,N2M
24                            DO 20 I=1,N1M
25          54        20      DP(I,J,K)=0.0
```

① 앞선 iniup.f에서 수행된 최적화와 마찬가지로 메모리 접근이 연속적으로 이루어 지도록 I, J, K의 루프 실행 순서를 K, J, I로 변경해 배열 UH, DP에 대한 접근 연속성을 높이고 있다.

F. uhcalc.f

< Original 코드 >

```
76                     C      INTERMEDIATE VELOCITY, U*=U^N+dU*
77
78                            DO 300 I=2,N1M
79                            DO 300 J=1,N2M
80                            DO 300 K=1,N3M
81        2695       300      UH(1,I,J,K)=U(1,I,J,K)+UH(1,I,J,K)
82
83                            DO 310 I=1,N1M
84                            DO 310 J=2,N2M
85                            DO 310 K=1,N3M
86        2695       310      UH(2,I,J,K)=U(2,I,J,K)+UH(2,I,J,K)
87
88                            DO 320 I=1,N1M
89                            DO 320 J=1,N2M
90                            DO 320 K=1,N3M
91        2753       320      UH(3,I,J,K)=U(3,I,J,K)+UH(3,I,J,K)
```

< 최적화 코드 >

```
76                     C      INTERMEDIATE VELOCITY, U*=U^N+dU*
77
78                            DO 300 K=1,N3M
79                            DO 300 J=1,N2M
80                            DO 300 I=2,N1M
81          60       300      UH(I,J,K,1)=U(I,J,K,1)+UH(I,J,K,1)
82
83                            DO 310 K=1,N3M
```

```
84                        DO 310 J=2,N2M
85                        DO 310 I=1,N1M
86        65       310    UH(I,J,K,2)=U(I,J,K,2)+UH(I,J,K,2)
87
88                        DO 320 K=1,N3M
89         1              DO 320 J=1,N2M
90                        DO 320 I=1,N1M
91        75       320    UH(I,J,K,3)=U(I,J,K,3)+UH(I,J,K,3)
```

① 불연속적인 메모리 접근을 피하기 위해 루프 실행 순서를 변경하였다.

G. rhs1.f

< Original 코드 >

```
20                        DO 10 J=1,N2M
21                        JP=J+1
22                        JM=J-1
23                        JUM=J-JMU(J)
24                        JUP=JPA(J)-J
25                        DO 10 I=2,N1M
26                        IP=I+1
27                        IM=I-1
28                        IUM=I-IMU(I)
29         7              IUP=IPA(I)-I
30                        DO 10 K=1,N3M
31       373              KP=KPA(K)
32                        KM=KMA(K)
34      1480              VISCOS=0.5*DX1Q/RE*(U(1,IP,J,K )-2.0*U(1,I,J,K)+U(1,IM,J,K ))
35                       >      +0.5*DX3Q/RE*(U(1,I ,J,KP)-2.0*U(1,I,J,K)+U(1,I ,J,KM))
36                       >      +JUM*JUP*0.5/RE*(HP(J)*U(1,I,JP,K)
37                       >                       -HC(J)*U(1,I,J ,K)
38                       >                       +HM(J)*U(1,I,JM,K))
39                       >      +(1-JUM)*0.5/RE*(2.0/H(2)/(H(1)+H(2))*U(1,I,2,K)
40                       >                       -2.0/H(1)/H(2)*U(1,I,1,K)
41                       >                       +2.0/H(1)/(H(1)+H(2))*U(1,I,0,K))
42                       >      +(1-JUP)*0.5/RE*(2.0/H(N2M)/(H(N2)+H(N2M))*U(1,I,N2M-1,K)
43                       >                       -2.0/H(N2M)/H(N2)*U(1,I,N2M,K)
44                       >                       +2.0/H(N2)/(H(N2)+H(N2M))*U(1,I,N2,K))
46       127              PRESSG1=DX1*(P(I,J,K)-P(IM,J,K))
48       228              BC_DOWN=0.5/RE*2.0/H(1)/(H(1)+H(2))*UBC1(1,I,K)
49                       >       +0.5/DY(1)*0.5*(U(2,I,1,K)+U(2,IM,1,K))*UBC1(1,I,K)
50                       >       +0.5/DY(1)*U(1,I,0,K)*0.5*(UBC1(2,I,K)+UBC1(2,IM,K))
52      1200              BC_UP   =0.5/RE*2.0/H(N2)/(H(N2)+H(N2M))*UBC2(1,I,K)
53                       >       -0.5/DY(N2M)*0.5*(U(2,I,N2,K)+U(2,IM,N2,K))*UBC2(1,I,K)
54                       >       -0.5/DY(N2M)*U(1,I,N2,K)*0.5*(UBC2(2,I,K)+UBC2(2,IM,K))
56         9              U2=0.5*(U(1,IP,J,K)+U(1,I ,J,K))
57        23              U1=0.5*(U(1,I ,J,K)+U(1,IM,J,K))
59       164              BC_IN = DX1*U1*0.5*UBC3(1,J,K)
60                       >       +0.5/RE*DX1Q*UBC3(1,J,K)
61         1              BC_OUT=-DX1*U2*0.5*UBC4(1,J,K)
62                       >       +0.5/RE*DX1Q*UBC4(1,J,K)
```

```
 65      165          BC=(1-JUM)*BC_DOWN
 66                  >   +(1-JUP)*BC_UP
 67                  >   +(1-IUM)*BC_IN
 68                  >   +(1-IUP)*BC_OUT
 70                   RDUH1(I,J,K)=1./DT*U(1,I,J,K)
 71                  >              -PRESSG1+VISCOS
 72                  >              +BC
..................................................

137        6      10   CONTINUE
```

< 최적화 코드 >

```
 25                   DO 10 K=1,N3M
 26                   KP=KPA(K)
 27                   KM=KMA(K)
 29                   DO I=2,N1M
 30                   IP=I+1
 31                   IM=I-1
 32        2          VISCOS1(I) =0.5/RE*(2.0/H(2)/(H(1)+H(2))*U(I,2,K,1)
 33                  >                          -2.0/H(1)/H(2)*U(I,1,K,1)
 34                  >                          +2.0/H(1)/(H(1)+H(2))*U(I,0,K,1))
 35        3          VISCOS2(I) =0.5/RE*(2.0/H(N2M)/(H(N2)+H(N2M))*U(I,N2M-1,K,1)
 36                  >                          -2.0/H(N2M)/H(N2)*U(I,N2M,K,1)
 37                  >                          +2.0/H(N2)/(H(N2)+H(N2M))*U(I,N2,K,1))
 38        2          BC_DOWN(I)=0.5/RE*2.0/H(1)/(H(1)+H(2))*UBC1(I,K,1)
 39                  >        +0.5/DY(1)*0.5*(U(I,1,K,2)+U(IM,1,K,2))*UBC1(I,K,1)
 40                  >        +0.5/DY(1)*U(I,0,K,1)*0.5*(UBC1(I,K,2)+UBC1(IM,K,2))
 41
 42                   BC_UP(I)   =0.5/RE*2.0/H(N2)/(H(N2)+H(N2M))*UBC2(I,K,1)
 43                  >        -0.5/DY(N2M)*0.5*(U(I,N2,K,2)+U(IM,N2,K,2))*UBC2(I,K,1)
 44                  >        -0.5/DY(N2M)*U(I,N2,K,1)*0.5*(UBC2(I,K,2)+UBC2(IM,K,2))
 45                   ENDDO
 47                   DO 10 J=1,N2M
 48                   JP=J+1
 49                   JM=J-1
 52                   FJUM=FJMU(J)
 53                   FJUP=FJPA(J)
 54        4          DO I=2,N1M
 55                   IP=I+1
 56                   IM=I-1
 59                   FIUM=FIMU(I)
 60       18          FIUP=FIPA(I)
 62      100          VISCOS=0.5*DX1Q/RE*(U(IP,J,K ,1)-2.0*U(I,J,K,1)+U(IM,J,K ,1))
 63                  >       +0.5*DX3Q/RE*(U(I ,J,KP,1)-2.0*U(I,J,K,1)+U(I ,J,KM,1))
 64                  >       +FJUM*FJUP*0.5/RE*(HP(J)*U(I,JP,K,1)
 65                  >                          -HC(J)*U(I,J ,K,1)
 66                  >                          +HM(J)*U(I,JM,K,1))
 67                  >       +(1.-FJUM)*VISCOS1(I)
 68                  >       +(1.-FJUP)*VISCOS2(I)
 70       14          PRESSG1=DX1*(P(I,J,K)-P(IM,J,K))
 73       14          U2=0.5*(U(IP,J,K,1)+U(I ,J,K,1))
```

```
  74          4          U1=0.5*(U(I ,J,K,1)+U(IM,J,K,1))
  76         32          BC_IN = DX1*U1*0.5*UBC3(J,K,1)
  77                    >          +0.5/RE*DX1Q*UBC3(J,K,1)
  78         30          BC_OUT=-DX1*U2*0.5*UBC4(J,K,1)
  79                    >          +0.5/RE*DX1Q*UBC4(J,K,1)
  82         45          BC=(1.-FJUM)*BC_DOWN(I)
  83                    >   +(1.-FJUP)*BC_UP(I)
  84                    >   +(1.-FIUM)*BC_IN
  85                    >   +(1.-FIUP)*BC_OUT
  87         65          RDUH1(I,J,K)=1./DT*U(I,J,K,1)
  88                    >              -PRESSG1+VISCOS
  89                    >              +BC
  90                     ENDDO
  91          4          DO 10 I=2,N1M
  92                     IP=I+1
  93                     IM=I-1
  96                     FIUM=FIMU(I)
  97                     FIUP=FIPA(I)
....................................................

 162
 163                  10    CONTINUE
```

① J, I, K 순으로 반복 실행되는 루프 계산을 K, J, I 순으로 바꿔 배열 U와 P에 대한 접근이 연속적이 되도록 하였다.

② 다음의 VISCOS, BC_DOWN, BC_UP계산은 J, I, K 루프 실행의 가장 안쪽에서 반복된다.

```
VISCOS=0.5*DX1Q/RE*(U(1,IP,J,K )-2.0*U(1,I,J,K)+U(1,IM,J,K ))
        +0.5*DX3Q/RE*(U(1,I ,J,KP)-2.0*U(1,I,J,K)+U(1,I ,J,KM))
        +JUM*JUP*0.5/RE*(HP(J)*U(1,I,JP,K)
                         -HC(J)*U(1,I,J ,K)
                         +HM(J)*U(1,I,JM,K))
        +(1-JUM)*0.5/RE*(2.0/H(2)/(H(1)+H(2))*U(1,I,2,K)
                         -2.0/H(1)/H(2)*U(1,I,1,K)
                         +2.0/H(1)/(H(1)+H(2))*U(1,I,0,K))
        +(1-JUP)*0.5/RE*(2.0/H(N2M)/(H(N2)+H(N2M))*U(1,I,N2M-1,K)
                         -2.0/H(N2M)/H(N2)*U(1,I,N2M,K)
                         +2.0/H(N2)/(H(N2)+H(N2M))*U(1,I,N2,K))


BC_DOWN=0.5/RE*2.0/H(1)/(H(1)+H(2))*UBC1(1,I,K)
          +0.5/DY(1)*0.5*(U(2,I,1,K)+U(2,IM,1,K))*UBC1(1,I,K)
          +0.5/DY(1)*U(1,I,0,K)*0.5*(UBC1(2,I,K)+UBC1(2,IM,K))

BC_UP   =0.5/RE*2.0/H(N2)/(H(N2)+H(N2M))*UBC2(1,I,K)
          -0.5/DY(N2M)*0.5*(U(2,I,N2,K)+U(2,IM,N2,K))*UBC2(1,I,K)
          -0.5/DY(N2M)*U(1,I,N2,K)*0.5*(UBC2(2,I,K)+UBC2(2,IM,K))
```

루프 반복을 K, J, I로 변경하면서 J 인덱스와 무관한 VISCOS의 일부분과

BC_DOWN, BC_UP 계산을 J 루프 밖으로 이동시켜 불필요한 계산이 반복되는 것을 막을 수 있다. 이를 위해 최적화된 코드에서는 J루프 밖에서 I루프를 돌려 VISCOS1(I), VISCOS2(I), BC_DOWN(I), BC_UP(I)를 계산하고 있다. 이를 통해 각 변수에 대해 K 루프 내에서 (N1M-1)*N2M회의 반복되는 계산을 (N1M-1)회로 줄일 수 있다.

③ 변수 IUM, IUP, JUM, JUP의 계산은 앞서 서브루틴 indicies에서 이미 계산돼 배열 FJMU, FJPA, FIMU, FIPA에 저장돼 있으므로 이 값을 각각 FJUM, FJUP, FIUM, FIUP으로 받아 사용하고 있다.

H. rhs2.f

< Original 코드 >

```
21                    DO 10 J=2,N2M
22                     JP=J+1
23                     JM=J-1
24                     JUM=J-JMV(J)
25                     JUP=JPA(J)-J
27                    DO 10 I=1,N1M
28                     IP=I+1
29                     IM=I-1
30         1           IUM=I-IMV(I)
31         5           IUP=IPA(I)-I
32                    DO 10 K=1,N3M
33                     KP=KPA(K)
34                     KM=KMA(K)
36      1011           VISCOS=0.5*DX1Q/RE*(U(2,IP,J,K)-2.*U(2,I,J,K)+U(2,IM,J,K))
37                    >        +0.5*DX3Q/RE*(U(2,I,J,KP)-2.0*U(2,I,J,K)+U(2,I,J,KM))
38                    >        +0.5/RE*(DYP(J)*U(2,I,JP,K)
39                    >                 -DYC(J)*U(2,I,J ,K)
40                    >                 +DYM(J)*U(2,I,JM,K))
42        20           PRESSG2=(P(I,J,K)-P(I,JM,K))/H(J)
44       782           BC_DOWN=0.5/RE*DYM(2)*UBC1(2,I,K)
45                    >         +1./H(2)*0.5*(U(2,I,2,K)+U(2,I,1,K))*0.5*UBC1(2,I,K)
46       299           BC_UP   =0.5/RE*DYP(N2M)*UBC2(2,I,K)
47                    >         -1./H(N2M)*0.5*(U(2,I,N2,K)+U(2,I,N2M,K))*0.5*UBC2(2,I,K)
49                     U2=1.0/H(J)*(DY(J)/2.0*U(1,IP,JM,K)+DY(JM)/2.0*U(1,IP,J,K))
50       646           U1=1.0/H(J)*(DY(J)/2.0*U(1,I ,JM,K)+DY(JM)/2.0*U(1,I ,J,K))
51       448           BC2=1.0/H(J)*(DY(J)/2.0*UBC4(1,JM,K)+DY(JM)/2.0*UBC4(1,J,K))
52                     BC1=1.0/H(J)*(DY(J)/2.0*UBC3(1,JM,K)+DY(JM)/2.0*UBC3(1,J,K))
53                     V2=0.5*(U(2,I,J,K)+U(2,IP,J,K))
54        47           V1=0.5*(U(2,I,J,K)+U(2,IM,J,K))
56        37           BC_IN =0.5*DX1Q/RE*UBC3(2,J,K)
57                    >        +0.5*DX1*U1*0.5*UBC3(2,J,K)
58                    >        +0.5*DX1*V1*BC1
59       312           BC_OUT=0.5*DX1Q/RE*UBC4(2,J,K)
60                    >        -0.5*DX1*U2*0.5*UBC4(2,J,K)
61                    >        -0.5*DX1*V2*BC2
```

```
 64          60              BC=(1-JUM)*BC_DOWN
 65                         >   +(1-JUP)*BC_UP
 66                         >   +(1-IUM)*BC_IN
 67                         >   +(1-IUP)*BC_OUT
 68
 69                          RDUH2(I,J,K)=1./DT*U(2,I,J,K)
 70                         >             -PRESSG2+VISCOS
 71                         >             +BC
.........................................

143           4        10    CONTINUE
```

< 최적화 코드 >

```
 25                          DO 10 K=1,N3M
 26                          KP=KPA(K)
 27                          KM=KMA(K)
 29                          DO J=2,N2M
 30                          JP=J+1
 31                          JM=J-1
 32           2              BC2(J)=1.0/H(J)*(DY(J)/2.0*UBC4(JM,K,1)+DY(JM)/2.0*UBC4(J,K,1))
 33                          BC1(J)=1.0/H(J)*(DY(J)/2.0*UBC3(JM,K,1)+DY(JM)/2.0*UBC3(J,K,1))
 34                          ENDDO
 35                          DO I=1,N1M
 36           1              BC_DOWN(I)=0.5/RE*DYM(2)*UBC1(I,K,2)
 37                         >         +1./H(2)*0.5*(U(I,2,K,2)+U(I,1,K,2))*0.5*UBC1(I,K,2)
 38           1              BC_UP(I)   =0.5/RE*DYP(N2M)*UBC2(I,K,2)
 39                         >         -1./H(N2M)*0.5*(U(I,N2,K,2)+U(I,N2M,K,2))*0.5*UBC2(I,K,2)
 40                          ENDDO
 42                          DO 10 J=2,N2M
 43                          JP=J+1
 44           6              JM=J-1
 47                          FJUM=FJMV(J)
 48           1              FJUP=FJPA(J)
 50                          DO 10 I=1,N1M
 51                          IP=I+1
 52                          IM=I-1
 55                          FIUM=FIMV(I)
 56                          FIUP=FIPA(I)
 58         287              VISCOS=0.5*DX1Q/RE*(U(IP,J,K,2)-2.*U(I,J,K,2)+U(IM,J,K,2))
 59                         >        +0.5*DX3Q/RE*(U(I,J,KP,2)-2.0*U(I,J,K,2)+U(I,J,KM,2))
 60                         >        +0.5/RE*(DYP(J)*U(I,JP,K,2)
 61                         >                 -DYC(J)*U(I,J ,K,2)
 62                         >                 +DYM(J)*U(I,JM,K,2))
 64          19              PRESSG2=(P(I,J,K)-P(I,JM,K))/H(J)
 66         141              U2=1.0/H(J)*(DY(J)/2.0*U(IP,JM,K,1)+DY(JM)/2.0*U(IP,J,K,1))
 67           3              U1=1.0/H(J)*(DY(J)/2.0*U(I ,JM,K,1)+DY(JM)/2.0*U(I ,J,K,1))
 68          15              V2=0.5*(U(I,J,K,2)+U(IP,J,K,2))
 69           7              V1=0.5*(U(I,J,K,2)+U(IM,J,K,2))
 71          41              BC_IN =0.5*DX1Q/RE*UBC3(J,K,2)
 72                         >        +0.5*DX1*U1*0.5*UBC3(J,K,2)
 73                         >        +0.5*DX1*V1*BC1(J)
```

```
74          34          BC_OUT=0.5*DX1Q/RE*UBC4(J,K,2)
75                     >        -0.5*DX1*U2*0.5*UBC4(J,K,2)
76                     >        -0.5*DX1*V2*BC2(J)
79          168         BC=(1.-FJUM)*BC_DOWN(I)
80                     >   +(1.-FJUP)*BC_UP(I)
81                     >   +(1.-FIUM)*BC_IN
82                     >   +(1.-FIUP)*BC_OUT
84          61          RDUH2(I,J,K)=1./DT*U(I,J,K,2)
85                     >             -PRESSG2+VISCOS
86                     >             +BC
.........................................

159                 10    CONTINUE
```

① J, I, K 순으로 반복 실행되는 루프 계산을 K, J, I 순으로 바꿔 배열 U와 P에 대한 접근이 연속적이 되도록 하였다.

② Original 코드에서 다음의 BC_DOWN, BC_UP, BC1, BC2 계산은 J, I, K 루프 실행의 가장 안쪽에서 반복된다

```
BC_DOWN=0.5/RE*DYM(2)*UBC1(2,I,K)
         +1./H(2)*0.5*(U(2,I,2,K)+U(2,I,1,K))*0.5*UBC1(2,I,K)
BC_UP   =0.5/RE*DYP(N2M)*UBC2(2,I,K)
         -1./H(N2M)*0.5*(U(2,I,N2,K)+U(2,I,N2M,K))*0.5*UBC2(2,I,K)

BC2=1.0/H(J)*(DY(J)/2.0*UBC4(1,JM,K)+DY(JM)/2.0*UBC4(1,J,K))
BC1=1.0/H(J)*(DY(J)/2.0*UBC3(1,JM,K)+DY(JM)/2.0*UBC3(1,J,K))
```

루프 반복을 K, J, I로 변경하면서 J 인덱스와 무관한 BC_DOWN, BC_UP 계산을 J 루프 밖으로 이동시키고, I 인덱스와 무관한 BC2, BC1 계산을 역시 J 루프 밖에서 처리함으로써 불필요한 반복 계산을 막는다. 최적화된 코드에서는 J루프 밖에서 I 루프를 돌려 BC_DOWN(I), BC_UP(I)를 계산하고 있고, 역시 J 루프 밖에서 또 다른 J 루프를 돌려 BC(I)1, BC2(I)를 계산하고 있다. 이를 통해 K 루프 내에서 (N2M-1)*N1M회의 반복되는 계산을 BC_DOWN, BC_UP과 BC1, BC2의 계산을 (N2M-1)회와 N1M회로 각각 줄일 수 있다.

③ 앞서의 rhs1에서처럼 FIUP, FIUM, FJUM, FJUP등은 미리 계산된 값을 가져와 사용하고 있다.

I. rhs3.f

< Original 코드 >

```
20                   DO 10 J=1,N2M
21                   JP=J+1
22                   JM=J-1
23                   JUM=J-JMU(J)
24                   JUP=JPA(J)-J
26                   DO 10 I=1,N1M
```

```
 27                      IP=I+1
 28                      IM=I-1
 29         1            IUM=I-IMV(I)
 30         1            IUP=IPA(I)-I
 32                      DO 10 K=1,N3M
 33       382            KP=KPA(K)
 34                      KM=KMA(K)
 36      1232            VISCOS=0.5*DX1Q/RE*(U(3,IP,J,K)-2.*U(3,I,J,K)+U(3,IM,J,K))
 37                     >      +0.5*DX3Q/RE*(U(3,I ,J,KP)-2.0*U(3,I,J,K)+U(3,I ,J,KM))
 38                     >      +JUM*JUP*0.5/RE*(HP(J)*U(3,I,JP,K)
 39                     >                  -HC(J)*U(3,I,J ,K)
 40                     >                  +HM(J)*U(3,I,JM,K))
 41                     >      +(1-JUM)*0.5/RE*(2.0/H(2)/(H(1)+H(2))*U(3,I,2,K)
 42                     >                  -2.0/H(1)/H(2)*U(3,I,1,K)
 43                     >                  +2.0/H(1)/(H(1)+H(2))*U(3,I,0,K))
 44                     >      +(1-JUP)*0.5/RE*(2.0/H(N2M)/(H(N2M)+H(N2))*U(3,I,N2M-1,K)
 45                     >                  -2.0/H(N2)/H(N2M)*U(3,I,N2M,K)
 46                     >                  +2.0/H(N2)/(H(N2M)+H(N2))*U(3,I,N2,K))
 48       104            PRESSG3=DX3*(P(I,J,K)-P(I,J,KM))
 50       594            BC_DOWN=0.5/RE*2.0/H(1)/(H(1)+H(2))*UBC1(3,I,K)
 51                     >       +0.5/DY(1)*0.5*(U(2,I,1,KM)+U(2,I,1,K))*UBC1(3,I,K)
 52                     >       +0.5/DY(1)*U(3,I,0,K)*0.5*(UBC1(2,I,K)+UBC1(2,I,KM))
 54       220            BC_UP   =0.5/RE*2.0/H(N2)/(H(N2M)+H(N2))*UBC2(3,I,K)
 55                     >       -0.5/DY(N2M)*0.5*(U(2,I,N2,KM)+U(2,I,N2,K))*UBC2(3,I,K)
 56                     >       -0.5/DY(N2M)*U(3,I,N2,K)*0.5*(UBC2(2,I,K)+UBC2(2,I,KM))
 58        10            U2=0.5*(U(1,IP,J,K)+U(1,IP,J,KM))
 59       771            U1=0.5*(U(1,I ,J,K)+U(1,I ,J,KM))
 60       310            BC2=0.5*(UBC4(1,J,K)+UBC4(1,J,KM))
 61        19            BC1=0.5*(UBC3(1,J,K)+UBC3(1,J,KM))
 62        18            W2=0.5*(U(3,IP,J,K)+U(3,I,J,K))
 63        39            W1=0.5*(U(3,IM,J,K)+U(3,I,J,K))
 65       363            BC_IN =0.5*DX1Q/RE*UBC3(3,J,K)
 66                     >      +0.5*DX1*U1*0.5*UBC3(3,J,K)
 67                     >      +0.5*DX1*W1*BC1
 68        50            BC_OUT=0.5*DX1Q/RE*UBC4(3,J,K)
 69                     >      -0.5*DX1*U2*0.5*UBC4(3,J,K)
 70                     >      -0.5*DX1*W2*BC2
 73        52            BC=(1-JUM)*BC_DOWN
 74                     >  +(1-JUP)*BC_UP
 75                     >  +(1-IUM)*BC_IN
 76                     >  +(1-IUP)*BC_OUT
 78         8            RDUH3(I,J,K)=1./DT*U(3,I,J,K)
 79                     >          -PRESSG3+VISCOS
 80                     >          +BC
..................................................
158        10       10   CONTINUE
```

< 최적화 코드 >

```
 24                      DO 10 K=1,N3M
 25                      KP=KPA(K)
 26                      KM=KMA(K)
```

```
28                        DO I=1,N1M
29          3              VISCOS1(I)=0.5/RE*(2.0/H(2)/(H(1)+H(2))*U(I,2,K,3)
30                       >                   -2.0/H(1)/H(2)*U(I,1,K,3)
31                       >                   +2.0/H(1)/(H(1)+H(2))*U(I,0,K,3))
32          1              VISCOS2(I)=0.5/RE*(2.0/H(N2M)/(H(N2M)+H(N2))*U(I,N2M-1,K,3)
33                       >                   -2.0/H(N2)/H(N2M)*U(I,N2M,K,3)
34                       >                   +2.0/H(N2)/(H(N2M)+H(N2))*U(I,N2,K,3))
35          8              BC_DOWN(I)=0.5/RE*2.0/H(1)/(H(1)+H(2))*UBC1(I,K,3)
36                       >       +0.5/DY(1)*0.5*(U(I,1,KM,2)+U(I,1,K,2))*UBC1(I,K,3)
37                       >       +0.5/DY(1)*U(I,0,K,3)*0.5*(UBC1(I,K,2)+UBC1(I,KM,2))
38
39          2              BC_UP(I)    =0.5/RE*2.0/H(N2)/(H(N2M)+H(N2))*UBC2(I,K,3)
40                       >       -0.5/DY(N2M)*0.5*(U(I,N2,KM,2)+U(I,N2,K,2))*UBC2(I,K,3)
41                       >       -0.5/DY(N2M)*U(I,N2,K,3)*0.5*(UBC2(I,K,2)+UBC2(I,KM,2))
42                        ENDDO
45                        DO 10 J=1,N2M
46                        JP=J+1
47                        JM=J-1
50                        FJUM=FJMU(J)
51          2              FJUP=FJPA(J)
53                        DO I=1,N1M
54                         IP=I+1
55                         IM=I-1
58                         FIUM=FIMV(I)
59                         FIUP=FIPA(I)
62        268              VISCOS=0.5*DX1Q/RE*(U(IP,J,K ,3)-2.0*U(I,J,K,3)+U(IM,J,K ,3))
63                       >      +0.5*DX3Q/RE*(U(I ,J,KP,3)-2.0*U(I,J,K,3)+U(I ,J,KM,3))
64                       >      +FJUM*FJUP*0.5/RE*(HP(J)*U(I,JP,K,3)
65                       >                         -HC(J)*U(I,J ,K,3)
66                       >                         +HM(J)*U(I,JM,K,3))
67                       >      +(1.-FJUM)*VISCOS1(I)
68                       >      +(1.-FJUP)*VISCOS2(I)
70         51              PRESSG3=DX3*(P(I,J,K)-P(I,J,KM))
72                         U2=0.5*(U(IP,J,K,1)+U(IP,J,KM,1))
73          2              U1=0.5*(U(I ,J,K,1)+U(I ,J,KM,1))
74                         W2=0.5*(U(IP,J,K,3)+U(I ,J,K ,3))
75         35              W1=0.5*(U(IM,J,K,3)+U(I ,J,K ,3))
77                         BC2=0.5*(UBC4(J,K,1)+UBC4(J,KM,1))
78                         BC1=0.5*(UBC3(J,K,1)+UBC3(J,KM,1))
80         61              BC_IN =0.5*DX1Q/RE*UBC3(J,K,3)
81                       >      +0.5*DX1*U1*0.5*UBC3(J,K,3)
82                       >      +0.5*DX1*W1*BC1
83                         BC_OUT=0.5*DX1Q/RE*UBC4(J,K,3)
84                       >      -0.5*DX1*U2*0.5*UBC4(J,K,3)
85                       >      -0.5*DX1*W2*BC2
88         28              BC=(1.-FJUM)*BC_DOWN(I)
89                       >   +(1.-FJUP)*BC_UP(I)
90                       >   +(1.-FIUM)*BC_IN
91                       >   +(1.-FIUP)*BC_OUT
93         78              RDUH3(I,J,K)=1./DT*U(I,J,K,3)
94                       >          -PRESSG3+VISCOS
95                       >          +BC
97                        ENDDO
98          4              DO 10 I=1,N1M
```

```
 99                      IP=I+1
100                      IM=I-1
103        4             FIUM=FIMV(I)
104                      FIUP=FIPA(I)
105
.................................................
182                10    CONTINUE
```

① rhs1에서 수행된 최적화와 마찬가지로 J, I, K 루프 실행을 K, J, I 루프로 변경하면서 메모리 접근이 연속적이 되도록 하고 있으며, 이 과정에서 J 인덱스와 무관한 VISCOS의 일부분과 BC_DOWN, BC_UP 계산을 J 루프 밖으로 이동시켜 불필요한 반복 계산을 줄이고 있다. FJUM, FJUP, FIUM, FIUP에 미리 계산된 것을 불러 쓰고 있는 것도 동일하다.

J. getduh1.f

A. TDMAI 호출 부분

< Original 코드 >

```
23                       DO 2 I=2,N1M
24                       IP=I+1
25                       IM=I-1
26                       IUM=I-IMU(I)
27                       IUP=IPA(I)-I
28                       DO 2 K=1,N3M
29                       DO 20 J=1,N2M
30                       JP=J+1
31                       JM=J-1
32        15             JUM=J-JMU(J)
33         2             JUP=JPA(J)-J
35      1763             V2=0.5*(U(2,I,JP,K)+U(2,IM,JP,K))
36                       V1=0.5*(U(2,I,J ,K)+U(2,IM,J ,K))
38      1609             APJ(J)=JUP*(
39                      >           JUM*(-0.5)*HP(J)/RE
40                      >         +(1-JUM)*(-0.5)/RE*2.0/H(2)/(H(1)+H(2))
41                      >      +0.5/DY(J)*V2/H(JP)*DY(J)/2.0
42                      >      )*DT
43       220             ACJ(J)= (JUM*JUP*0.5*HC(J)/RE
44                      >        +(1-JUM)*0.5/RE*2.0/H(1)/H(2)
45                      >        +(1-JUP)*0.5/RE*2.0/H(N2)/H(N2M)
46                      >        +0.5/DY(J)*(JUP*V2/H(JP)*DY(JP)/2.0
47                      >                   -JUM*V1/H(J )*DY(JM)/2.0)
48                      >        )*DT
49                      >       +1.0
50       150             AMJ(J)=JUM*(
51                      >           JUP*(-0.5)*HM(J)/RE
52                      >        +(1-JUP)*(-0.5)/RE*2.0/H(N2M)/(H(N2M)+H(N2))
53                      >      -0.5/DY(J)*V1/H(J)*DY(J)/2.0
54                      >      )*DT
```

```
55      12          R2(J)=RDUH1(I,J,K)*DT
56              20  CONTINUE
58                  CALL TDMA(AMJ,ACJ,APJ,R2,R2,1,N2M)
59                  DO 21 J=1,N2M
60    1077      21  UH(1,I,J,K)=R2(J)
61               2  CONTINUE
```

< Original 코드의 TDMA 서브루틴>

```
 4                  SUBROUTINE TDMA(A,B,C,R,X,NS,NF)
 5                  PARAMETER (M1=257,M2=65,M3=129)
 6                  REAL BET(M1),GAM(0:M1),A(M1),B(M1),C(M1),R(M1),X(M1)
 8      17          DO 10 J=NS,NF
 9     210          BET(J)=B(J)-A(J)*GAM(J-1)
10     620          GAM(J)=C(J)/BET(J)
11    1105          X(J)=(R(J)-A(J)*X(J-1))/BET(J)
12              10  CONTINUE
13       5          DO 20 J=NF,NS,-1
14     284          X(J)=X(J)-GAM(J)*X(J+1)
15       1      20  CONTINUE
16                  RETURN
17       7          END
```

< 최적화 코드 >

```
30                  DO 2 K=1,N3M
31                  DO 20 J=1,N2M
32       1          JP=J+1
33                  JM=J-1
36                  FJUM=FJMU(J)
37       2          FJUP=FJPA(J)
38                  DO 20 I=2,N1M
39                  IP=I+1
40                  IM=I-1
42       7          V2=0.5*(U(I,JP,K,2)+U(IM,JP,K,2))
43       1          V1=0.5*(U(I,J ,K,2)+U(IM,J ,K,2))
45      36          APJ(I,J)=FJUP*(
46                 >              FJUM*(-0.5)*HP(J)/RE
47                 >           +(1.-FJUM)*(-0.5)/RE*2.0/H(2)/(H(1)+H(2))
48                 >      +0.5/DY(J)*V2/H(JP)*DY(J)/2.0
49                 >      )*DT
50      52          ACJ(I,J)= (FJUM*FJUP*0.5*HC(J)/RE
51                 >         +(1.-FJUM)*0.5/RE*2.0/H(1)/H(2)
52                 >         +(1.-FJUP)*0.5/RE*2.0/H(N2)/H(N2M)
53                 >         +0.5/DY(J)*(FJUP*V2/H(JP)*DY(JP)/2.0
54                 >                          -FJUM*V1/H(J )*DY(JM)/2.0)
55                 >          )*DT
56                 >        +1.0
57      12          AMJ(I,J)=FJUM*(
58                 >              FJUP*(-0.5)*HM(J)/RE
59                 >         +(1.-FJUP)*(-0.5)/RE*2.0/H(N2M)/(H(N2M)+H(N2))
60                 >      -0.5/DY(J)*V1/H(J)*DY(J)/2.0
```

```
61                          >        )*DT
62          60               R2(I,J)=RDUH1(I,J,K)*DT
63                      20   CONTINUE
65                           CALL TDMAI(AMJ,ACJ,APJ,R2,R2,1,N2M,2,N1M)
66                           DO 21 J=1,N2M
67                           DO 21 I=2,N1M
68          51          21   UH(I,J,K,1)=R2(I,J)
69                       2   CONTINUE
```

<최적화 코드에서 사용한 TDMAI 서브루틴>

```
 4                           SUBROUTINE TDMAI(A,B,C,R,X,NS,NF,NIS,NIF)
 5                           include 'param.h'
 6                           REAL A(M1,M2),B(M1,M2),C(M1,M2),R(M1,M2),X(M1,M2)
 7                           REAL BET,GAM(M1,0:M2)
10                           J=NS
11                           DO I=NIS,NIF
12           1               BET=B(I,J)
13           1               GAM(I,J)=C(I,J)/BET
14           6               X(I,J)=R(I,J)/BET
15                           ENDDO
16                           DO 10 J=NS+1,NF
17           1               DO I=NIS,NIF
18          30               BET=B(I,J)-A(I,J)*GAM(I,J-1)
19          41               GAM(I,J)=C(I,J)/BET
20         317               X(I,J)=(R(I,J)-A(I,J)*X(I,J-1))/BET
21           3               ENDDO
22                        10 CONTINUE
23                           DO 20 J=NF-1,NS,-1
24                           DO I=NIS,NIF
25          53               X(I,J)=X(I,J)-GAM(I,J)*X(I,J+1)
26                           ENDDO
27                        20 CONTINUE
```

① I, K, J 순으로 반복 실행되는 루프 계산을 K, J, I 순으로 바꿔 배열 U와 UH, RDUH에 대한 접근이 연속적이 되도록 하였다.

② JUM, JUP을 계산하는 대신 앞에서 계산해 저장해둔 값을 FJUM과 FJUP으로 받아 사용하고 있으며, IUM과 IUP은 코드에서 사용되지 않기 때문에 최적화 코드에서는 삭제되었다.

③ 최적화 코드에서 루프 실행순서가 K, J, I로 바뀌면서 각 K, I에 대해 J루프 내에서 계산되던 1차원 배열 APJ, ACJ, AMJ, R2가 I루프 내에서 계산되기 위해 2차원 배열 APJ(I,J), ACJ(I,J), AMJ(I,J), R2(I,J)로 변경되었다. Original 코드에서 TDMA 서브루틴은 각 K, I인덱스에 대해 계산되는 1차원 배열 APJ(J), ACJ(J), AMJ(J), R2(J)를 입력으로 받아 결과를 R2(J)로 내놓는다. 최적화 코드에서는 2차원 배열을 입력, 출력 인수로 사용하므로 이를 위해 TDMA 대신 TDMAI 루틴을 만들어 사용하고 있다. TDMAI 내부에서 2차원 배열을 처리하므로 J의 범위뿐 아니라 I의 범위도 인수

로 넘겨 주었다.

④ 전체적으로 I, K루프의 반복 회수만큼 호출되는 TDMA 루틴과 비교해 I루프 반복 실행을 루틴 내부에서 처리하는 TDMAI는 K루프의 반복만큼만 호출되어 호출 회수가 감소되었고 그만큼 한 번에 더 많은 양의 일을 처리함으로써 코드 효율성을 높이고 있다.

④ TDMAI에서 J=NS일 때 GAM(I,J-1)=0, X(I,J-1)=0이 되어 루프 내의 A(I,J)*GAM(I,J-1), A(I,J)*X(I,J-1)의 반복 연산이 무의미하므로 J=NS인 경우를 루프 밖에서 따로 처리하고 있다. 그리고 최적화 코드에서는 임시 변수인 1차원 배열 BET() 대신 스칼라 변수 BET를 이용해 불필요한 메모리 낭비를 막고 있다.

B. TDMAJ 호출 부분

< Original 코드 >

```
63                    DO 1 J=1,N2M
64                    DO 1 K=1,N3M
65          5         DO 10 I=2,N1M
66          8         IP=I+1
67                    IM=I-1
68         29         IUM=I-IMU(I)
69         13         IUP=IPA(I)-I
71          9         U2=0.5*(U(1,IP,J,K)+U(1,I ,J,K))
72         66         U1=0.5*(U(1,I ,J,K)+U(1,IM,J,K))
73         29         API(I)= (-0.5*DX1Q/RE
74                   >           +DX1*U2*0.5)*IUP
75                   >         *DT
76         30         ACI(I)= 1.0+(
77                   >       +DX1Q/RE
78                   >       +DX1*(U2*0.5-U1*0.5)
79                   >       )*DT
81         74         AMI(I)= (-0.5*DX1Q/RE
82                   >           -DX1*U1*0.5)*IUM
83                   >         *DT
85         28         R1(I)=UH(1,I,J,K)
86                10  CONTINUE
88                    CALL TDMA(AMI,ACI,API,R1,R1,2,N1M)
89                    DO 11 I=2,N1M
90         54     11  UH(1,I,J,K)=R1(I)
91                 1  CONTINUE
```

< 최적화 코드 >

```
71                    DO 1 K=1,N3M
72                    DO 10 J=1,N2M
73                    DO 10 I=2,N1M
74                    IP=I+1
75                    IM=I-1
```

```
 78          1           FIUM=FIMU(I)
 79          1           FIUP=FIPA(I)
 81         31           U2=0.5*(U(IP,J,K,1)+U(I ,J,K,1))
 82         10           U1=0.5*(U(I ,J,K,1)+U(IM,J,K,1))
 83         11           API(I,J)= (-0.5*DX1Q/RE
 84                     >           +DX1*U2*0.5)*FIUP
 85                     >         *DT
 86         26           ACI(I,J)= 1.0+(
 87                     >       +DX1Q/RE
 88                     >       +DX1*(U2*0.5-U1*0.5)
 89                     >       )*DT
 91         21           AMI(I,J)= (-0.5*DX1Q/RE
 92                     >           -DX1*U1*0.5)*FIUM
 93                     >         *DT
 95         18           R1(I,J)=UH(I,J,K,1)
 96                 10   CONTINUE
 98                      CALL TDMAJ(AMI,ACI,API,R1,R1,2,N1M,1,N2M)
 99                      DO 11 J=1,N2M
100                      DO 11 I=2,N1M
101         35      11   UH(I,J,K,1)=R1(I,J)
102                  1   CONTINUE
```

< 최적화 코드에서 사용한 TDMAJ 서브루틴 >

```
  4                      SUBROUTINE TDMAJ(A,B,C,R,X,NS,NF,NJS,NJF)
  5                      include 'param.h'
  6                      REAL A(M1,*),B(M1,*),C(M1,*),R(M1,*),X(M1,*)
  7                      REAL BET,GAM(0:M1,M2)
  8                      DO J=NJS,NJF-1,2
  9                          I=NS
 10                          BET1=1./B(I,J)
 11                          BET2=1./B(I,J+1)
 12          1               GAM(I,J)=C(I,J)*BET1
 13                          GAM(I,J+1)=C(I,J+1)*BET2
 14          1               X(I,J)=R(I,J)*BET1
 15          1               X(I,J+1)=R(I,J+1)*BET2
 16                      DO I=NS+1,NF
 17          2               BET1=1./(B(I,J)-A(I,J)*GAM(I-1,J))
 18        137               BET2=1./(B(I,J+1)-A(I,J+1)*GAM(I-1,J+1))
 19          9               GAM(I,J)=C(I,J)*BET1
 20         28               GAM(I,J+1)=C(I,J+1)*BET2
 21        191               X(I,J)=(R(I,J)-A(I,J)*X(I-1,J))*BET1
 22         68               X(I,J+1)=(R(I,J+1)-A(I,J+1)*X(I-1,J+1))*BET2
 23                      ENDDO
 24                      DO I=NF-1,NS,-1
 25        109               X(I,J)=X(I,J)-GAM(I,J)*X(I+1,J)
 26        102               X(I,J+1)=X(I,J+1)-GAM(I,J+1)*X(I+1,J+1)
 27                      ENDDO
 28          1           ENDDO
 30                      IF(MOD(NJF-NJS+1,2).eq.1) THEN
 31                          J=NJF
 32                          I=NS
 33                          BET1=1./B(I,J)
 34                          GAM(I,J)=C(I,J)*BET1
```

```
35                              X(I,J)=R(I,J)*BET1
36                          DO I=NS+1,NF
37                              BET1=1./(B(I,J)-A(I,J)*GAM(I-1,J))
38                              GAM(I,J)=C(I,J)*BET1
39          2                   X(I,J)=(R(I,J)-A(I,J)*X(I-1,J))*BET1
40                          ENDDO
41                          DO I=NF-1,NS,-1
42                              X(I,J)=X(I,J)-GAM(I,J)*X(I+1,J)
43                          ENDDO
44                          ENDIF
46                          RETURN
47                          END
```

① 전체적으로 TDMAI를 호출하는 부분과 동일한 최적화 과정을 거쳤다. 즉, 루프 순서를 바꿔 배열 접근이 연속적이 되도록 하였고, 이 과정에서 1차원 배열 AMI, ACI, API, R1 등이 2차원 배열로 변경되었다. 1차원 배열을 처리하는 TDMA대신 2차원으로 변경된 배열을 처리하는 TDMAJ 루틴을 만들어 사용하고 있다. TDMAJ는 TDMA와 비교해 J루프 반복을 내부에서 처리하게 되며, 따라서 J의 범위가 인수에 추가되었다.

② TDMAJ 루틴에서는 계산에 의존성이 없는 바깥쪽 루프 J에 대해 factor 2의 unrolling을 시도 하였다.

```
DO 20 J = NJS,NJF-1,2
    DO I = NF-1,NS,-1
        X(I,J)=X(I,J)-GAM(I,J)*X(I+1,J)
        X(I,J+1)=X(I,J+1)-GAM(I,J+1)*X(I+1,J+1)
    ENDDO
20 CONTINUE
```

③ I루프 내에서 불필요한 연산을 줄이기 위해 I=NS일 때를 루프 밖에서 따로 처리하고 있는 점, 그리고 임시 변수인 1차원 배열 BET() 대신 스칼라 변수 BET를 이용해 불필요한 메모리 낭비를 줄이고 있는 점 등은 TDMAI에서 수행된 최적화와 동일하다.

C. CTDMA3I 호출 부분

< Original 코드 >

```
93                          DO 3 I=2,N1M
94                          IP=I+1
95                          IM=I-1
96                          IUM=I-IMU(I)
97                          IUP=IPA(I)-I
98                          DO 3 J=1,N2M
99           1               DO 30 K=1,N3M
100        492               KP=KPA(K)
101                          KM=KMA(K)
103         51               W2=0.5*(U(3,IM,J,KP)+U(3,I,J,KP))
```

```
104          41              W1=0.5*(U(3,IM,J,K )+U(3,I,J,K ))
106         730              APK(K)=(
107                         >          -0.5*DX3Q/RE
108                         >          +0.5*DX3*W2*0.5
109                         >          )*DT
110          18              ACK(K)=1.+(
111                         >          +DX3Q/RE
112                         >          +0.5*DX3*(W2*0.5-W1*0.5)
113                         >        )*DT
114         343              AMK(K)=(
115                         >          -0.5*DX3Q/RE
116                         >          -0.5*DX3*W1*0.5
117                         >           )*DT
118        1163              R3(K)=UH(1,I,J,K)
119                     30   CONTINUE
120           1              CALL CTDMA3(AMK,ACK,APK,R3,UH,I,J,1,N3M)
121                      3   CONTINUE
```

< Original 코드의 CTDMA3 서브루틴 >

```
 4                           SUBROUTINE CTDMA3(A,B,C,R,X,I,J,NV,N)
 5                           PARAMETER (M1=257,M2=65,M3=129)
 6                           COMMON/DIM/N1,N2,N3,N1M,N2M,N3M
 7                           REAL A(M3),B(M3),C(M3),R(M3)
 8                           REAL X(3,0:M1,0:M2,0:M3)
 9                           REAL BET(M3),GAM(M3)
10                           REAL P(M3),Q(M3)
12                           BET(1)=B(1)
13           2               GAM(1)=C(1)/BET(1)
14           4               X(NV,I,J,1)=R(1)/BET(1)
15                           P(1)=C(N)
16                           Q(1)=A(1)/BET(1)
18                           DO 10 K=2,N-1
19          23               BET(K)=B(K)-A(K)*GAM(K-1)
20          97               GAM(K)=C(K)/BET(K)
21         558               P(K)=-P(K-1)*GAM(K-1)
22         126               Q(K)=-A(K)/BET(K)*Q(K-1)
23         214         10    X(NV,I,J,K)=(R(K)-A(K)*X(NV,I,J,K-1))/BET(K)
25           7               P(N-1)=A(N)-P(N-2)*GAM(N-2)
26           1               Q(N-1)=(C(N-1)-A(N-1)*Q(N-2))/BET(N-1)
28                           X(NV,I,J,N)=R(N)
29                           P(N)=B(N)
30                           DO 20 K=1,N-1
31         132               X(NV,I,J,N)=X(NV,I,J,N)-P(K)*X(NV,I,J,K)
32         112         20    P(N)=P(N)-P(K)*Q(K)
33                           X(NV,I,J,N)=X(NV,I,J,N)/P(N)
35          14               GAM(N-1)=0.0
36                           DO 30 K=N-1,1,-1
37         503         30    X(NV,I,J,K)=X(NV,I,J,K)-GAM(K)*X(NV,I,J,K+1)-Q(K)*X(NV,I,J,N)
```

< 최적화 코드 >

```
104                    DO 3 J=1,N2M
105                    DO 30 K=1,N3M
106                    KP=KPA(K)
107                    KM=KMA(K)
108                    DO 30 I=2,N1M
109                    IP=I+1
110                    IM=I-1
112      51            W2=0.5*(U(IM,J,KP,3)+U(I,J,KP,3))
113      10            W1=0.5*(U(IM,J,K ,3)+U(I,J,K ,3))
115      27            APK(I,K)=(
116                   >         -0.5*DX3Q/RE
117                   >         +0.5*DX3*W2*0.5
118                   >         )*DT
119      20            ACK(I,K)=1.+(
120                   >         +DX3Q/RE
121                   >         +0.5*DX3*(W2*0.5-W1*0.5)
122                   >        )*DT
123      28            AMK(I,K)=(
124                   >         -0.5*DX3Q/RE
125                   >         -0.5*DX3*W1*0.5
126                   >          )*DT
127      32            R3(I,K)=UH(I,J,K,1)
128              30    CONTINUE
129                    CALL CTDMA3I(AMK,ACK,APK,R3,UH(0,0,0,1),J,N3M,2,N1M)
130               3    CONTINUE
```

< 최적화 코드의 CTDMA3I 서브루틴 >

```
 4                     SUBROUTINE CTDMA3I(A,B,C,R,X,J,N,NIS,NIF)
 5                     include 'param.h'
 6                     COMMON/DIM/N1,N2,N3,N1M,N2M,N3M
 7                     REAL A(M1,M3),B(M1,M3),C(M1,M3),R(M1,M3)
 8                     REAL X(0:M1,0:M2,0:M3)
 9                     REAL GAM(M1,M3)
10                     REAL P(M1,M3),Q(M1,M3)
12                     DO I=NIS,NIF
13        1            BET=1./B(I,1)
14        3            GAM(I,1)=C(I,1)*BET
15        2            X(I,J,1)=R(I,1)*BET
16                     P(I,1)=C(I,N)
17                     Q(I,1)=A(I,1)*BET
18                     ENDDO
20                     DO 10 K=2,N-1
21                     DO I=NIS,NIF
22      125            BET=1./(B(I,K)-A(I,K)*GAM(I,K-1))
23       69            GAM(I,K)=C(I,K)*BET
24      207            P(I,K)=-P(I,K-1)*GAM(I,K-1)
25       66            Q(I,K)=-A(I,K)*BET*Q(I,K-1)
26      255            X(I,J,K)=(R(I,K)-A(I,K)*X(I,J,K-1))*BET
27                     ENDDO
28               10    CONTINUE
30        1            DO I=NIS,NIF
31        1            BET=1./(B(I,N-1)-A(I,N-1)*GAM(I,N-2))
32                     P(I,N-1)=A(I,N)-P(I,N-2)*GAM(I,N-2)
```

```
33        2          Q(I,N-1)=(C(I,N-1)-A(I,N-1)*Q(I,N-2))*BET
35                   X(I,J,N)=R(I,N)
36                   P(I,N)=B(I,N)
37                   ENDDO
38                   DO 20 K=1,N-1
39                   DO I=NIS,NIF
40      112          X(I,J,N)=X(I,J,N)-P(I,K)*X(I,J,K)
41       39          P(I,N)=P(I,N)-P(I,K)*Q(I,K)
42                   ENDDO
43             20    CONTINUE
44                   DO I=NIS,NIF
45        1          X(I,J,N)=X(I,J,N)/P(I,N)
47        1          GAM(I,N-1)=0.0
48                   ENDDO
49        1          DO 30 K=N-1,1,-1
50                   DO I=NIS,NIF
51      162          X(I,J,K)=X(I,J,K)-GAM(I,K)*X(I,J,K+1)-Q(I,K)*X(I,J,N)
52                   ENDDO
53             30    CONTINUE
```

① Original 코드에서 I, J, K 순으로 루프 반복이 실행되는 것을 J, K, I로 변경하여 배열 U, UH에 대한 접근 연속성을 높였다.

② 각 I, J에 대해 K루프 내에서 계산되던 1차원 배열 APK, ACK, AMK, R3가 I루프 내에서 계산되기 위해 2차원 배열 APK(I,K), ACK(I,K), AMK(I,K), R3(I,K)로 변경되었다. 이로 인해 1차원 배열을 처리하는 루틴 CTDMA3(AMK,ACK,APK,R3,UH,I,J,1,N3M)는 2차원 배열을 받아 처리하는 새로운 루틴 CTDMA3I(AMK,ACK,APK,R3,UH(0,0,0,1), J,N3M,2,N1M)로 변경되었다.

③ 루프 I, J의 반복 회수만큼 반복 호출되는 CTDMA3와 비교해 CTDMA3I는 J루프의 반복만큼만 호출되어 호출 회수가 줄었고 그만큼 한 번에 더 많은 양의 일을 처리함으로써 코드 효율성을 높이고 있다. 루프 I는 루틴 CTDMA3I 내부에서 처리되고 있으며, 이를 위해 I의 범위가 인수로 전달된다.

④ 루틴 CTDMA3에서 아래와 같은 부분은 데이터 접근이 불연속적이어서 많은 캐시 실패를 일으키게 된다.

```
      DO 10 K = 1,N-1
         ...
   10 X(NV,I,J,K)=(R(K)-A(K)*X(NV,I,J,K-1))/BET(K)
```

이것이 최적화 코드의 루틴 CTDMA3I에서는 다음과 같이 처리된다.

```
      DO 10 K=2,N-1
         DO I=NIS,NIF
         ...
         X(I,J,K)=(R(I,K)-A(I,K)*X(I,J,K-1))*BET
      ENDDO
 10   CONTINUE
```

I루프가 K루프 내부에서 반복됨으로써 데이터 접근의 연속성이 더 높아지고 있으며,

4차원 배열 UH전체를 받지 않고 3차원 배열 X로 받아 처리하면서 접근 연속성을 높이며 아울러 불필요한 메모리 낭비를 줄이고 있다.

K. getduh2.f

A. TDMAI 호출 부분

< Original 코드 >

```
23                  DO 2 I=1,N1M
24                   IP=I+1
25                   IM=I-1
26                   IUM=I-IMV(I)
27                   IUP=IPA(I)-I
28                  DO 2 K=1,N3M
29                   KP=KPA(K)
30                   KM=KMA(K)
31                  DO 20 J=2,N2M
32                   JP=J+1
33                   JM=J-1
34     100           JUM=J-JMV(J)
35                   JUP=JPA(J)-J
37      21           V2=0.5*(U(2,I,JP,K)+U(2,I,J ,K))
38      23           V1=0.5*(U(2,I,J ,K)+U(2,I,JM,K))
40     114           APJ(J)=JUP*(
41                  >      -0.5*DYP(J)/RE
42                  >      +1.0/H(J)*V2*0.5
43                  >      )*DT
44    1426           ACJ(J)=1.0+(
45                  >      +0.5*DYC(J)/RE
46                  >      +1.0/H(J)*(V2*0.5-V1*0.5)
47                  >      )*DT
48     157           AMJ(J)=JUM*(
49                  >      -0.5*DYM(J)/RE
50                  >      -1.0/H(J)*V1*0.5
51                  >      )*DT
53             C     M21UH
54     420           V2=0.5*(U(2,IP,J,K)+U(2,I ,J,K))
55    1735           V1=0.5*(U(2,I ,J,K)+U(2,IM,J,K))
56    1082           UH2=1.0/H(J)*(DY(J)/2.*UH(1,IP,JM,K)+DY(JM)/2.*UH(1,IP,J,K))
57      14           UH1=1.0/H(J)*(DY(J)/2.*UH(1,I ,JM,K)+DY(JM)/2.*UH(1,I ,J,K))
58      20           RM21UH=0.5*DX1*(IUP*V2*UH2-IUM*V1*UH1)
60      57           R2(J)=DT*(RDUH2(I,J,K)-RM21UH)
62       6       20  CONTINUE
63                  CALL TDMA(AMJ,ACJ,APJ,R2,R2,2,N2M)
64       2           DO 21 J=2,N2M
65      26       21  UH(2,I,J,K)=R2(J)
66                2  CONTINUE
```

< 최적화 코드 >

```
30                        DO 2 K=1,N3M
31                        KP=KPA(K)
32                        KM=KMA(K)
33                        DO 20 J=2,N2M
34                        JP=J+1
35                        JM=J-1
38                        FJUM=FJMV(J)
39          2             FJUP=FJPA(J)
40          1             DO 20 I=1,N1M
41                        IP=I+1
42                        IM=I-1
45                        FIUM=FIMV(I)
46         12             FIUP=FIPA(I)
48          6             V2=0.5*(U(I,JP,K,2)+U(I,J ,K,2))
49         13             V1=0.5*(U(I,J ,K,2)+U(I,JM,K,2))
51         23             APJ(I,J)=FJUP*(
52                       >         -0.5*DYP(J)/RE
53                       >         +1.0/H(J)*V2*0.5
54                       >         )*DT
55         35             ACJ(I,J)=1.0+(
56                       >         +0.5*DYC(J)/RE
57                       >         +1.0/H(J)*(V2*0.5-V1*0.5)
58                       >         )*DT
59          7             AMJ(I,J)=FJUM*(
60                       >         -0.5*DYM(J)/RE
61                       >         -1.0/H(J)*V1*0.5
62                       >         )*DT
64                 C      M21UH
65          5             V2=0.5*(U(IP,J,K,2)+U(I ,J,K,2))
66         15             V1=0.5*(U(I ,J,K,2)+U(IM,J,K,2))
67         43             UH2=1.0/H(J)*(DY(J)/2.*UH(IP,JM,K,1)+DY(JM)/2.*UH(IP,J,K,1))
68         12             UH1=1.0/H(J)*(DY(J)/2.*UH(I ,JM,K,1)+DY(JM)/2.*UH(I ,J,K,1))
69         33             RM21UH=0.5*DX1*(FIUP*V2*UH2-FIUM*V1*UH1)
71         31             R2(I,J)=DT*(RDUH2(I,J,K)-RM21UH)
73          2        20   CONTINUE
74                        CALL TDMAI(AMJ,ACJ,APJ,R2,R2,2,N2M,1,N1M)
75                        DO 21 J=2,N2M
76          1             DO 21 I=1,N1M
77         69        21   UH(I,J,K,2)=R2(I,J)
78                    2   CONTINUE
```

① getduh1.f에서의 TDMAI 호출과 동일한 최적화 과정이다.

B. TDMAJ 호출 부분

< Original 코드 >

```
68                        DO 1 J=2,N2M
69                        JP=J+1
70                        JM=J-1
71                        DO 1 K=1,N3M
72                        KP=KPA(K)
```

```
 73                     KM=KMA(K)
 74          5          DO 10 I=1,N1M
 75          2          IP=I+1
 76                     IM=I-1
 77          1          IUM=I-IMV(I)
 78         20          IUP=IPA(I)-I
 80         28          U2=1.0/H(J)*(DY(J)/2.0*U(1,IP,JM,K)+DY(JM)/2.0*U(1,IP,J,K))
 81         68          U1=1.0/H(J)*(DY(J)/2.0*U(1,I ,JM,K)+DY(JM)/2.0*U(1,I ,J,K))
 82         12          API(I)=IUP*(
 83                    >          -0.5*DX1Q/RE
 84                    >          +0.5*DX1*U2*0.5
 85                    >        )*DT
 86         44          ACI(I)=1.0+(
 87                    >              +DX1Q/RE
 88                    >              +0.5*DX1*(U2*0.5-U1*0.5)
 89                    >       )*DT
 90         80          AMI(I)=IUM*(
 91                    >          -0.5*DX1Q/RE
 92                    >          -0.5*DX1*U1*0.5
 93                    >        )*DT
 95          7          R1(I)=UH(2,I,J,K)
 96                 10  CONTINUE
 97                     CALL TDMA(AMI,ACI,API,R1,R1,1,N1M)
 98                     DO 11 I=1,N1M
 99         57      11  UH(2,I,J,K)=R1(I)
100                  1  CONTINUE
```

< 최적화 코드 >

```
 80                     DO 1 K=1,N3M
 81                     KP=KPA(K)
 82                     KM=KMA(K)
 83                     DO 10 J=2,N2M
 84                     JP=J+1
 85                     JM=J-1
 86          1          DO 10 I=1,N1M
 87                     IP=I+1
 88                     IM=I-1
 91                     FIUM=FIMV(I)
 92          7          FIUP=FIPA(I)
 94         42          U2=1.0/H(J)*(DY(J)/2.0*U(IP,JM,K,1)+DY(JM)/2.0*U(IP,J,K,1))
 95         13          U1=1.0/H(J)*(DY(J)/2.0*U(I ,JM,K,1)+DY(JM)/2.0*U(I ,J,K,1))
 96         13          API(I,J)=FIUP*(
 97                    >          -0.5*DX1Q/RE
 98                    >          +0.5*DX1*U2*0.5
 99                    >        )*DT
100         40          ACI(I,J)=1.0+(
101                    >              +DX1Q/RE
102                    >              +0.5*DX1*(U2*0.5-U1*0.5)
103                    >       )*DT
104         18          AMI(I,J)=FIUM*(
105                    >          -0.5*DX1Q/RE
```

```
106                    >          -0.5*DX1*U1*0.5
107                    >         )*DT
109        3            R1(I,J)=UH(I,J,K,2)
110                  10 CONTINUE
111                     CALL TDMAJ(AMI,ACI,API,R1,R1,1,N1M,2,N2M)
112                     DO 11 J=2,N2M
113                     DO 11 I=1,N1M
114       32         11 UH(I,J,K,2)=R1(I,J)
115                   1 CONTINUE
```

① getduh1.f에서의 TDMAJ 호출과 동일한 최적화 과정이다.

C. CTDMA3I 호출 부분

< Original 코드 >

```
102                     DO 3 I=1,N1M
103                     IP=I+1
104                     IM=I-1
105                     IUM=I-IMV(I)
106                     IUP=IPA(I)-I
107                     DO 3 J=2,N2M
108                     JP=J+1
109                     JM=J-1
110                     DO 30 K=1,N3M
111        8            KP=KPA(K)
112                     KM=KMA(K)
114     1178            W2=1.0/H(J)*(DY(J)/2.0*U(3,I,JM,KP)+DY(JM)/2.0*U(3,I,J,KP))
115      650            W1=1.0/H(J)*(DY(J)/2.0*U(3,I,JM,K )+DY(JM)/2.0*U(3,I,J,K ))
117       55            APK(K)=(
118                    >        -0.5*DX3Q/RE
119                    >        +0.5*DX3*W2/2.0
120                    >         )*DT
121     1038            ACK(K)=1.+(
122                    >        +DX3Q/RE
123                    >        +0.5*DX3*(W2/2.0-W1/2.0)
124                    >         )*DT
125       15            AMK(K)=(
126                    >        -0.5*DX3Q/RE
127                    >        -0.5*DX3*W1/2.0
128                    >         )*DT
129       33            R3(K)=UH(2,I,J,K)
130                  30 CONTINUE
131                     CALL CTDMA3(AMK,ACK,APK,R3,UH,I,J,2,N3M)
132                   3 CONTINUE
```

< 최적화 코드 >

```
117                     DO 3 J=2,N2M
118                     JP=J+1
```

```
119                         JM=J-1
120        1                DO 30 K=1,N3M
121                         KP=KPA(K)
122                         KM=KMA(K)
123                         DO 30 I=1,N1M
124                         IP=I+1
125                         IM=I-1
127       69                W2=1.0/H(J)*(DY(J)/2.0*U(I,JM,KP,3)+DY(JM)/2.0*U(I,J,KP,3))
128       31                W1=1.0/H(J)*(DY(J)/2.0*U(I,JM,K ,3)+DY(JM)/2.0*U(I,J,K ,3))
130        5                APK(I,K)=(
131                        >       -0.5*DX3Q/RE
132                        >       +0.5*DX3*W2/2.0
133                        >        )*DT
134       52                ACK(I,K)=1.+(
135                        >       +DX3Q/RE
136                        >       +0.5*DX3*(W2/2.0-W1/2.0)
137                        >        )*DT
138       35                AMK(I,K)=(
139                        >       -0.5*DX3Q/RE
140                        >       -0.5*DX3*W1/2.0
141                        >        )*DT
142       17                R3(I,K)=UH(I,J,K,2)
143                     30  CONTINUE
144                         CALL CTDMA3I(AMK,ACK,APK,R3,UH(0,0,0,2),J,N3M,1,N1M)
145                      3  CONTINUE
```

① getduh1.f에서의 CTDMA3I 호출과 동일한 최적화 과정이다.

L. tdma.f

$$\begin{bmatrix} B_1 & C_1 & & \cdots\cdots & & & \\ A_2 & B_2 & C_2 & \ldots\ldots & & & \\ & & & \cdots\cdots & & & \\ & & & \cdots\cdots & & & \\ & & & \cdots\cdots & A_{NF-1} & B_{NF-1} & C_{NF-1} \\ & & & \cdots\cdots & 0 & A_{NF} & B_{NF} \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ \vdots \\ \vdots \\ X_{NF-1} \\ X_{NF} \end{bmatrix} = \begin{bmatrix} R_1 \\ R_2 \\ \vdots \\ \vdots \\ R_{NF-1} \\ R_{NF} \end{bmatrix}$$

위와 같이 Banded Tri-diagonal matrix로 구성된 선형방정식을 풀기 위해 Tomas-algorithm을 사용한 루틴이다. 위의 선형방정식은 아래 프로그램에서 사용된 변수를 그대로 적용하였다.

Original 코드에서는 TDMA 루틴 하나를 이용해 이 부분을 처리하고 있는데, 최적화 코드에서는 이 TDMA 루틴을 TDMAI, TDMAJ, TDMAI0 등으로 분리하여, 최적화를 수행하고 있

다. TDMAI와 TDMAI0는 앞서 다루었으므로 여기에서는 루틴 TDMAI0만 언급한다.

A. tdmai0.f

< 최적화 코드 >

```
4                        SUBROUTINE TDMAI0(A,B,C,R,X,NS,NF,NIS,NIF)
5                        include 'param.h'
6                        REAL A(0:M1,M2),B(0:M1,M2),C(0:M1,M2),R(0:M1,M2),X(0:M1,M2)
7                        REAL BET,GAM(0:M1,0:M2)
10                       J=NS
11                       DO I=NIS,NIF
12                       BET=B(I,J)
13                       GAM(I,J)=C(I,J)/BET
14                       X(I,J)=R(I,J)/BET
15                       ENDDO
16                       DO 10 J=NS+1,NF
17                       DO I=NIS,NIF
18        11              BET=B(I,J)-A(I,J)*GAM(I,J-1)
19        16              GAM(I,J)=C(I,J)/BET
20        87              X(I,J)=(R(I,J)-A(I,J)*X(I,J-1))/BET
21                       ENDDO
22                    10 CONTINUE
23                       DO 20 J=NF-1,NS,-1
24                       DO I=NIS,NIF
25        12              X(I,J)=X(I,J)-GAM(I,J)*X(I,J+1)
26                       ENDDO
27                    20 CONTINUE
30                       RETURN
31                       END
```

① 서브루틴 TAKEDP에서 FFT계산 후 호출 하는 루틴으로, TDMAI와 동일한 루틴이다. 단지, 루틴에서 처리되는 2차원 배열 A, B, C, R, X의 첫 번째 차원 시작 인덱스가 1이 아니라 0부터 시작된다는 것만이 다르다.

M. rhsdp.f

< Original 코드 >

```
19                       DO 10 J=1,N2M
20                       JP=J+1
21                       JM=J-1
22                       JUM=J-JMU(J)
23                       JUP=JPA(J)-J
25                       DO 10 I=1,N1M
26         7              IP=I+1
27                        IM=I-1
28                        IUM=I-IMV(I)
29         4              IUP=IPA(I)-I
31                       DO 10 K=1,N3M
```

```
32      2          KP=KPA(K)
33                 KM=KMA(K)
35   1000          DIVUH=(IUP*UH(1,IP,J,K)-IUM*UH(1,I,J,K))*DX1
36               >      +(JUP*UH(2,I,JP,K)-JUM*UH(2,I,J,K))/DY(J)
37               >      +(UH(3,I,J,KP)-UH(3,I,J,K))*DX3
39     46          CBC=(1-JUM)*UBC1(2,I,K)/DY(J)
40               >   -(1-JUP)*UBC2(2,I,K)/DY(J)
41               >   +(1-IUM)*UBC3(1,J,K)*DX1
42               >   -(1-IUP)*UBC4(1,J,K)*DX1
43    138          RDP(I,J,K)=(DIVUH-CBC)/DT
44      3       10 CONTINUE
```

< 최적화 코드 >

```
21                 DO 10 K=1,N3M
22                 KP=KPA(K)
23                 KM=KMA(K)
25                 DO 10 J=1,N2M
26                 JP=J+1
27                 JM=J-1
30                 FJUM=FJMU(J)
31                 FJUP=FJPA(J)
33                 DO 10 I=1,N1M
34                 IP=I+1
35                 IM=I-1
38     20          FIUM=FIMV(I)
39      6          FIUP=FIPA(I)
41     66          DIVUH=(FIUP*UH(IP,J,K,1)-FIUM*UH(I,J,K,1))*DX1
42               >      +(FJUP*UH(I,JP,K,2)-FJUM*UH(I,J,K,2))/DY(J)
43               >      +(UH(I,J,KP,3)-UH(I,J,K,3))*DX3
45     55          CBC=(1.-FJUM)*UBC1(I,K,2)/DY(J)
46               >   -(1.-FJUP)*UBC2(I,K,2)/DY(J)
47               >   +(1.-FIUM)*UBC3(J,K,1)*DX1
48               >   -(1.-FIUP)*UBC4(J,K,1)*DX1
49     15          RDP(I,J,K)=(DIVUH-CBC)/DT
50              10 CONTINUE
```

① J, I, K순으로 반복 실행되는 루프 계산을 K, J, I 순으로 바꿔 배열 UH, RDP에 대한 접근이 연속적이 되도록 하였다. 아울러 UBC1(2,I,K), UBC2(2,I,K), UBC3(1,J,K), UBC4(1,J,K) 배열도 UBC1(I,K,2), UBC2(I,K,2), UBC3(J,K,1), UBC4(J,K,1)으로 변경해 접근 연속성을 높이고 있다.

② IUM, IUP, JUM, JUP을 계산하는 대신 앞에서 계산해 저장해둔 값을 FIUM, FIUP, FJUM과 FJUP으로 받아 사용하고 있다.

N. takedp.f

이 부분은 내용이 길어 5개 부분으로 나눠 비교 설명한다.

< Original 코드 >

```
29                 C      Cosine transform of RDP in X1 direction
31                        IPC(0)=0      ! Initialize
32                        DO 100 J=1,N2M
33                        DO 100 K=1,N3M
34                        DO L=0,N1M-1
35          56            R1(L)=RDP(L+1,J,K)
36                        END DO
37                        CALL DDCT(N1M,-1,R1,IPC,WC)
38                        DO   L=0,N1M-1
39          48            CRDP(L,J,K)=R1(L)
40                        END DO
41                 100    CONTINUE
44                 C      FFT of RDP in X3 direction
46                        IPF(0)=0      ! Initialize
47                        DO 110 L=0,N1M-1
48                        DO 110 J=1,N2M
49           9            DO K=0,N3M-1
50         704            R2(K)=CRDP(L,J,K+1)
51                        END DO
52                        CALL RDFT(N3M,1,R2,IPF,WF)
53                        DO 111 M=0,N3M/2-1
54         146            FRDP_R(L,J,M)=R2(M*2)
55         434      111   FRDP_I(L,J,M)=R2(M*2+1)
56          19            FRDP_R(L,J,N3M/2)=R2(1)
57                 110    CONTINUE
```

< 최적화 코드 >

```
26                 C      Cosine transform of RDP in X1 direction
28                        DO 100 K=1,N3M
29                        call cosfftj_d(RDP(1,1,K),M1,CRDP(0,1,K),M1+1,2*N1M,N2M,1.d0)
30                 100    CONTINUE
33                 C      FFT of RDP in X3 direction
35                        DO 110 J=1,N2M
36           1            DO 110 LS=0,N1M-1,IBLK
37           1                    LE=MIN(LS+IBLK-1,N1M-1)
38                 c===
39           1            DO K=0,N3M-1
40                        DO L=LS,LE
41                            L1=L-LS
42          63                R2(K,L1)=CRDP(L,J,K+1)
43                        ENDDO
44           3            ENDDO
46                        call rcfft_d(r2,n3m+2,r2,n3m+2,n3m,le-ls+1,-1,1.d0)
48                        DO M=0,N3M/2-1
49           2            DO L=LS,LE
50                            L1=L-LS
51          38                FRDP_R(L,J,M)=R2(M*2,L1)
52          25                FRDP_I(L,J,M)=R2(M*2+1,L1)
53                        ENDDO
```

```
54      1         ENDDO
55                DO L=LS,LE
56                   L1=L-LS
57                   FRDP_R(L,J,N3M/2)=R2(2*(N3M/2),L1)
58                ENDDO
59           c===
60           110  CONTINUE
```

① Original 코드에서 배열 RDP에 대한 코사인 변환을 수행하는 DDCT루틴 대신 최적화된 cosfftj_d 루틴을 사용하고 있다. 최적화 코드에서는 임시 변수(original 코드의 R1)를 사용하지 않고 RDP를 직접 cosfftj_d로 전달해 그 결과를 CRDP로 직접 리턴받고 있다. DDCT가 1차원 배열을 처리하는 반면 cosfftj_d는 2차원 배열에 대해 코사인 변환을 수행해 한 번에 더 많은 계산을 수행하도록 하고 있다.

② L, J루프 반복을 J, L루프 반복으로 변경하고 루프 L을 크기 IBLK로 block화 하여 처리하고 있다. original 코드에서 루틴 RDFT가 1차원 배열 R2(K)를 처리하는 반면, 최적화 코드에서의 루틴 rcfft_d는 2차원 배열 R2(N3M,IBLK)를 처리하고 있다.

< Original 코드 >

```
60           C    Solution of Poisson eq. ; REAL part
61
62                DO 70 L=0,N1M-1
63                DO 70 M=0,N3M/2
64
65                DO 71 J=1,N2M
66      7         CMJ(J)=PMJ(J)
67    181         CCJ(J)=PCJ(J)-AK3(M)-AK1(L)
68      5         CPJ(J)=PPJ(J)
69     13         R(J)=FRDP_R(L,J,M)
70            71  CONTINUE
72           C    Special treatment to remove singularity when l=k=0
73
74                IF (L.EQ.0.AND.M.EQ.0) THEN
75                   CPJ(N2M-1)=0.0
76                   CALL TDMA(CMJ,CCJ,CPJ,R,R,1,N2M-1)
77                ENDIF
78                IF (L.NE.0.OR.M.NE.0) CALL TDMA(CMJ,CCJ,CPJ,R,R,1,N2M)
79      1         DO 73 J=1,N2M
80    155     73  FDP_R(L,J,M)=R(J)
81                IF (L.EQ.0.AND.M.EQ.0) FDP_R(L,N2M,M)=0.
82
83            70  CONTINUE
```

< 최적화 코드 >

```
63           C    Solution of Poisson eq. ; REAL part
```

```
64
65                        DO 70 M=0,N3M/2
66
67                        DO 71 J=1,N2M
68                        DO 71 L=0,N1M-1
69        5               CMJ(L,J)=PMJ(J)
70       26               CCJ(L,J)=PCJ(J)-AK3(M)-AK1(L)
71                        CPJ(L,J)=PPJ(J)
72       22               R(L,J)=FRDP_R(L,J,M)
73                 71     CONTINUE
75                C       Special treatment to remove singularity when l=k=0
76
77                        IF (M.EQ.0) THEN
78                            CPJ(0,N2M-1)=0.0
79                            CALL TDMAI0(CMJ,CCJ,CPJ,R,R,1,N2M-1,0,0)
80                            CALL TDMAI0(CMJ,CCJ,CPJ,R,R,1,N2M,1,N1M-1)
81                        ELSE
82                            CALL TDMAI0(CMJ,CCJ,CPJ,R,R,1,N2M,0,N1M-1)
83                        ENDIF
84                        DO J=1,N2M
85                        DO L=0,N1M-1
86       16               FDP_R(L,J,M)=R(L,J)
87                c       IF (L.EQ.0.AND.M.EQ.0) FDP_R(L,N2M,M)=0.
88                        ENDDO
89                        ENDDO
90
91                 70     CONTINUE
92                        FDP_R(0,N2M,0)=0.
```

① Original 코드에서의 L, M, J 루프 실행을 M, J, L의 순서로 변경해 배열 FRDP_R(L,J,M)에 대한 접근이 연속적으로 처리되도록 하였다.

② TDMA의 최적화 루틴인 TDMAI0를 사용하고 있으며 이를 위해 앞서 살펴보았듯이 original 코드에서의 1차원 배열 CMJ, CCJ, CPJ, R는 2차원 배열로 구성돼 최적화된 루틴 TDMAI0로 전달되고 있다.

③ Original 코드에서의 IF (L.EQ.0.AND.M.EQ.0) FDP_R(L,N2M,M)=0. 부분을 FDP_R(0,N2M,0)=0.로 대체 하였다.

< Original 코드 >

```
86                C       Solution of Poisson eq. ; Image part
87
88                        DO 80 L=0,N1M-1
89                        DO 80 M=1,N3M/2-1
90        1               DO 81 J=1,N2M
91        4               CMJ(J)=PMJ(J)
92       13               CCJ(J)=PCJ(J)-AK3(M)-AK1(L)
93        2               CPJ(J)=PPJ(J)
94      221        81     R(J)=FRDP_I(L,J,M)
```

```
 95
 96                        CALL TDMA(CMJ,CCJ,CPJ,R,R,1,N2M)
 97
 98                        DO 83 J=1,N2M
 99        162      83     FDP_I(L,J,M)=R(J)
100          5      80     CONTINUE
```

< 최적화 코드 >

```
 95                 C      Solution of Poisson eq. ; Image part
 96
 97                        DO 80 M=1,N3M/2-1
 98                        DO 81 J=1,N2M
 99                        DO 81 L=0,N1M-1
100          1             CMJ(L,J)=PMJ(J)
101         14             CCJ(L,J)=PCJ(J)-AK3(M)-AK1(L)
102                        CPJ(L,J)=PPJ(J)
103         23      81     R(L,J)=FRDP_I(L,J,M)
104
105                        CALL TDMAI0(CMJ,CCJ,CPJ,R,R,1,N2M,0,N1M-1)
106
107                        DO 83 J=1,N2M
108                        DO 83 L=0,N1M-1
109         11      83     FDP_I(L,J,M)=R(L,J)
110                 80     CONTINUE
```

① Original 코드에서의 L, M, J 루프 실행을 M, J, L의 순서로 변경해 배열 FRDP_I(L,J,M)에 대한 접근의 연속성을 높이고 TDMA의 최적화 루틴인 TDMAI0를 사용하고 있다.

< Original 코드 >

```
103                 C      Inverse FFT of FDP in X3 direction
104
105                        DO 200 L=0,N1M-1
106                        DO 200 J=1,N2M
107
108          1             DO 21 M=0,N3M/2-1
109        353      21     R2(2*M)=FDP_R(L,J,M)
110                        DO 22 M=1,N3M/2-1
111        407      22     R2(2*M+1)=FDP_I(L,J,M)
112                        R2(1)=FDP_R(L,J,N3M/2)
113
114          1             CALL RDFT(N3M,-1,R2,IPF,WF)
115
116                        DO K=0,N3M-1
117          7             R2(K)=R2(K)*2/N3M
118                        END DO
```

```
 119
 120                          DO 23 K=1,N3M
 121        615       23      CDP(L,J,K)=R2(K-1)
 122
 123         16       200  CONTINUE
```

<최적화 코드>

```
 113                 C        Inverse FFT of FDP in X3 direction
 114
 115                          scale=1./float(n3m)
 116                          DO 200 J=1,N2M
 117                          DO 200 LS=0,N1M-1,IBLK
 118                                 LE=MIN(LS+IBLK-1,N1M-1)
 119                 c====
 120                              M=0
 121                          DO L=LS,LE
 122                              L1=L-LS
 123                              R2(2*M,L1)=FDP_R(L,J,M)
 124                              R2(2*M+1,L1)=0.
 125                          ENDDO
 126                          DO M=1,N3M/2-1
 127          1               DO L=LS,LE
 128                              L1=L-LS
 129          9                   R2(2*M,L1)=FDP_R(L,J,M)
 130         38                   R2(2*M+1,L1)=FDP_I(L,J,M)
 131                          ENDDO
 132                          ENDDO
 133                              M=N3M/2
 134                          DO L=LS,LE
 135                              L1=L-LS
 136                              R2(2*M,L1)=FDP_R(L,J,M)
 137                              R2(2*M+1,L1)=0.
 138                          ENDDO
 139
 140                          call crfft_d(r2,n3m+2,r2,n3m+2,n3m,le-ls+1,1,scale)
 141
 142          1               DO K=1,N3M
 143                          if(ls.ne.0) then
 144          1               DO L=LS,LE
 145                              L1=L-LS
 146         31               CDP(L,J,K)=R2(K-1,L1)
 147                          ENDDO
 148                          else
 149                          CDP(0,J,K)=R2(K-1,0)/2.
 150                          DO L=LS+1,LE
 151                              L1=L-LS
 152          1               CDP(L,J,K)=R2(K-1,L1)
 153          5               ENDDO
 154                          endif
 155          2               ENDDO
 156                 c====
```

```
157          1     200   CONTINUE
```

① L, J루프 반복을 J, L루프 반복으로 변경해 접근의 연속성을 높이고 FFT를 위해 루틴 RDFT를 사용하는 대신 최적화된 루틴 crfft_d를 사용하고 있다. original 코드에서 루틴 RDFT가 1차원 배열을 처리하는 반면, 최적화 루틴 crfft_d는 2차원 배열에 대한 계산을 수행하고 있다.

< Original 코드 >

```
126                C     Inverse Cosine transform of CDP in X1 direction
127
128                      DO 40 J=1,N2M
129                      DO 40 K=1,N3M
130
131                      DO 41 L=0,N1M-1
132          53    41    R1(L)=CDP(L,J,K)
133                      R1(0)=R1(0)/2
134
135                      CALL DDCT(N1M,1,R1,IPC,WC)
136
137                      DO JJ=0,N1M-1
138          13          R1(JJ)=R1(JJ)*2/N1M
139                      END DO
140
141                      DO 42 I=1,N1M
142          44    42    DP(I,J,K)=R1(I-1)
143
144                40    CONTINUE
```

< 최적화 코드 >

```
160                C     Inverse Cosine transform of CDP in X1 direction
161
162                      scale=2./N1M
163                      DO 40 K=1,N3M
164                      call cosfftk_d(CDP(0,1,K),M1+1,DP(1,1,K),M1,2*N1M,N2M,scale)
165                40    CONTINUE
```

① 배열 CDP에 대한 코사인 변환을 수행하는 DDCT루틴 대신 최적화된 cosfftk_d 루틴을 사용하고 있다. 최적화 코드에서는 임시 변수(original 코드의 R1)를 사용하지 않고 CDP를 직접 cosfftk_d로 전달해 그 결과를 DP로 직접 리턴받고 있으며, DDCT가 1차원 배열을 처리하는 반면 cosfftj_d는 2차원 배열에 대해 코사인 변환을 수행하고 있다.

O. upcalc.f

< Original 코드 >

```
21                C      U1 VELOCITY UPDATE
23                         DO 10 I=2,N1M
24                         DO 10 K=1,N3M
25                         DO 10 J=1,N2M
26      3121               U(1,I,J,K)=UH(1,I,J,K)
27                        >          -DT*(DP(I,J,K)-DP(I-1,J,K))*DX1
28                      10  CONTINUE
30                         DO 11 I=1,N1
31                         DO 11 K=1,N3M
32         4               U(1,I,0,K)=UBC1(1,I,K)
33         7               U(1,I,N2,K)=UBC2(1,I,K)
34                      11  CONTINUE
35                         DO 12 J=1,N2M
36                         DO 12 K=1,N3M
37         1               U(1,1,J,K)=UBC3(1,J,K)
38         5               U(1,N1,J,K)=UBC4(1,J,K)
40                      12  CONTINUE
41
42                c      U2 VELOCITY UPDATE
43
44                         DO 20 I=1,N1M
45         9               DO 20 K=1,N3M
46                         DO 20 J=2,N2M
47      3013               U(2,I,J,K)=UH(2,I,J,K)
48                        >          -DT*(DP(I,J,K)-DP(I,J-1,K))/H(J)
49                      20  CONTINUE
50
51                         DO 21 I=1,N1
52                         DO 21 K=1,N3M
53         2               U(2,I,1,K)=UBC1(2,I,K)
54         6               U(2,I,N2,K)=UBC2(2,I,K)
55                      21  CONTINUE
56                         DO 22 J=1,N2M
57                         DO 22 K=1,N3M
58         2               U(2,0,J,K)=UBC3(2,J,K)
59         3               U(2,N1,J,K)=UBC4(2,J,K)
60                      22  CONTINUE
61
62                C      U3 VELOCITY UPDATE
63
64                         DO 30 I=1,N1M
65                         DO 30 K=1,N3M
66         4               KM=KMA(K)
67                         DO 30 J=1,N2M
68      3098               U(3,I,J,K)=UH(3,I,J,K)
69                        >          -DT*(DP(I,J,K)-DP(I,J,KM))*DX3
70         2            30  CONTINUE
71
72                         DO 31 I=1,N1
73                         DO 31 K=1,N3M
```

```
74         5          U(3,I,0,K)=UBC1(3,I,K)
75         5          U(3,I,N2,K)=UBC2(3,I,K)
76                31  CONTINUE
77                    DO 32 J=1,N2M
78                    DO 32 K=1,N3M
79         3          U(3,0,J,K)=UBC3(3,J,K)
80         2          U(3,N1,J,K)=UBC4(3,J,K)
81                32  CONTINUE
83              C     PRESSURE UPDATE
85                    DO 40 I=1,N1M
86                    DO 40 J=1,N2M
87                    DO 40 K=1,N3M
88      1703          P(I,J,K)=P(I,J,K)+DP(I,J,K)
89                40  CONTINUE
```

< 최적화 코드 >

```
21              C     U1 VELOCITY UPDATE
22
23                    DO K=1,N3M
24              c=====
25                    DO I=1,N1
26         3          U(I,0,K,1)=UBC1(I,K,1)
27                    ENDDO
28                    DO J=1,N2M
29         1             U(1,J,K,1)=UBC3(J,K,1)
30                    DO I=2,N1M
31        85             U(I,J,K,1)=UH(I,J,K,1)
32                   >         -DT*(DP(I,J,K)-DP(I-1,J,K))*DX1
33                    ENDDO
34         1             U(N1,J,K,1)=UBC4(J,K,1)
35                    ENDDO
36                    DO I=1,N1
37         4             U(I,N2,K,1)=UBC2(I,K,1)
38                    ENDDO
39

50              c     U2 VELOCITY UPDATE
51
52              c     DO 22 K=1,N3M
53                    DO I=1,N1
54         1             U(I,1,K,2)=UBC1(I,K,2)
55                    ENDDO
56                       J=1
57                       U(0,J,K,2)=UBC3(J,K,2)
58                       U(N1,J,K,2)=UBC4(J,K,2)
59                    DO J=2,N2M
60         1             U(0,J,K,2)=UBC3(J,K,2)
61                    DO I=1,N1M
62        70             U(I,J,K,2)=UH(I,J,K,2)
63                   >         -DT*(DP(I,J,K)-DP(I,J-1,K))/H(J)
64                    ENDDO
```

```
65        2              U(N1,J,K,2)=UBC4(J,K,2)
66                     ENDDO
67                     DO I=1,N1
68        2              U(I,N2,K,2)=UBC2(I,K,2)
69                     ENDDO

80                C      U3 VELOCITY UPDATE
83                     KM=KMA(K)
84                     DO I=1,N1M
85        3              U(I,0,K,3)=UBC1(I,K,3)
86                     ENDDO
87                     DO J=1,N2M
88        1              U(0,J,K,3)=UBC3(J,K,3)
89                     DO I=1,N1M
90        80             U(I,J,K,3)=UH(I,J,K,3)
91                    >           -DT*(DP(I,J,K)-DP(I,J,KM))*DX3
92                     ENDDO
93        3              U(N1,J,K,3)=UBC4(J,K,3)
94                     ENDDO
95                     DO I=1,N1M
96        1              U(I,N2,K,3)=UBC2(I,K,3)
97                     ENDDO
98
108               C      PRESSURE UPDATE
109
110               c      DO 40 K=1,N3M
111                      DO 40 J=1,N2M
112                      DO 40 I=1,N1M
113       65             P(I,J,K)=P(I,J,K)+DP(I,J,K)
114                  40  CONTINUE
115               c=====
116                      enddo
```

① Original 코드에서 U1, U2, U3 velocity update 부분은 모두 I, K, J루프 반복과 I, K 루프 J, K루프 반복으로 각각 구성돼 있다. 이것을 K, J, I의 순으로 변경하여 배열 U, UH, DP, P에 대한 접근이 연속적이 되도록 하였다.

② 루프 실행 순서를 변경하면, 세 부분은 모두 K루프에 대해 동일한 구조를 가지므로 K루프를 merge하였다.

7. 순차코드 최적화 결과

	1cp P690+ 1.7GHZ
Original Code	16.24sec/time step
Tuned Code	2.09sec/time step
Speedup	7.77

최적화된 코드는 time step 기준으로 original 코드 대비 7.77배(문제크기: 257 x 65 x 129)의 성능향상을 보여 주었다. 코드 전체적으로는 주로 다차원 배열에 대한 메모리 접근이 연속적이 되도록 루프 실행순서를 변경하고, 성능이 우수한 Fourier 변환 루틴을 이용함으로써 성능향상을 얻고 있다. 루프 내에서 불필요하게 반복 계산되는 부분은 루프 밖으로 이동시키고, 서브루틴에서 발생하는 캐시실패를 최소화하기 위해 루틴에서 처리되는 배열의 구조를 변경시키는 방법도 사용하고 있다. 기타 코드 전체에서 불필요하게 반복되는 계산을 줄이고 루프 merge를 통한 최적화 등을 통해 성능향상을 얻고 있다.

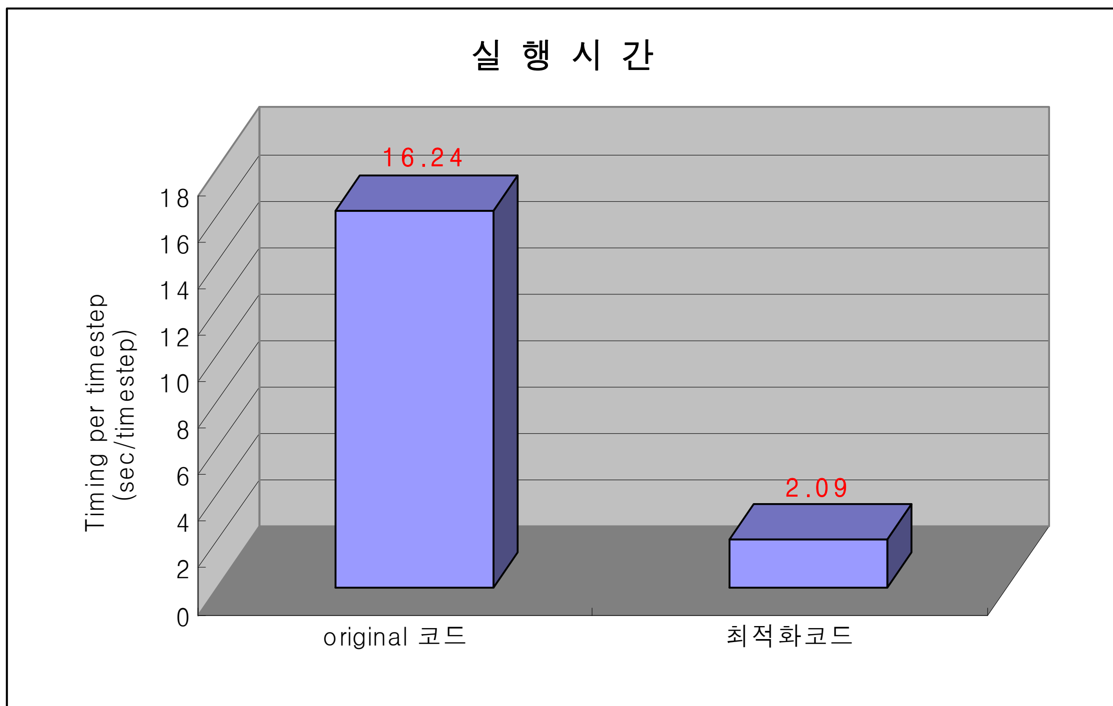


그림.II.2 코드의 성능 비교

8. OpenMP 병렬화 코드 비교 및 분석

다음의 소스 코드들에 OpenMP directive를 삽입하여 병렬화 하였다. 이들 중 대표적인 다음 몇 가지에 대해 병렬화 과정을 살펴 본다.

```
iniup.f
divcheck.f
cfl.f
getup.f
bcont.f
Uhcalc.f
rhs1.f
rhs2.f
rhs3.f
getduh1.f
getduh2.f
getduh3.f
rshsp.f
takedp.f
upcalc.f
plane_up.f
```

8.1. iniup.f

< 최적화 순차코드 >

```
20                   DO 10 K=1,N3M
21                   DO 10 J=0,N2
22                   DO 10 I=0,N1
23      4            U(I,J,K,1)=0.0
24      4            U(I,J,K,2)=0.0
25                   U(I,J,K,3)=0.0
26           10      CONTINUE
29           C       IMPOSE U VELOCITY
30                   DO 20 K=1,N3M
31                   DO 20 J=0,N2
32                   DO 20 I=0,N1
33      1    20      U(I,J,K,1)=1.0
36           C       IMPOSE ZERO-PRESSURE FLUCTUATIONS
37                   DO 60 K=1,N3M
38                   DO 60 J=1,N2M
39                   DO 60 I=1,N1M
40      2            P(I,J,K)=0.0
41                60 CONTINUE
```

<OpenMP code>

```
20                 !$omp parallel do
21                       DO 60 K=1,N3M
22                       DO 10 J=0,N2
23                       DO 10 I=0,N1
24        78             U(I,J,K,1)=0.0
25        79             U(I,J,K,2)=0.0
26        78             U(I,J,K,3)=0.0
27                 10    CONTINUE
30                 C     IMPOSE U VELOCITY
31                 c!$omp parallel do
32                 c     DO 20 K=1,N3M
33                       DO 20 J=0,N2
34                       DO 20 I=0,N1
35                 20    U(I,J,K,1)=1.0
38                 C     IMPOSE ZERO-PRESSURE FLUCTUATIONS
39                 c!$omp parallel do
40                 c     DO 60 K=1,N3M
41                       DO 60 J=1,N2M
42                       DO 60 I=1,N1M
43        60             P(I,J,K)=0.0
44                    60 CONTINUE
```

① 배열을 초기화하는 부분을 parallel do 지시어를 이용해 병렬화 하였다.

8.2. divcheck.f

< 최적화 순차코드 >

```
18                       DO 20 K=1,N3M
19         1             DO 20 J=1,N2M
20                       DO 20 I=1,N1M
21                       KP=KPA(K)
22                       KM=KMA(K)
23       101             DIV=ABS((U(I+1,J,K,1)-U(I,J,K,1))*DX1
24                      >        +(U(I,J+1,K,2)-U(I,J,K,2))/DY(J)
25                      >        +(U(I,J,KP,3)-U(I,J,K,3))*DX3)
26                       IF (DIV.GT.DIVMAX) THEN
27                           IMAX = I
28                           JMAX = J
29                           KMAX = K
30                       ENDIF
31        45             DIVMAX = AMAX1(DIV,DIVMAX)
32                    20 CONTINUE
```

<OpenMP code>

```
18                 !$omp parallel do private(KP,KM,DIV) reduction(max:DIVMAX)
19                       DO 20 K=1,N3M
20                       KP=KPA(K)
21                       KM=KMA(K)
22                       DO 20 J=1,N2M
```

```
23                          DO 20 I=1,N1M
24                          DIV=ABS((U(I+1,J,K,1)-U(I,J,K,1))*DX1
25                         >        +(U(I,J+1,K,2)-U(I,J,K,2))/DY(J)
26                         >        +(U(I,J,KP,3)-U(I,J,K,3))*DX3)
32        217               DIVMAX = AMAX1(DIV,DIVMAX)
33                      20  CONTINUE
```

① 최대값 DIVMAX를 구하는 부분을 병렬화하면서 여러 스레드에서 공유되지 않아야 되는 KP, KM, DIV는 private 변수로 처리하고 있다.

② 판단구문 if를 제거하고 보조 지시어 reduction과 연산 max를 이용해 원하는 최대값을 병렬로 얻고 있다.

8.3. cfl.f

< 최적화 순차코드 >

```
19                          DO 10 K=1,N3M
20                          KP=KPA(K)
21                          DO 10 J=1,N2M
22          1               JP=J+1
23                          DO 10 I=1,N1M
24                          IP=I+1
25        183               CFLL=ABS(U(I,J,K,1)+U(IP,J,K,1))*0.5*DX1
26                         >      +ABS(U(I,J,K,2)+U(I,JP,K,2))*0.5/DY(J)
27                         >      +ABS(U(I,J,K,3)+U(I,J,KP,3))*0.5*DX3
28          2               CFLM=AMAX1(CFLM,CFLL)
29          2         10    CONTINUE
```

<OpenMP code>

```
19                    !$omp parallel do private(KP,JP,IP,CFLL) reduction(max:CFLM)
20                          DO 10 K=1,N3M
21                          KP=KPA(K)
22                          DO 10 J=1,N2M
23                          JP=J+1
24          1               DO 10 I=1,N1M
25                          IP=I+1
26                          CFLL=ABS(U(I,J,K,1)+U(IP,J,K,1))*0.5*DX1
27                         >      +ABS(U(I,J,K,2)+U(I,JP,K,2))*0.5/DY(J)
28                         >      +ABS(U(I,J,K,3)+U(I,J,KP,3))*0.5*DX3
29        329               CFLM=AMAX1(CFLM,CFLL)
30          1         10    CONTINUE
```

① 최대값 CFLM 구하는 부분을 병렬화하면서 여러 스레드에서 공유되지 않아야 되는 KP, JP, IP, CFLL을 private 변수로 처리하고 있다.

② 보조 지시어 reduction과 연산 max를 이용해 최대값 CFLM을 병렬로 얻고 있다.

8.4. getup.f

< 최적화 순차코드 >

```
14                  C       INITIALIZE THE INTERMEDIATE VELOCITY AND PRESSURE
15                          DO 10 K=1,N3M
16                          DO 10 J=0,N2
17                          DO 10 I=0,N1
18        12                UH(I,J,K,1)=0.0
19       169                UH(I,J,K,2)=0.0
20         2                UH(I,J,K,3)=0.0
21                   10     CONTINUE
22                          DO 20 K=1,N3M
23         1                DO 20 J=1,N2M
24                          DO 20 I=1,N1M
25        54         20     DP(I,J,K)=0.0
```

<OpenMP code>

```
14                  C       INITIALIZE THE INTERMEDIATE VELOCITY AND PRESSURE
15                  !$omp parallel do
16                          DO 20 K=1,N3M
17                          DO 10 J=0,N2
18       224                DO 10 I=0,N1
19       117                UH(I,J,K,1)=0.0
20       101                UH(I,J,K,2)=0.0
21       107                UH(I,J,K,3)=0.0
22                   10     CONTINUE
23                  c       DO 20 K=1,N3M
24                          DO 20 J=1,N2M
25                          DO 20 I=1,N1M
26       212         20     DP(I,J,K)=0.0
```

① K 루프를 merge하고, parallel do를 삽입해 병렬화 하였다.

8.5. bcont.f

< 최적화 순차코드 >

```
21                          DO 10 K=1,N3M
22                          DO 10 I=1,N1
24                  c       Lower Wall boundary conditions
25         3                UBC1(I,K,1)=0.0   ! U(1,I,0,K)   Lower WALL : No-slip condition
26                          UBC1(I,K,2)=0.0   ! U(2,I,1,K)
27                          UBC1(I,K,3)=0.0   ! U(3,I,0,K)
36                  C       For MAIN SIMULATION
37                          UBC2(I,K,1)=1.0   ! U(1,I,N2,K)
38                          UBC2(I,K,2)=U(I,N2M,K,2)   ! U(2,I,N2,K)
39         1                UBC2(I,K,3)=U(I,N2M,K,3)   ! U(3,I,N2,K)
41                   10     CONTINUE
```

```
50              C      BOUNDARY CONDITION 2001/12/29
52                     DO 15 K=1,N3M
53                     DO 15 I=1,N1M/3
54        1     15     UBC1(I,K,1)=U(I,0,K,1)
56              C      INLET BOUNDARY CONDITION
57                     DO 20 K=1,N3M
58                     DO 20 J=1,N2M
59                        UBC3(J,K,1)=1.0
60        2               UBC3(J,K,2)=0.0
61                        UBC3(J,K,3)=0.0
62              20     CONTINUE
```

<OpenMP code>

```
22              !$omp parallel do
23                     DO 20 K=1,N3M
24                     DO 10 I=1,N1
26              c      Lower Wall boundary conditions
27        4            UBC1(I,K,1)=0.0   ! U(1,I,0,K)   Lower WALL : No-slip condition
28        3            UBC1(I,K,2)=0.0   ! U(2,I,1,K)
29        5            UBC1(I,K,3)=0.0   ! U(3,I,0,K)
30
31              c      Upper Wall boundary conditions
32
38              C      For MAIN SIMULATION
39        2            UBC2(I,K,1)=1.0   ! U(1,I,N2,K)
40        1            UBC2(I,K,2)=U(I,N2M,K,2)   ! U(2,I,N2,K)
41        4            UBC2(I,K,3)=U(I,N2M,K,3)   ! U(3,I,N2,K)
43               10    CONTINUE
52              C      BOUNDARY CONDITION 2001/12/29
55                     DO 15 I=1,N1M/3
56              15     UBC1(I,K,1)=U(I,0,K,1)
58              C      INLET BOUNDARY CONDITION
60                     DO 20 J=1,N2M
61                        UBC3(J,K,1)=1.0
62                        UBC3(J,K,2)=0.0
63                        UBC3(J,K,3)=0.0
64              20     CONTINUE
```

① K 루프를 merge하고, parallel do를 삽입해 병렬화 하였다.

< 최적화 순차코드 >

```
66                     UC=0.0
67                     DO 31 K=1,N3M
68                     DO 31 J=1,N2M
69        5            UC=UC+U(N1,J,K,1)*DY(J)/DX3
70              31     CONTINUE
71                     UC=UC/ALZ/ALY                ! BULK VELOCITY
72                     DO 32 K=1,N3M
73                     DO 32 J=1,N2M
```

```
 74     4          UBC4(J,K,1)=U(N1,J,K,1)-DT*DX1*UC*(U(N1,J,K,1)-U(N1M,J,K,1))
 75     1          UBC4(J,K,2)=U(N1,J,K,2)-DT*DX1*UC*(U(N1,J,K,2)-U(N1M,J,K,2))
 76     6     32   UBC4(J,K,3)=U(N1,J,K,3)-DT*DX1*UC*(U(N1,J,K,3)-U(N1M,J,K,3))
 78           C    THE INTERGAL OF MASS FLUX MUST BE ZERO!!!!
 79                Q_IN=0.
 80                Q_UP=0.
 81                Q_DOWN=0.
 82                Q_EX=0.
 83                DO 33 K=1,N3M
 84                DO 33 J=1,N2M
 85     1          Q_IN=Q_IN+UBC3(J,K,1)*DY(J)/DX3
 86           33   Q_EX=Q_EX+UBC4(J,K,1)*DY(J)/DX3
 88                DO 35 K=1,N3M
 89                DO 35 I=1,N1M
 90                Q_UP=Q_UP+UBC2(I,K,2)/DX1/DX3
 91           35   Q_DOWN=Q_DOWN+UBC1(I,K,2)/DX1/DX3
 93                WRITE(*,*)
 94                WRITE(*,100) Q_IN,Q_UP,Q_EX
 96           99   FORMAT(4(E15.7,X))
 97          100 FORMAT('MASS FLUX : Q_IN=',E10.4,X,'Q_UP=',E10.4,X,'Q_EX=',E10.4)
 99                RATE=(Q_IN+Q_DOWN-Q_UP)/Q_EX
101                DO 34 K=1,N3M
102                DO 34 J=1,N2M
103                UBC4(J,K,1)=RATE*UBC4(J,K,1)
104                UBC4(J,K,2)=RATE*UBC4(J,K,2)
105     1     34   UBC4(J,K,3)=RATE*UBC4(J,K,3)
```

<OpenMP code>

```
 68                UC=0.0
 69           !$omp parallel do reduction(+:UC)
 70                DO 31 K=1,N3M
 71                DO 31 J=1,N2M
 72     5          UC=UC+U(N1,J,K,1)*DY(J)/DX3
 73           31   CONTINUE
 74                UC=UC/ALZ/ALY                    ! BULK VELOCITY
 75           !$omp parallel do
 76                DO 32 K=1,N3M
 77                DO 32 J=1,N2M
 78     5          UBC4(J,K,1)=U(N1,J,K,1)-DT*DX1*UC*(U(N1,J,K,1)-U(N1M,J,K,1))
 79     4          UBC4(J,K,2)=U(N1,J,K,2)-DT*DX1*UC*(U(N1,J,K,2)-U(N1M,J,K,2))
 80     4     32   UBC4(J,K,3)=U(N1,J,K,3)-DT*DX1*UC*(U(N1,J,K,3)-U(N1M,J,K,3))
 82           C    THE INTERGAL OF MASS FLUX MUST BE ZERO!!!!
 83                Q_IN=0.
 84                Q_UP=0.
 85                Q_DOWN=0.
 86                Q_EX=0.
 87           !$omp parallel do reduction(+:Q_IN,Q_UP,Q_DOWN,Q_EX)
 88                DO 35 K=1,N3M
 89                DO 33 J=1,N2M
 90                Q_IN=Q_IN+UBC3(J,K,1)*DY(J)/DX3
 91           33   Q_EX=Q_EX+UBC4(J,K,1)*DY(J)/DX3
 94                DO 35 I=1,N1M
 95                Q_UP=Q_UP+UBC2(I,K,2)/DX1/DX3
```

```
 96                35      Q_DOWN=Q_DOWN+UBC1(I,K,2)/DX1/DX3
 98                        WRITE(*,*)
 99                        WRITE(*,100) Q_IN,Q_UP,Q_EX
101                99      FORMAT(4(E15.7,X))
102             100     FORMAT('MASS FLUX : Q_IN=',E10.4,X,'Q_UP=',E10.4,X,'Q_EX=',E10.4)
104                        RATE=(Q_IN+Q_DOWN-Q_UP)/Q_EX
106                !$omp parallel do
107                        DO 34 K=1,N3M
108                        DO 34 J=1,N2M
109                        UBC4(J,K,1)=RATE*UBC4(J,K,1)
110                        UBC4(J,K,2)=RATE*UBC4(J,K,2)
111                34      UBC4(J,K,3)=RATE*UBC4(J,K,3)
```

① parallel do를 이용해 루프를 병렬화 하였고, UC,Q_IN,Q_UP,Q_DOWN,Q_EX 등 루프의 반복 동안 누적 계산되는 변수들은 스레드 별로 병렬 계산된 후 다시 합쳐져야 하므로 reduction과 연산 '+'를 사용하고 있다.

8.6. uhcalc.f

< 최적화 순차코드 >

```
 76               C       INTERMEDIATE VELOCITY, U*=U^N+dU*
 77
 78                        DO 300 K=1,N3M
 79                        DO 300 J=1,N2M
 80                        DO 300 I=2,N1M
 81        60      300     UH(I,J,K,1)=U(I,J,K,1)+UH(I,J,K,1)
 82
 83                        DO 310 K=1,N3M
 84                        DO 310 J=2,N2M
 85                        DO 310 I=1,N1M
 86        65      310     UH(I,J,K,2)=U(I,J,K,2)+UH(I,J,K,2)
 87
 88                        DO 320 K=1,N3M
 89         1              DO 320 J=1,N2M
 90                        DO 320 I=1,N1M
 91        75      320     UH(I,J,K,3)=U(I,J,K,3)+UH(I,J,K,3)
```

<OpenMP code>

```
 77               C       INTERMEDIATE VELOCITY, U*=U^N+dU*
 78
 79                !$omp parallel do
 80                        DO 320 K=1,N3M
 81                        DO 300 J=1,N2M
 82                        DO 300 I=2,N1M
 83       133      300     UH(I,J,K,1)=U(I,J,K,1)+UH(I,J,K,1)
 84
 85                c!$omp parallel do
 86                c       DO 310 K=1,N3M
```

```
87                      DO 310 J=2,N2M
88                      DO 310 I=1,N1M
89      133     310     UH(I,J,K,2)=U(I,J,K,2)+UH(I,J,K,2)
90
91              c!$omp parallel do
92              c       DO 320 K=1,N3M
93                      DO 320 J=1,N2M
94                      DO 320 I=1,N1M
95      146     320     UH(I,J,K,3)=U(I,J,K,3)+UH(I,J,K,3)
```

① parallel do를 이용해 루프를 병렬화 하였다.

8.7. rhs1.f

< 최적화 순차코드 >

```
25              DO 10 K=1,N3M
26              KP=KPA(K)
27              KM=KMA(K)
29              DO I=2,N1M
30              IP=I+1
31              IM=I-1
32      2       VISCOS1(I) =0.5/RE*(2.0/H(2)/(H(1)+H(2))*U(I,2,K,1)
33             >                        -2.0/H(1)/H(2)*U(I,1,K,1)
34             >                        +2.0/H(1)/(H(1)+H(2))*U(I,0,K,1))
35      3       VISCOS2(I) =0.5/RE*(2.0/H(N2M)/(H(N2)+H(N2M))*U(I,N2M-1,K,1)
36             >                        -2.0/H(N2M)/H(N2)*U(I,N2M,K,1)
37             >                        +2.0/H(N2)/(H(N2)+H(N2M))*U(I,N2,K,1))
38      2       BC_DOWN(I)=0.5/RE*2.0/H(1)/(H(1)+H(2))*UBC1(I,K,1)
39             >        +0.5/DY(1)*0.5*(U(I,1,K,2)+U(IM,1,K,2))*UBC1(I,K,1)
40             >        +0.5/DY(1)*U(I,0,K,1)*0.5*(UBC1(I,K,2)+UBC1(IM,K,2))
42              BC_UP(I)   =0.5/RE*2.0/H(N2)/(H(N2)+H(N2M))*UBC2(I,K,1)
43             >        -0.5/DY(N2M)*0.5*(U(I,N2,K,2)+U(IM,N2,K,2))*UBC2(I,K,1)
44             >        -0.5/DY(N2M)*U(I,N2,K,1)*0.5*(UBC2(I,K,2)+UBC2(IM,K,2))
45              ENDDO
47              DO 10 J=1,N2M
48              JP=J+1
49              JM=J-1
52              FJUM=FJMU(J)
53              FJUP=FJPA(J)
54      4       DO I=2,N1M
55              IP=I+1
56              IM=I-1
59              FIUM=FIMU(I)
60      18      FIUP=FIPA(I)
62      100     VISCOS=0.5*DX1Q/RE*(U(IP,J,K ,1)-2.0*U(I,J,K,1)+U(IM,J,K ,1))
63             >       +0.5*DX3Q/RE*(U(I ,J,KP,1)-2.0*U(I,J,K,1)+U(I ,J,KM,1))
64             >       +FJUM*FJUP*0.5/RE*(HP(J)*U(I,JP,K,1)
65             >                        -HC(J)*U(I,J ,K,1)
66             >                        +HM(J)*U(I,JM,K,1))
67             >       +(1.-FJUM)*VISCOS1(I)
68             >       +(1.-FJUP)*VISCOS2(I)
```

```
 70         14         PRESSG1=DX1*(P(I,J,K)-P(IM,J,K))
 73         14         U2=0.5*(U(IP,J,K,1)+U(I ,J,K,1))
 74          4         U1=0.5*(U(I ,J,K,1)+U(IM,J,K,1))
 76         32         BC_IN = DX1*U1*0.5*UBC3(J,K,1)
 77                   >        +0.5/RE*DX1Q*UBC3(J,K,1)
 78         30         BC_OUT=-DX1*U2*0.5*UBC4(J,K,1)
 79                   >        +0.5/RE*DX1Q*UBC4(J,K,1)
 82         45         BC=(1.-FJUM)*BC_DOWN(I)
 83                   >   +(1.-FJUP)*BC_UP(I)
 84                   >   +(1.-FIUM)*BC_IN
 85                   >   +(1.-FIUP)*BC_OUT
 87         65         RDUH1(I,J,K)=1./DT*U(I,J,K,1)
 88                   >            -PRESSG1+VISCOS
 89                   >            +BC
 90                    ENDDO
 91          4         DO 10 I=2,N1M
 92                    IP=I+1
 93                    IM=I-1
 96                    FIUM=FIMU(I)
 97                    FIUP=FIPA(I)
....................................................
....................................................
163                10  CONTINUE
```

<OpenMP code>

```
  24                   !$omp   parallel do default(shared)
  25                   !$omp& private(IP,IM,JP,JM,KP,KM,FIUM,FIUP,FJUM,FJUP,
  26                   !$omp& VISCOS,PRESSG1,BC_DOWN,BC_UP,BC_IN,BC_OUT,BC,
  27                   !$omp& U1,U2,V1,V2,W1,W2,
  28                   !$omp& API,ACI,AMI,APJ,ACJ,AMJ,APK,ACK,AMK,
  29                   !$omp& RM11U_N,RM12V_N,RM13W_N)
  30                       DO 10 K=1,N3M
  31                       KP=KPA(K)
  32                       KM=KMA(K)
  52          8            DO 10 J=1,N2M
  53                       JP=J+1
  54                       JM=J-1
  57                       FJUM=FJMU(J)
  58                       FJUP=FJPA(J)
  59         13            DO 10 I=2,N1M
  60                       IP=I+1
  61                       IM=I-1
  64                       FIUM=FIMU(I)
  65                       FIUP=FIPA(I)
  67                       VISCOS=0.5*DX1Q/RE*(U(IP,J,K ,1)-2.0*U(I,J,K,1)+U(IM,J,K ,1))
  68                      >        +0.5*DX3Q/RE*(U(I ,J,KP,1)-2.0*U(I,J,K,1)+U(I ,J,KM,1))
  69                      >        +FJUM*FJUP*0.5/RE*(HP(J)*U(I,JP,K,1)
  70                      >                          -HC(J)*U(I,J ,K,1)
  71                      >                          +HM(J)*U(I,JM,K,1))
  74                      >        +(1.-FJUM)*0.5/RE*(2.0/H(2)/(H(1)+H(2))*U(I,2,K,1)
  75                      >                          -2.0/H(1)/H(2)*U(I,1,K,1)
  76                      >                          +2.0/H(1)/(H(1)+H(2))*U(I,0,K,1))
  77                   >        +(1.-FJUP)*0.5/RE*(2.0/H(N2M)/(H(N2)+H(N2M))*U(I,N2M-1,K,1)
```

```
  78                        >                         -2.0/H(N2M)/H(N2)*U(I,N2M,K,1)
  79                        >                         +2.0/H(N2)/(H(N2)+H(N2M))*U(I,N2,K,1))
  81                         PRESSG1=DX1*(P(I,J,K)-P(IM,J,K))
  82                         BC_DOWN=0.5/RE*2.0/H(1)/(H(1)+H(2))*UBC1(I,K,1)
  83                        >        +0.5/DY(1)*0.5*(U(I,1,K,2)+U(IM,1,K,2))*UBC1(I,K,1)
  84                        >        +0.5/DY(1)*U(I,0,K,1)*0.5*(UBC1(I,K,2)+UBC1(IM,K,2))
  85                         BC_UP   =0.5/RE*2.0/H(N2)/(H(N2)+H(N2M))*UBC2(I,K,1)
  86                        >        -0.5/DY(N2M)*0.5*(U(I,N2,K,2)+U(IM,N2,K,2))*UBC2(I,K,1)
  87                       >        -0.5/DY(N2M)*U(I,N2,K,1)*0.5*(UBC2(I,K,2)+UBC2(IM,K,2))
  91                         U2=0.5*(U(IP,J,K,1)+U(I ,J,K,1))
  92                         U1=0.5*(U(I ,J,K,1)+U(IM,J,K,1))
  94                         BC_IN = DX1*U1*0.5*UBC3(J,K,1)
  95                        >        +0.5/RE*DX1Q*UBC3(J,K,1)
  96                         BC_OUT=-DX1*U2*0.5*UBC4(J,K,1)
  97                        >        +0.5/RE*DX1Q*UBC4(J,K,1)
 100                         BC=(1.-FJUM)*BC_DOWN
 101                        >   +(1.-FJUP)*BC_UP
 102                        >   +(1.-FIUM)*BC_IN
 103                        >   +(1.-FIUP)*BC_OUT
  105          1695           RDUH1(I,J,K)=1./DT*U(I,J,K,1)
 106                        >             -PRESSG1+VISCOS
 107                        >             +BC
.....................................................
.....................................................
 172            2        10    CONTINUE
```

① parallel do를 이용해 제일 바깥쪽 K루프를 병렬화 하였다.

② 순차코드 최적화 과정에서 J루프에 독립적인 부분을 J루프 밖으로 이동시켜 불필요한 계산의 반복을 줄이는 방법을 이용하였다. 병렬 코드에서는 이 부분을 그대로 병렬화 시키지 않고 다시 원래의 모양으로 되돌린 후 병렬화를 시도하고 있다. 불필요하게 반복되는 부분을 J루프 밖에서 계산하는 최적화된 코드를 병렬화해 비교해 본 결과 스레드 개수가 4개 이상이 되면서 위와 같이 원래의 코드를 병렬화 했을 때 더 성능이 좋게 나옴을 확인할 수 있었다.

8.8. getduh1.f

< 최적화 순차코드 >

```
  30                         DO 2 K=1,N3M
  31                         DO 20 J=1,N2M
  32            1            JP=J+1
  33                         JM=J-1
  36                         FJUM=FJMU(J)
  37            2            FJUP=FJPA(J)
  38                         DO 20 I=2,N1M
  39                         IP=I+1
  40                         IM=I-1
  42            7            V2=0.5*(U(I,JP,K,2)+U(IM,JP,K,2))
  43            1            V1=0.5*(U(I,J ,K,2)+U(IM,J ,K,2))
```

```
45    36        APJ(I,J)=FJUP*(
46             >            FJUM*(-0.5)*HP(J)/RE
47             >          +(1.-FJUM)*(-0.5)/RE*2.0/H(2)/(H(1)+H(2))
48             >       +0.5/DY(J)*V2/H(JP)*DY(J)/2.0
49             >       )*DT
50    52        ACJ(I,J)= (FJUM*FJUP*0.5*HC(J)/RE
51             >         +(1.-FJUM)*0.5/RE*2.0/H(1)/H(2)
52             >         +(1.-FJUP)*0.5/RE*2.0/H(N2)/H(N2M)
53             >         +0.5/DY(J)*(FJUP*V2/H(JP)*DY(JP)/2.0
54             >                    -FJUM*V1/H(J )*DY(JM)/2.0)
55             >         )*DT
56             >        +1.0
57    12        AMJ(I,J)=FJUM*(
58             >            FJUP*(-0.5)*HM(J)/RE
59             >         +(1.-FJUP)*(-0.5)/RE*2.0/H(N2M)/(H(N2M)+H(N2))
60             >       -0.5/DY(J)*V1/H(J)*DY(J)/2.0
61             >       )*DT
62    60        R2(I,J)=RDUH1(I,J,K)*DT
63          20  CONTINUE
65              CALL TDMAI(AMJ,ACJ,APJ,R2,R2,1,N2M,2,N1M)
66              DO 21 J=1,N2M
67              DO 21 I=2,N1M
68    51    21  UH(I,J,K,1)=R2(I,J)
69           2  CONTINUE
```

<OpenMP code>

```
30          !$omp parallel do private(JP,JM,IP,IM,FJUM,FJUP,V1,V2,
31          !$omp& APJ,ACJ,AMJ,R2)
32              DO 2 K=1,N3M
33              DO 20 J=1,N2M
34              JP=J+1
35              JM=J-1
38              FJUM=FJMU(J)
39              FJUP=FJPA(J)
40    12        DO 20 I=2,N1M
41              IP=I+1
42              IM=I-1
44    30        V2=0.5*(U(I,JP,K,2)+U(IM,JP,K,2))
45              V1=0.5*(U(I,J ,K,2)+U(IM,J ,K,2))
47    79        APJ(I,J)=FJUP*(
48             >            FJUM*(-0.5)*HP(J)/RE
49             >          +(1.-FJUM)*(-0.5)/RE*2.0/H(2)/(H(1)+H(2))
50             >       +0.5/DY(J)*V2/H(JP)*DY(J)/2.0
51             >       )*DT
52   141        ACJ(I,J)= (FJUM*FJUP*0.5*HC(J)/RE
53             >         +(1.-FJUM)*0.5/RE*2.0/H(1)/H(2)
54             >         +(1.-FJUP)*0.5/RE*2.0/H(N2)/H(N2M)
55             >         +0.5/DY(J)*(FJUP*V2/H(JP)*DY(JP)/2.0
56             >                    -FJUM*V1/H(J )*DY(JM)/2.0)
57             >         )*DT
58             >        +1.0
59   144        AMJ(I,J)=FJUM*(
60             >            FJUP*(-0.5)*HM(J)/RE
```

```
 61                        >            +(1.-FJUP)*(-0.5)/RE*2.0/H(N2M)/(H(N2M)+H(N2))
 62                        >          -0.5/DY(J)*V1/H(J)*DY(J)/2.0
 63                        >          )*DT
 64        157              R2(I,J)=RDUH1(I,J,K)*DT
 65          1        20    CONTINUE
 67                         CALL TDMAI(AMJ,ACJ,APJ,R2,R2,1,N2M,2,N1M)
 68                         DO 21 J=1,N2M
 69                         DO 21 I=2,N1M
 70        167        21    UH(I,J,K,1)=R2(I,J)
 71                    2    CONTINUE
```

< 최적화 순차코드 >

```
 71                         DO 1 K=1,N3M
 72                         DO 10 J=1,N2M
 73                         DO 10 I=2,N1M
 74                         IP=I+1
 75                         IM=I-1
 78          1              FIUM=FIMU(I)
 79          1              FIUP=FIPA(I)
 81         31              U2=0.5*(U(IP,J,K,1)+U(I ,J,K,1))
 82         10              U1=0.5*(U(I ,J,K,1)+U(IM,J,K,1))
 83         11              API(I,J)= (-0.5*DX1Q/RE
 84                        >            +DX1*U2*0.5)*FIUP
 85                        >          *DT
 86         26              ACI(I,J)= 1.0+(
 87                        >          +DX1Q/RE
 88                        >          +DX1*(U2*0.5-U1*0.5)
 89                        >          )*DT
 91         21              AMI(I,J)= (-0.5*DX1Q/RE
 92                        >            -DX1*U1*0.5)*FIUM
 93                        >          *DT
 95         18              R1(I,J)=UH(I,J,K,1)
 96                   10    CONTINUE
 98                         CALL TDMAJ(AMI,ACI,API,R1,R1,2,N1M,1,N2M)
 99                         DO 11 J=1,N2M
100                         DO 11 I=2,N1M
101         35        11    UH(I,J,K,1)=R1(I,J)
102                    1    CONTINUE
```

<OpenMP code>

```
 73                  !$omp parallel do private(IP,IM,FIUM,FIUP,U1,U2,
 74                  !$omp& API,ACI,AMI,R1)
 75                         DO 1 K=1,N3M
 76                         DO 10 J=1,N2M
 77         28              DO 10 I=2,N1M
 78                         IP=I+1
 79                         IM=I-1
 82                         FIUM=FIMU(I)
 83                         FIUP=FIPA(I)
 85                         U2=0.5*(U(IP,J,K,1)+U(I ,J,K,1))
```

```
 86                           U1=0.5*(U(I ,J,K,1)+U(IM,J,K,1))
 87          112              API(I,J)= (-0.5*DX1Q/RE
 88                          >          +DX1*U2*0.5)*FIUP
 89                          >        *DT
 90           72              ACI(I,J)= 1.0+(
 91                          >       +DX1Q/RE
 92                          >       +DX1*(U2*0.5-U1*0.5)
 93                          >      )*DT
 95          106              AMI(I,J)= (-0.5*DX1Q/RE
 96                          >          -DX1*U1*0.5)*FIUM
 97                          >        *DT
 99          126              R1(I,J)=UH(I,J,K,1)
100                     10    CONTINUE
102                           CALL TDMAJ(AMI,ACI,API,R1,R1,2,N1M,1,N2M)
103                           DO 11 J=1,N2M
104                           DO 11 I=2,N1M
105          100        11    UH(I,J,K,1)=R1(I,J)
106                      1    CONTINUE
```

< 최적화 순차코드 >

```
104                           DO 3 J=1,N2M
105                           DO 30 K=1,N3M
106                           KP=KPA(K)
107                           KM=KMA(K)
108                           DO 30 I=2,N1M
109                           IP=I+1
110                           IM=I-1
112           51              W2=0.5*(U(IM,J,KP,3)+U(I,J,KP,3))
113           10              W1=0.5*(U(IM,J,K ,3)+U(I,J,K ,3))
115           27              APK(I,K)=(
116                          >        -0.5*DX3Q/RE
117                          >        +0.5*DX3*W2*0.5
118                          >        )*DT
119           20              ACK(I,K)=1.+(
120                          >        +DX3Q/RE
121                          >        +0.5*DX3*(W2*0.5-W1*0.5)
122                          >      )*DT
123           28              AMK(I,K)=(
124                          >        -0.5*DX3Q/RE
125                          >        -0.5*DX3*W1*0.5
126                          >         )*DT
127           32              R3(I,K)=UH(I,J,K,1)
128                     30    CONTINUE
129                           CALL CTDMA3I(AMK,ACK,APK,R3,UH(0,0,0,1),J,N3M,2,N1M)
130                      3    CONTINUE
```

<OpenMP code>

```
108                  !$omp parallel do private(KP,KM,W1,W2,
109                  !$omp& APK,ACK,AMK,R3)
110                           DO 3 J=1,N2M
```

```
111                          DO 30 K=1,N3M
112                          KP=KPA(K)
113                          KM=KMA(K)
114            76            DO 30 I=2,N1M
115                          IP=I+1
116                          IM=I-1
118            78            W2=0.5*(U(IM,J,KP,3)+U(I,J,KP,3))
119                          W1=0.5*(U(IM,J,K ,3)+U(I,J,K ,3))
121            60            APK(I,K)=(
122                         >         -0.5*DX3Q/RE
123                         >         +0.5*DX3*W2*0.5
124                         >         )*DT
125           147            ACK(I,K)=1.+(
126                         >         +DX3Q/RE
127                         >         +0.5*DX3*(W2*0.5-W1*0.5)
128                         >        )*DT
129            54            AMK(I,K)=(
130                         >         -0.5*DX3Q/RE
131                         >         -0.5*DX3*W1*0.5
132                         >          )*DT
133           186            R3(I,K)=UH(I,J,K,1)
134                      30  CONTINUE
135                          CALL CTDMA3I(AMK,ACK,APK,R3,UH(0,0,0,1),J,N3M,2,N1M)
136                       3  CONTINUE
```

① parallel do 지시어를 이용해 가장 바깥쪽 루프에 대한 루프 병렬화를 수행하였다.

9. OpenMP 병렬화 결과

최적화된 순차코드를 병렬화했을 때 1 time step 기준으로 얼마만큼의 성능향상을 얻었는지 아래 표에 정리해 두었다.

	Large (257x65x129)	Parallel Speed up	Very large (513x65x129)	Parallel Speed up
Original (Serial)	16.23 s			
Tuned (Serial)	2.09 s		4.70 s	
2cp	1.19 s	1.76	2.50 s	1.88
4cp	0.62 s	3.37	1.25 s	3.76
8cp	0.37 s	5.65	0.64 s	7.34
16cp	0.21 s	9.95	0.37 s	12.70
32cp	0.17 s	12.29	0.29 s	16.21

위 결과는 환경변수 SPINLOOPTIME과 YIELDLOOPTIME을 각각 10000과 40000으로 설정해 얻은 것이다. 두 환경변수를 설정하지 않고(디폴트: SPINLOOPTIME=40, YIELDLOOPTIME=0) 코드를 실행한 결과는 아래와 같다.

	Large (257x65x129)	Parallel Speed up	Very large (513x65x129)	Parallel Speed up
Original (Serial)	16.23 s			
Tuned (Serial)	2.09 s		4.70 s	
2cp	1.21 s	1.73	2.65 s	1.77
4cp	0.66 s	3.13	1.30 s	3.62
8cp	0.42 s	4.98	1.00 s	4.70
16cp	0.74 s	2.82	1.03 s	4.56
32cp	1.30 s	1.61	1.49 s	3.15

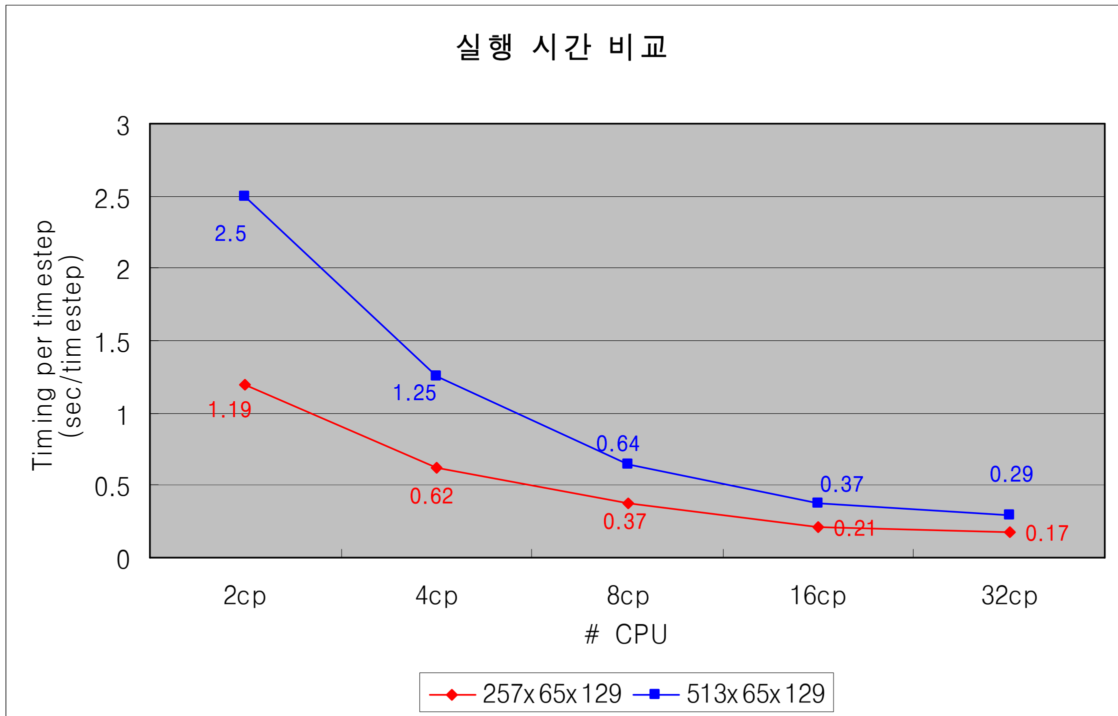


그림 II.3 실행 시간 비교

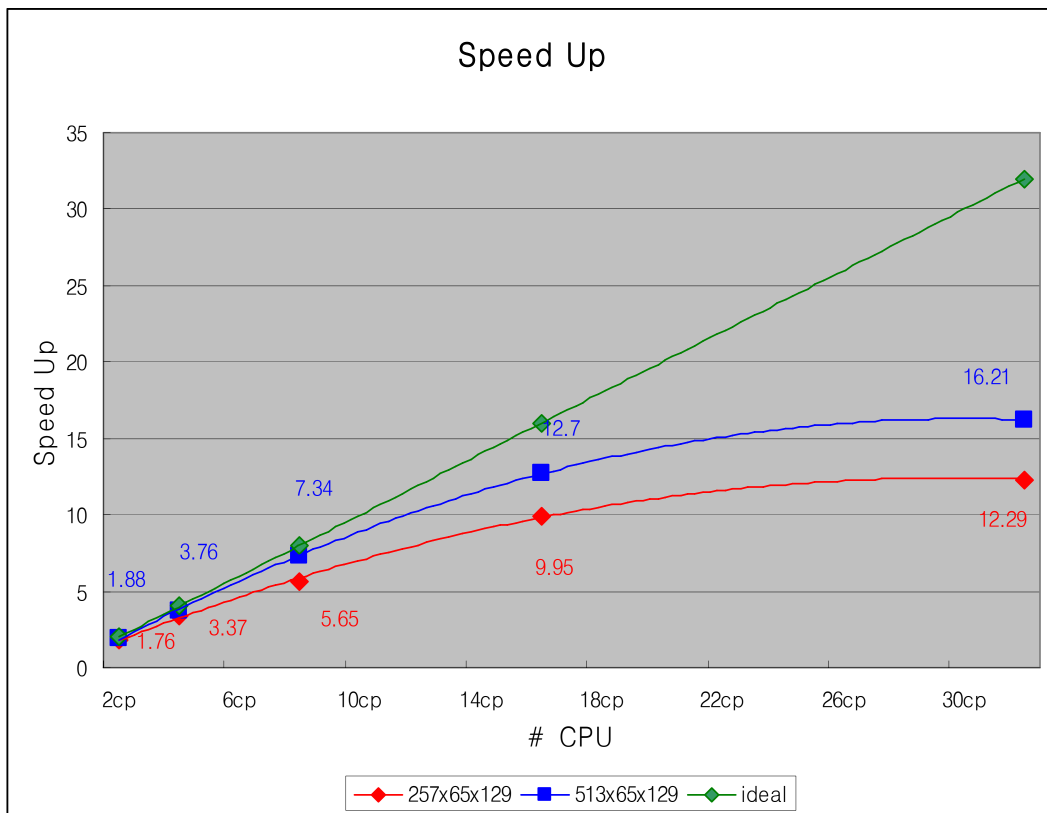


그림 II.4 병렬화 성능

참조: 환경변수 SPINLOOPTIME, YIELDLOOPTIME

스레드가 작업을 실행하지 않는 idle상태는 sleeping과 spin-waiting 상태로 구분해 볼 수 있으며, 일단 스레드가 sleeping 상태로 들어 가게 되면 해당 스레드가 재 구동되기 위해서 많은 비용(cost)이 든다. SPINLOOPTIME은 스레드가 sleeping 상태로 들어 가기 전에 spin-waiting 상태로 있게 되는 시간을 설정한다. 스레드는 설정된 시간 동안 spin-waiting 상태로 있다 sleeping 상태가 되는데 이때 YIELDLOOPTIME 값이 0보다 큰 값을 가지면 바로 sleeping 상태로 가지 않고 yield() 시스템 호출을 해서, 프로세서를 다른 스레드에게 반환하지만 여전히 실행 가능한 상태로 남아있게 된다. YIELDLOOPTIME은 스레드가 sleeping 상태로 들어가기 전에 yielding 상태로 있게 되는 시간을 설정한다. sleeping 상태에 있는 스레드가 재실행되는데 더 많은 비용이 소비되므로 dedicate하게 사용하는 시스템의 경우, 적절한 환경변수 설정으로 스레드가 가급적 sleeping 상태로 들어가지 않도록 하는 것이 코드 성능에 유리하다. 반면 많은 사용자와 프로세스가 사용하는 busy 시스템의 경우 spinning에 많은 시간이 소비되면 전체 시스템 성능을 나쁘게 할 수 있다. 관련된 최적의 조건은 프로세서 클럭 등과 같은 다양한 시스템 특성에 의존하며 실험에 의해 그 값을 찾을 수 밖에 없다. 여기에서는 SPINLOOPTIME과 YIELDLOOPTIME을 비교적 높게 설정해서 병렬 코드의 성능뿐 아니라 scalability도 좋아지는 결과를 얻을 수 있었다.

10. Summary

최적화된 순차코드는 time step 기준으로 original 순차코드 대비 7.77(문제크기: 257 x 65 x 129)배의 성능향상을 보여 주었다. 전체적으로는 주로 메모리 접근의 연속성을 높이는 코드 수정과, 성능이 우수한 Fourier 변환 루틴을 사용함으로써 향상된 성능을 얻고 있다.

OpenMP를 이용한 병렬화 작업은 parallel do를 이용한 루프 병렬화를 수행한 것이 대부분이고, 루프 계산을 통해 최대값을 얻거나 배열의 합을 얻는 부분을 병렬화하는 경우에는 parallel do와 더불어 보조 지시어 reduction을 이용하였다.