기술 보고서

ISBN:

2015년 사용자 코드 성능 분석 및 최적병렬화 지원보고서

2015.12

국가슈퍼컴퓨팅연구소
슈퍼컴퓨팅응용실
권오경, 강지훈, 류훈, 명훈주, 이승민

목 차

지원 개요 및 목록 ····· 1
1. 폴리머 채널 난류 코드 전처리 ····· 3
2. Interpolation Scaling Function ····· 4
3. 역해석 기반의 3차원 구조물 손상 계산 코드 ····· 7
4. VASP 최적화 ····· 10
5. SPH(Smoothed Particle Hydrodynamics) ····· 19
6. Free-stream 난류경계층해석 ····· 21
7. 채널코드 후처리 병렬화 ····· 24
8. 다체계에서 핵의성질 병렬화 ····· 25
9. WRF OPENACC 최적화 ····· 27
10. NCEP CFS 기후모델 Tachyon2 최적화 ····· 30

2015년도 사용자 코드 성능 분석 및 최적병렬화 지원보고서

○ 지원 개요

- 슈퍼컴퓨터 5호기 IBM 및 SUN 시스템 각각에 최적화된 사용자 코드 성능 분석 및 병렬확장성을 고려한 병렬화 지원
- 2011년부터 성능최적화 내부 인력 확보와 KISTI 상주 인력들과의 유기적 협조를 통한 업무 수행
- 연구지원프로그램 혹은 사용자의 요청을 통한 최적화 병렬화 대상 접수
- 대상 프로그램 사용자과의 협력 회의/이메일/전화 등을 통해 병렬화에 필요한 프로그램 정보 획득

<표 1>2015년 최적화/병렬화 지원 현황

순번	분류	연구자	소속	코드명	진행 내용	결과
1	공동연구	김경연	한밭대	폴리머 채널 난류 코드 전처리	초기생성 MPI 병렬화	2D 영역분할을 통한 4096코어에서 수행가능
2	공동연구	성형진	KAIST	Taylor-Couette annular flow 코드	하이브리드 (MPI+OpenMP) 병렬화	병렬화 개발 완료
3	공동연구	김우연	KAIST	Interpolation Scaling Function	CPU/GPU/Xeon Phi 병렬화	4608코어에서 1024코어 대비 3.08배 성능 향상
4	공동연구	우상욱	부경대	생체거대분자 구조	LAMMPS 코드 기반 MPI 병렬화	순차 코드 준비중 (12월 개발 완료)
5	공동연구	강준원	홍익대	역해석 기반의 3차원 구조물 손상 계산 코드	MPI 병렬화	순차코드 대비 12배 향상(32코어) / symmetric row scale preconditioning을 통한 선형계산 안정도 150배 향상
6	공동연구	유승화	KAIST	나노와이어 해석 코드 병렬화	GPU 병렬화	41배 성능향상
7	공동연구	최정일	연세대	멀티 블록 기반 병렬 프레임워크 개발	MPI 병렬화	3D 영역분할을 통한 개발완료
8	공동연구	김우연	KAIST	RS-DFT 슈뢰딩거 솔버 최적화	성능최적화	다중레벨 병렬화를 통한 중복고유값 계산성능 logN배 향상(N: CPU 서브그룹 개수)
9	신청	전체	-	VASP 최적화	성능최적화	1.11배(Intel), 1.74배(PGI), 2.89배(GCC) 성능향상

10	신청	정민중	KISTI	SPH	OpenMP 병렬화	2.92배 성능 향상
11	신청	성형진	KAIST	Free-stream 난류경계층해석	MPI 병렬화	6,144코어까지 병렬확장성 확보 / MPI DDT 활용 19% 병렬성능 향상 / 동적할당을 통한 90% 평균메모리감소
12	신청	전체	-	Intel MPI Benchmarks 최적화	성능최적화	옵션 조정을 통한 2.47배 성능 향상
13	신청	최정일	연세대	채널코드 Post-processing 병렬화	후처리 MPI 병렬화	2D 영역분할을 통한 2048코어에서 수행가능
14	신청	하은자	숭실대	다체계에서 핵의성질	MPI 병렬화	순차코드대비 576배 성능 향상(64코어)
15	신청	조민수	KISTI	WRF 가속기 병렬화	OpenACC 성능최적화	MPS 적용으로 단일노드에서 1.82배(20코어), 멀티노드에서 5.24배(2노드/10ppn) 향상
16	신청	서경환	부산대	NCEP CFS 기후모델 Tachyon2 최적화	성능최적화	AIX 내부/ESSL/XLF 함수를 제외하고 Tachyon2로 이식

1. 폴리머 채널 난류 코드 전처리

- 코드 개요
  - 작년에 개발한 폴리머 채널 난류코드를 대규모 도메인에서 수행하기 위한 전처리 순차 코드의 병렬화
  - Fortran으로 작성되어 있으며 Spectral영역의 작은 규모의 3차원 도메인(128,127,64)을 큰 규모의 도메인(1536, 383, 3072)로 확장하는 코드
  - 코드는 크게 세가지 단계로 진행
    1. Spectral에서 Physical 영역으로 변환
    2. Interpolation
    3. Physical 영역에서 Spectral 영역으로 변환
  - 변환해야 할 각 변수는 10개
- 개발 내용
  - 순차 코드에서는 메모리 부족(한 변수당 57G가 필요)으로 x와 z방향으로 2차원 영역 분할로 병렬 설계
  - 각 단계 모두 x와 z방향으로 2D FFT수행이 필요하므로, 나눠진 영역에 대해서 각각 전치 통신(MPI_Alltoall)을 한 뒤 해당 축에 대해서 1D FFT 수행
  - 메모리 최적화를 위해서 전치시 메모리 구조를 MPI DDT(Derived Data Type)을 통해서 최적화
  - 대규모 코어(1000코어이상)을 지원하기 위해서 입력파일과 출력파일에 대해서 각 변수별로 다른 디렉토리에서, 각 코어(MPI 랭크)별로 별도의 파일 입출력을 수행
- 결과
  - 순차 코드에서는 메모리 부족으로 수행할 수 없었지만 병렬화를 통해서 2048과 4096코어에서 수행 성공
  - MPI DDT 최적화로 병렬화 대비 약 10% 성능향상

2. Interpolation Scaling Function

• 코드 개요

- 자유경계조건 상에서 푸아송 방정식을 적은 격자수로 원하는 정확도를 효과적으로 풀 수 있는 방법인 Interpolation Scaling Function 코드 개발
- 이러한 문제를 풀기 위한 코드는 외산 소프트웨어에 의존성이 강하며, 이에 대해 국산화하기 위해 KAIST 김우연 교수연구실과 공동 개발 수행 (KAIST 김우연 교수연구실은 솔버의 알고리즘 개발, KISTI 최적병렬화 팀은 해당 개발물의 최적화 및 병렬화 수행)
- 현재까지 Interpolation Scaling Function 방법을 활용하여 수행한 수치 실험에서 사용한 최대 CPU수는 4000개 수준이고 이중 실 제 병렬 성능이 나오는 크기는 약 500개 수준 (1차원으로 병렬화 수행을 했기 때문에 병렬확장성이 나오지 않는 상황)

• 개발 내용

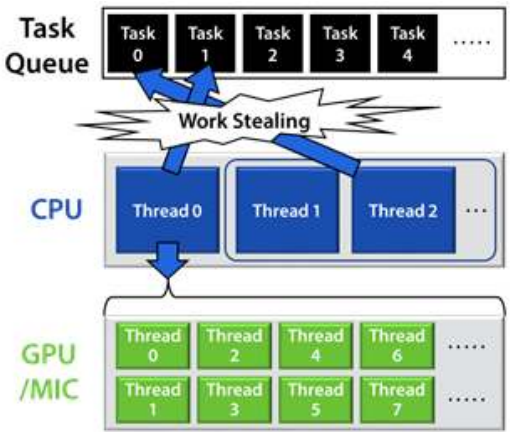
- 하이브러리 병렬화(MPI+OpenMP) 개발

  · MPI를 이용해 이중으로 반복문을 병렬화하고, 내부 반복문은 OpenMP를 활용하여 병렬화해 총 3차원으로 병렬화 수행하여 병렬 성능이 4000개 이상의 코어에서 나올 수 있게 구현

- 가속기 기반 고성능시스템에서 활용할 수 있게 구현

  · OpenMP를 활용해서 CPU와 다중의 가속기 카드를 동시에 활용할 수 있게 구현

  · 밑에 그림처럼 CPU와 가속기 비율에 대해 로드밸런싱을 정적으로 하지 않고 동적으로 사용하기 위해 work-stealing 모델 적용



<그림> CPU와 가속기를 효율적으로 동시에 활용하기 위해 work-stealing 방법 적용

- NVIDIA GPU 카드 지원을 위해서 CUDA로 개발

· 행렬 곱셈 계산시 최적화를 위해서 CUDA의 stream을 활용하여 cublas의 dgemm연산 수행시 최적화 수행

- Xeon Phi 가속기 카드 지원을 위해 Native와 Offload 모드 개발

· CPU 및 Phi 계산시 벡터화를 통해서 최적화 수행

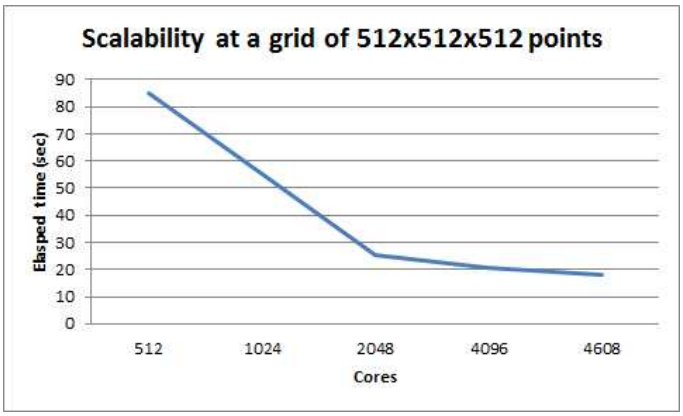
· 행렬 곱셈 계산시 최적화를 위해서 MKL의 다중 dgemm 함수 호출을 최적화해서 지원하는 cblas_dgemm_batch 함수(Intel Cluster Studio 2016부터 지원)를 활용

· 비동기통신을 활용하여 계산과 통신 중첩을 통한 최적화 수행

• 결과

- 목표한 크기의 계산을 4608 코어 이상까지 병렬확장성(strong-scalability) 확보

· 1024코어 대비 3.08배 성능 향상

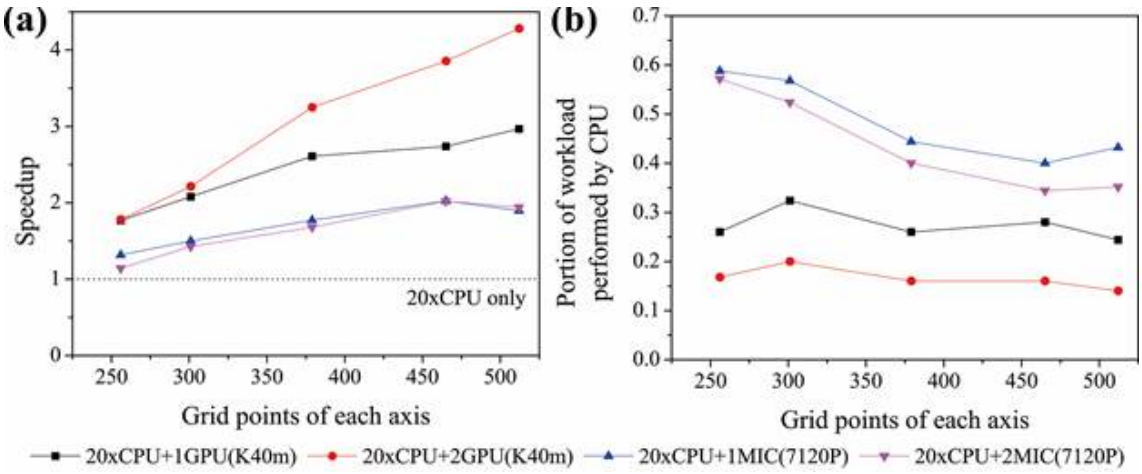


<그림> 512x512x512 포인트 격자에서
4608코어에서 병렬확장성 확보

- CPU기반의 코드를 가속기로 포팅 및 최적화

· NVIDIA GPU카드(K40)는 500x500x500포인트 격자 기준으로 20코어 CPU 사용대비 2장 사용시 약4배, 1장 사용시 약 3배 성능 향상

· Xeon Phi카드(7102P)는 500x500x500포인트 격자 기준으로 20코어 CPU 사용대비 1.7배 성능향상

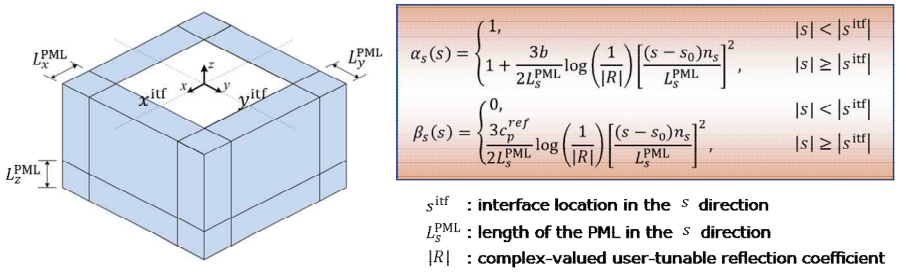


<그림> Interpolation Scaling Function의 가속기 성능결과 (a: 격자 포인트수별 20코어 CPU대비 성능향상도, b는 격자 포인수별 CPU의 워크로드 비율)

3. 역해석 기반의 3차원 구조물 손상 계산 코드

• 코드 및 지원 요청 개요

- 탄성파, 초음파, 마이크로파 등의 측정파형을 분석하여 사회기반 구조물의 균열, 공극, 강도저하 등 손상상태를 정밀하게 진단하는 기술의 중요성이 공공안전 및 구조물의 유지관리 차원에서 지속적으로 증대되고 있음. 이를 위한 방법으로서 측정 신호의 전체파형 역해석 (full-waveform inversion) 기법이 최근 주목받고 있는데, 일반적으로 역해석은 진단 대상 구조물에 대한 파동전파 정해석을 반복함으로써 수행됨
- 수치해석 기반의 3차원 파동전파 정해석은 소요 메모리와 계산시간이 커서 단일 프로세서로는 정밀하고 효율적인 계산이 어렵고, 이를 반복하는 역해석의 경우에는 이러한 문제가 더욱 심화됨
- 본 코드의 최적병렬화 지원는 홍익대학교 토목공학과의 강준원 교수님의 제안요청서를 바탕으로 시작되었으며, 3차원 파동전파의 유한요소해석을 시간에 따라 반복하는 역해석 루틴 병렬화 및 선형 시스템 행렬의 stability를 개선하기 위한 conditioning 최적화가 공동연구의 주요 목적

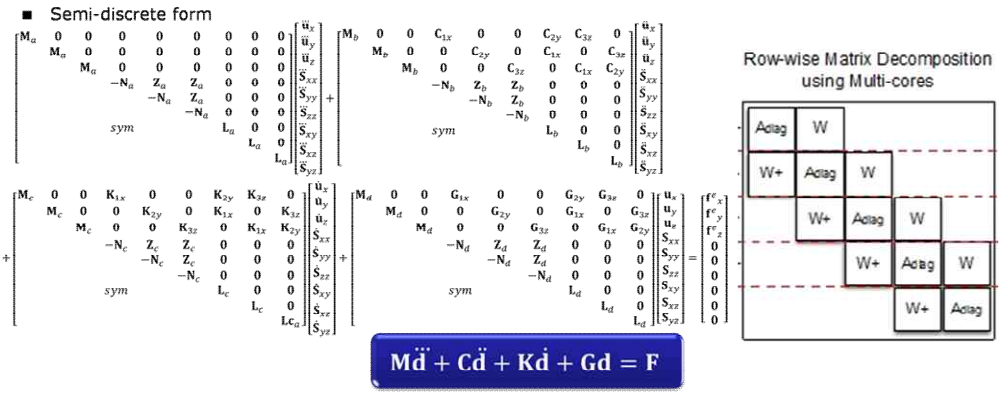


<그림> PML을 이용한 구조물 형상구축 개요 및 파형해석 지배방정식

· 수행방법

- 주요 시스템 행렬의 저장 알고리즘 변환을 통한 메모리 소요량 감소
  * 탄성파 해석을 위한 3계도 미분방정식의 선형화 및 그 결과 (M/C/K/G) 요소행렬을 Compressed Sparse Row 자료구조로 변환해 저장 <그림 2>
  * 행 기준의 (Row-wise) 요소 행렬 Decomposition을 통한 도메인 병렬화
- BiConjugate Gradient STABilized (BiCGSTAB) 알고리즘을 이용한 선형시스템 해의 순환 병렬 계산 (Iterative Parallel Computing) 모듈의 구현
  * 일반적인 CG에 비해 수렴성이 높으며 대칭 행렬로 이루어진 선형시스템 해를 빠른 시간 안에 구할 수 있다는 장점이 있지만, 행렬의 condition number가 일정 이상 값이 되면 (안정도가 낮음: singular에 가까워짐을 의미) Jacobi iteration에 비해, 쉽게 Breakdown이 발생해 해를 풀지 못하는 경우가 생기는 단점이 있다. (이론적으로는 condition #가 무한대가 아니면 해가 존재)
  * 선형시스템 행렬이 대칭인 경우 CG 등 condition number에 덜 민감한 알고

리즘이 존재하나 수렴 속도가 느려 문제가 된다. 본 코드에서는 time step을 적어도 5000번 이상 두고 반복적으로 해석을 수행해야 하므로 single time-step 에서의 해석 속도가 가급적 빨라야 한다는 제한 조건이 존재하여, BiCGSTAB 알고리즘을 선형시스템 계산의 방법으로 채택하였다.

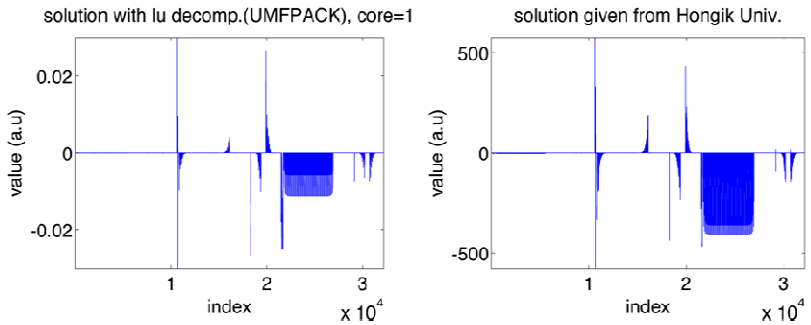


<그림> M/C/L/G 행렬의 패턴 및 Row-wise Decomposition의 개념도

- Row-scaling 방법에 기반한 대칭 Gauss-Siedel Preconditioning 루틴의 개발

  * M/C/L/G 중 전체 system 행렬 (M+C+K+G) 의 condition number를 높이는 이유가 대부분 M에 있다는 사실을 확인하고, 일반적으로 대칭행렬에 효과적으로 알려진 symmetric Gauss-Siedel (GS) Preconditioning 모듈을 개발해 적용하였다. 위의 <그림>에서 보듯, M 행렬은 전체 행렬의 상위 1/6 정도의 블락에만 분포하므로, 대칭 GS는 Row-symmetric-scaling 방법과 동일해짐

  * 실제로 이 경우 시스템 행렬의 condition number는 매우 높은 편이다 (on the order of 10^{13}). 하지만 그 값이 Inf가 아니므로 이론적으로 선형시스템의 솔루션은 존재하여, direct 방법 (LU Decomposition) 으로 풀면 해를 얻을 수 있다. 그러나 preconditioning 없이 iterative 방법으로 풀면 솔루션이 달라짐을 확인할 수 있다. <아래 그림>



<그림> 파일럿 문제에 대해 direct 방법 (좌), iterative 방법으로 전처리없이 푼 결과 (우)>

· 결과

- 테스트 환경

  * Intel parallel studio 2015
  * 노드당 20 Xeon E5-2680 v2 CPUs (2.8GHz), 256GB Memory, 2노드
  * 문제크기: 선형행렬 DOF 166509 and 354051
  * Time step 5000개

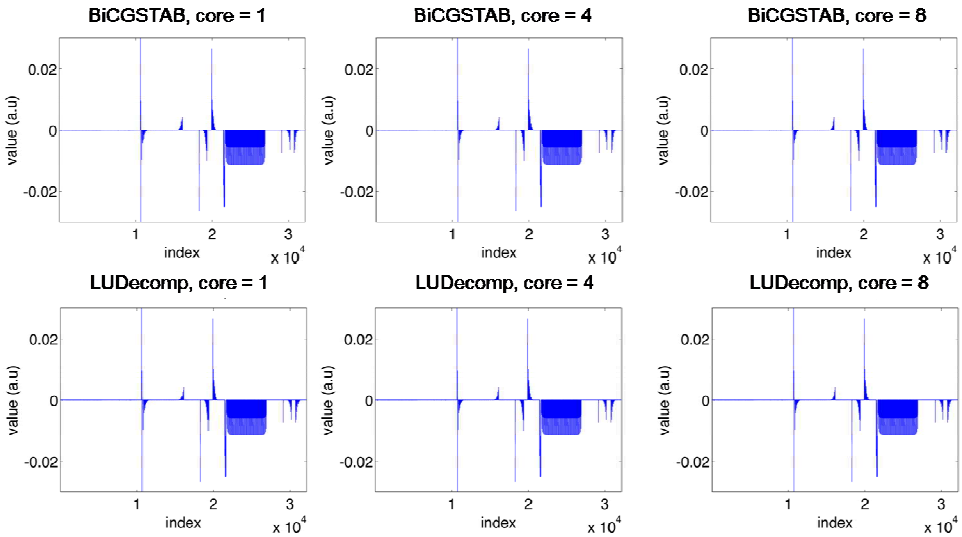
- 결과 (CPUs vs. 계산시간)

# of CPUs	Wall-time (sec)	
	DOF = 166509	DOF = 354051
1 [a)]	55100	129200
2	49850	116800
4	31400	74100
8	20100	46800
16	11850	26450
32	8150	16650

[a)] Direct Method (LU Decomposition) 사용하면 Memory Overflow

* Direct Method로는 이미 풀기 어려운 수준이지만, 큰 CPU #에서의 확장성을 테스트하기에는 문제 수준이 아직 작음. 풀어본 적이 없는 더 큰 수준의 문제 구성 (input deck)을 홍익대 측에서 마치면 strong scalability의 stress test를 보강할 계획

* Preconditoning의 효과로 condition number를 약 150배 정도 감소시키는 데 성공하였음. 작은 문제 (그림 3에서 푼 파일럿 문제) 에 대해 적용해 계산해 보았을 때 솔루션의 정확성이 reference 솔루션과 일치함을 확인함. <그림 4>



<그림> 파일럿 문제에 대해 direct 방법 (아래), 개발된 솔버로 푼 결과 (위)

4. VASP 최적화

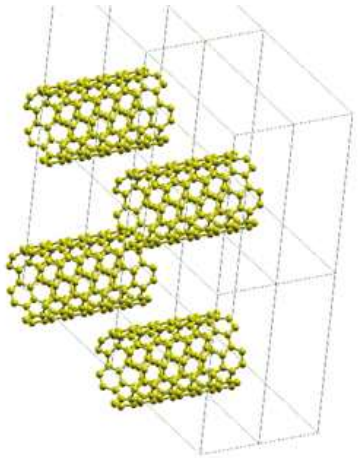
· 코드 지원 개요

- 슈퍼컴퓨터에서 빌드 환경 및 설정에 따른 VASP(Vienna Ab initio Simulation Package)의 성능을 비교 테스트. 이를 통해 최적화 빌드 방법을 제공

· 수행방법

- 테스트 시스템 : 타키온2
- 버전 : VASP 5.3.5
- 테스트 계산 모델

탄소나노튜브(6,6)의 SCF(Self Consistent Field) 1회 계산
Unit-cell에 72개의 탄소 원자 포함
288개의 전자(core electron 제외, pseudo potential 방법 적용)
튜브 축 방향 8개 k-point 계산
Energy cutoff 는 400 eV로 설정



<그림> 탄소나노튜브(6,6) Unit-cell 구조

- 빌드방법

* VASP 빌드를 위해서는 컴파일러와 수치 라이브러리가 필요함. 타키온2 시스템에는 여러 종류의 컴파일러와 수치 라이브러리가 설치되어 있으며 이 중 적절한 조합을 선택하여 빌드해야 함

* 추천 컴파일러 및 라이브러리

· 타키온2 시스템에서 사용 가능한 컴파일러는 Intel, PGI, GNU 컴파일러 있다. 설치된 버전 중 최신 버전을 기준으로 비교하였을 때 PGI 컴파일러를 사용한 경우가 성능이 좋음. 하지만 최적화 라이브러리를 사용하는 경우에는 전반적으로 Intel 컴파일러를 사용하였을 때의 성능이 가장 좋음. 따라서 VASP 빌드 시 Intel 컴파일러의 최신 버전을사용하는 것을 권장. Intel 컴파일러를 사용한 경우가 GNU 대비 약 1.38배 좋은 성능을 나타내고 있음

· VASP에 사용되는 라이브러리는 Linear algebra (LAPACK), FFT (FFTW3), MPI 가 있음. 이 중 가장 좋은 성능을 나타낸 것은 MKL, Intel FFTW3 interface, mvapich2 라이브러리를 사용하였을 때. 단 이 결과는 일반적은 SCF 계산에 대한 것으로 다른 계산을 수행할 경우는 최적 조합이 달라질 수 있음. mvapich2 라이브러리를 사용하는 경우 openmpi 대비 약 1.43배 좋은 성능을 나타냄

* 환경 설정

· VASP 빌드 시 필요한 컴파일러 및 MPI 환경 설정을 해야함. 타키온2 시스템에서는 module 명령을 통해 아래와 같이 설정할 수 있음

```
$ module load compiler/intel-2013 mpi/mvapich2-2.0
```

* 내부 라이브러리 코드 (vasp.5.lib) 빌드

· VASP 코드는 두 가지로 제공되는데 메인 코드와 필요한 내부 라이브러리 코드. 메인 코드 빌드 시 내부 라이브러리 코드를 참조하는 구조로 되어 있어 메인 코드 빌드 전에 반드시 내부 라이브러리 코드를 먼저 빌드 해야 함

· Intel 컴파일러를 사용하는 경우에는 "vasp.5.lib" 디렉터리 안에 "makefile.linux_ifc_p4" 파일을 복사하여 Makefile로 사용. 다른 컴파일러에 대한 Makefile 예제도 포함되어 있으니 다음과 같이 복사하여 사용할 수 있음

```
$ cp makefile.linux_ifc_p4 Makefile
```

· 추가로 복사한 Makefile에 "FC" 부분을 "ifc"에서 "ifort" 로 수정해야 한다.

전)

```
FC=ifc
```

후)

```
FC=ifort
```

· Makefile 복사 및 수정 후 make 명령어를 통해 빌드하면 됨. 혹시 기존에 빌드한 파일이 남아 있는 경우 object 파일과 라이브러리 파일을 삭제 후 다시 빌드하면 됨. 정상적으로 빌드가 되었으면 "libdmy.a" 파일이 생성된다.

```
$ rm *.o *.a
$ make
```

* 메인 코드 (vasp.5.3.5) 빌드

· vasp.5.lib 빌드 후 vasp.5.3.5 디렉터리로 이동 후 메인 코드를 빌드하면 된다. 메인 코드 역시 컴파일러 별 Makefile 예제를 제공하고 있으며 Intel 컴파일러를 사용할 때는 "makefile.linux_ifc_p4" 파일을 복사 후 수정하여 사용하면 됨

- 빌드 최적화 및 성능 테스트

* 컴파일러 옵션

· Main loop를 담당하는 계산이 전체 계산 시간의 대부분을 차지한다. 컴파일러별로 최적화 옵션을 적용하면 상당 부분 성능 향상을 얻음. 특정 최적화 옵션

의 경우 결과에 영향을 줄 수 있으나, 심각하게 영향을 주는 옵션은 발견되지 않음

		O0	O1	O2	O3	O2_opt	O3_opt
LOOP	POTLOK	2.43	2.04	1.74	1.83	1.81	1.66
	SETDIJ	0.23	0.20	0.24	0.21	0.17	0.09
	EDDAV	405.98	376.47	374.35	368.12	369.50	364.56
	DOS	0.01	0.06	0.08	0.11	0.03	0.01
EDIFF	CHARGE	17.78	16.34	16.44	16.14	15.75	15.59
	FORLOC	1.72	0.87	0.80	0.79	0.75	0.74
	FORNL	55.76	52.96	52.86	52.29	51.95	51.88
	STRESS	171.38	159.73	158.95	157.47	156.51	156.10
	FORHAR	2.26	1.28	1.22	1.23	1.16	1.13
	MIXING	0.12	0.05	0.13	0.08	0.17	0.05
	OFIELD	0.00	0.02	0.01	0.01	0.01	0.00
LOOP+EDIFF		657.67	610.02	606.82	598.28	597.81	591.81
Elapsed time		710.84	660.89	660.10	648.51	648.13	639.47

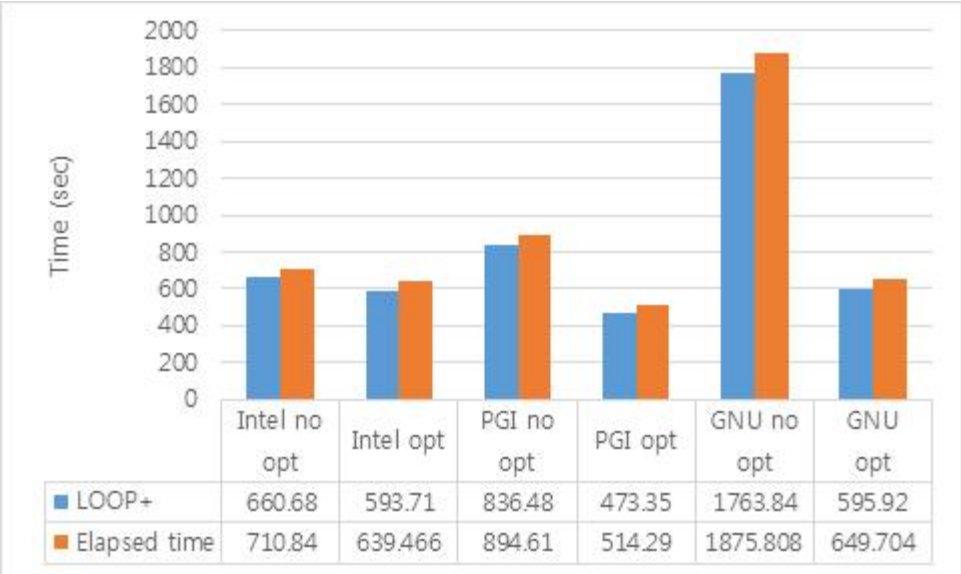
<표> Intel 2013 컴파일러 옵션 별 성능 (Opt option : -O3 -ip -ftz -xSSE4.2)

		O0	O1	O2	O3	O4	O3_opt	O4_opt
LOOP	POTLOK	3.31	3.12	2.11	2.10	2.10	2.07	2.11
	SETDIJ	0.17	0.16	0.22	0.27	0.19	0.14	0.17
	EDDAV	459.29	433.71	315.98	309.03	310.74	309.23	305.68
	DOS	0.01	0.14	0.13	0.09	0.14	0.07	0.10
EDIFF	CHARGE	26.95	23.97	11.36	11.11	11.41	11.43	11.29
	FORLOC	1.23	1.13	1.15	1.12	1.15	1.13	1.13
	FORNL	84.26	75.17	37.15	37.18	36.79	37.38	36.70
	STRESS	254.52	227.85	113.85	113.27	112.66	115.28	111.82
	FORHAR	2.89	1.80	1.60	1.61	1.58	1.55	1.60
	MIXING	0.12	0.08	0.11	0.13	0.12	0.12	0.15
	OFIELD	0.00	0.01	0.00	0.00	0.01	0.00	0.00
LOOP+EDIFF		832.75	767.14	483.66	475.91	476.89	478.40	470.75
Elapsed time		894.61	827.29	527.90	520.38	521.21	522.69	514.29

<표> PGI 2014 컴파일러 옵션 별 성능

		O0	O1	O2	O3
LOOP	POTLOK	4.72	2.05	1.92	1.71
	SETDIJ	0.21	0.25	0.21	0.14
	EDDAV	912.61	481.88	458.49	442.16
	DOS	0.01	0.07	0.10	0.01
EDIFF	CHARGE	61.66	12.47	10.32	8.89
	FORLOC	1.94	0.87	0.87	0.75
	FORNL	192.84	45.91	40.29	34.77
	STRESS	581.35	138.76	109.89	104.28
	FORHAR	3.01	1.35	1.17	1.14
	MIXING	0.15	0.05	0.05	0.05
	OFIELD	0.00	0.01	0.00	0.00
LOOP+EDIFF		1758.50	683.67	623.31	593.90
Elapsed time		1875.81	746.44	681.32	649.70

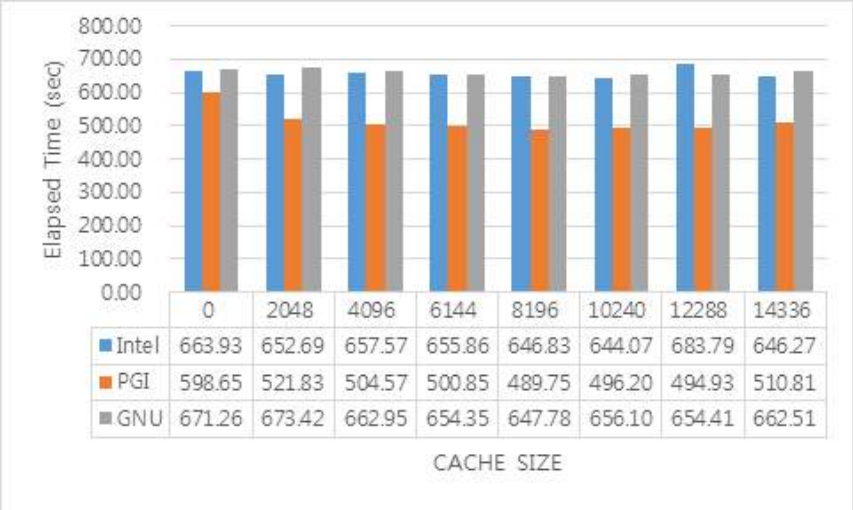
<표> GNU 컴파일러 (4.4.6) 옵션 별 성능



<그림> 컴파일러 별 성능 비교 (PGI가 제일 좋음)

* 캐쉬 사이즈

· 내장된 FFT 루틴에서 사용하는 캐시 크기로 외부 FFT 라이브러리를 사용할 경우에는 적용되지 않음. 테스트 결과 캐시 크기에 따라 큰 성능 차이는 나타지 않으나, 8096 정도로 설정했을 경우에 가장 좋은 성능을 나타냄



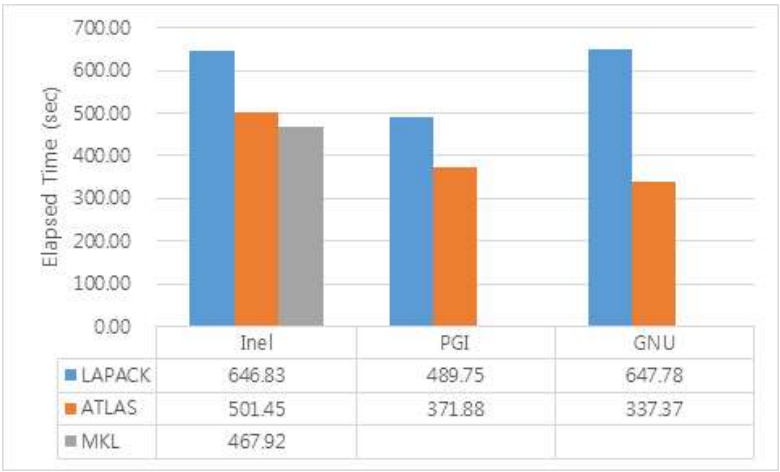
	0	2048	4096	6144	8196	10240	12288	14336
Intel	663.93	652.69	657.57	655.86	646.83	644.07	683.79	646.27
PGI	598.65	521.83	504.57	500.85	489.75	496.20	494.93	510.81
GNU	671.26	673.42	662.95	654.35	647.78	656.10	654.41	662.51

<그림> 캐쉬 크기에 따른 컴파일러 별 성능 비교

* 최적화 라이브러리

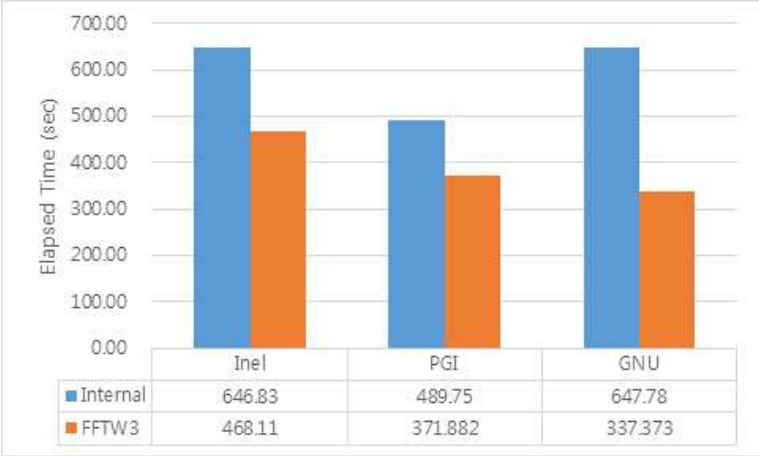
· LinearAlgebra의 성능은 Intel 컴파일러 에서는 MKL, PGI와 GNU 컴파일러에서는 ATLAS가 우수하다. 특히 해당 GNU 컴파일러로 빌드한 ATLAS의 성능이 가장 우수함. 또한 VASP 내부의 FFT 루틴 보다는 FFTW3를 링크하여 사용하는 것이 더 좋은 성능을 나타냄. Intel 컴파일러의 경우 MKL과 FFTW3 라이브러리를 사용하지 않는 경우에 성능이 매우 떨어짐. 따라서 최적화 라이브러리를 하나씩 적용하였을 때는 PGI나 GNU 컴파일러의 성능이 더 좋게 나오는 반면, MKL과 FFTW3 라이브러리를 모두 적용하였을 때는 Intel 컴파일러의 성능이 가장 우수한 것으로 나타남

· IntelFFTW3interface루틴을 사용한 경우 FFTW3를 사용한 경우보다 성능이 좋지 않음. 하지만 병렬 버전의 경우는 Intel FFTW3 interface가 조금 더 좋음

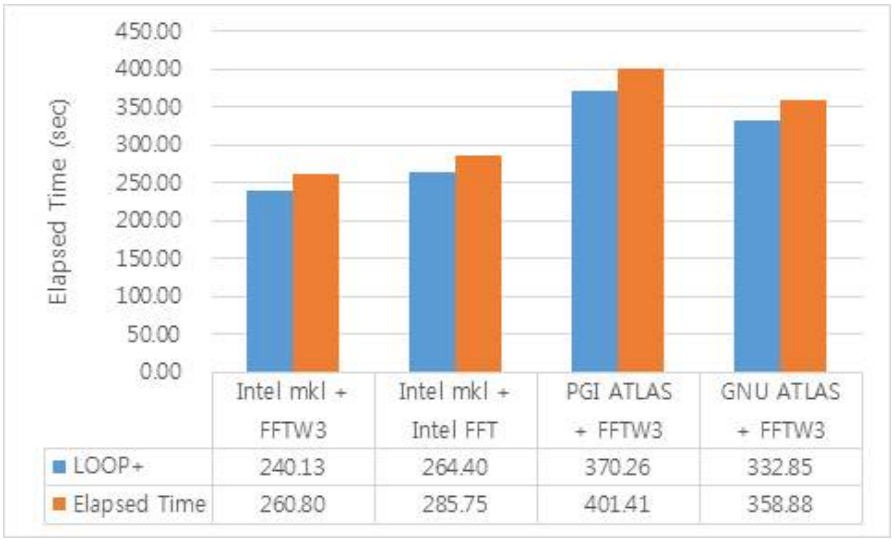


	Inel	PGI	GNU
LAPACK	646.83	489.75	647.78
ATLAS	501.45	371.88	337.37
MKL	467.92		

<그림> Linear Algebra 라이브러리(LAPACK, ATLAS, MKL) 별 성능



<그림> FFT (Interal, FFTW3) 라이브러리별 성능



<그림> 컴파일러 별 최적화 라이브러리 성능 비교

· GOTOBLAS의 경우는 링크하여 사용할 경우 결과 값이 달라짐

· 행렬 계산을 벡터 계산으로 치환하는 옵션은 컴파일러나 BLAS 라이브러리 종류에 관계없이 성능에 거의 영향을 주지 않음을 확인 (-DRPROMU_DGEMV, -DRACCMU_DGEMV)

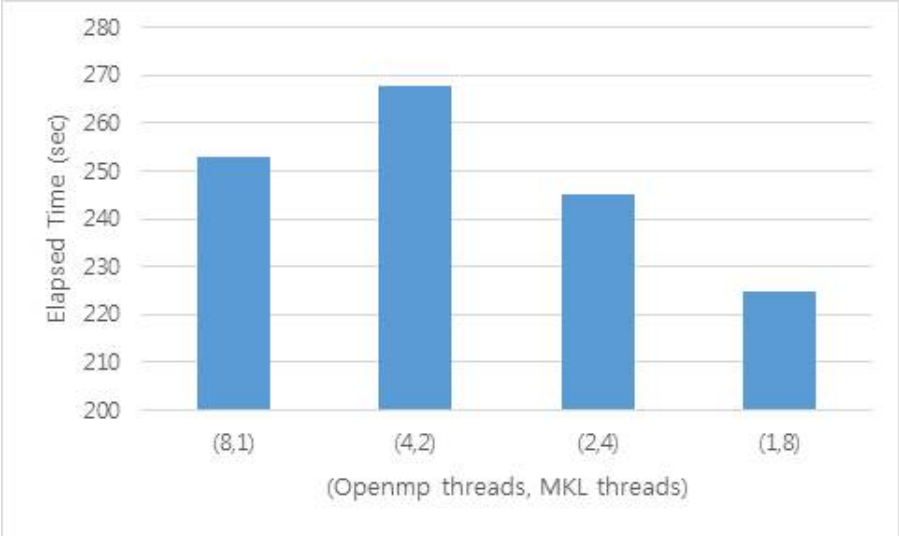
* 쓰레드 병렬화

· Thread 병렬을 지원하는 라이브러리를 사용하여 8 CPU에서 계산을 수행할 수 있음. MKL은 OpenMP thread 보다는 Intel에서 제공하는 MKL thread를 사용하는 것이 성능이 더 좋음. FFT 라이브러리의 thread 병렬화는 성능향상에 도움이 되지 않고, ATLAS의 thread 병령 성능이 MKL 보다 더 좋음

· 하지만 기대보다는 성능향상이 미미한 수준이어서 hybrid 계산에 의한 병렬 계산은 효율이 떨어질 것으로 예상. 다만, 메모리 문제로 MPI 프로레스를 노드

에 전부 할당하지 못하는 경우에는 여유 CPU 자원을 활용하는 방법으로 사용이 가능

<그림> 라이브러리 별 thread 병렬 성능 비교 (IFFT : Intel FFTW3 interface)



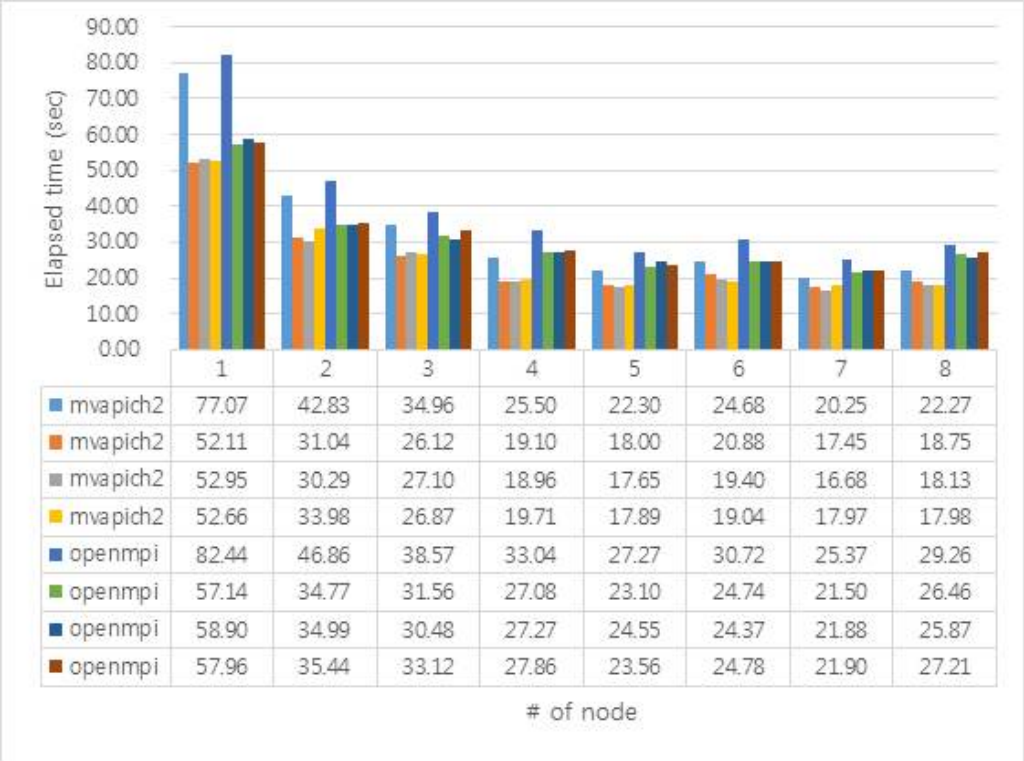
<그림> MKL thread 병렬 성능

* MPI 병렬화

· 결과는 mvapich2의 성능이 openmpi 보다 좋음. MPI 병렬 계산 (mvapich2)에서는 FFT 라이브러리는 큰 차이는 없으나 intel FFT를 사용하는 것이 성능이 나음

· mvapich2, openmpi

Linear Algebra는 MPI 병렬과 무관하므로 가장 성능이 좋은 MKL에 대하여 테스트

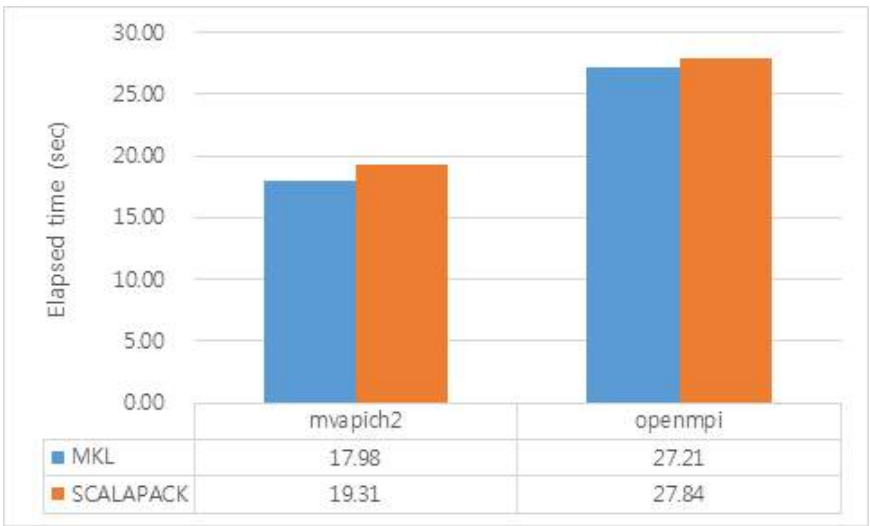


	1	2	3	4	5	6	7	8
mvapich2	77.07	42.83	34.96	25.50	22.30	24.68	20.25	22.27
mvapich2	52.11	31.04	26.12	19.10	18.00	20.88	17.45	18.75
mvapich2	52.95	30.29	27.10	18.96	17.65	19.40	16.68	18.13
mvapich2	52.66	33.98	26.87	19.71	17.89	19.04	17.97	17.98
openmpi	82.44	46.86	38.57	33.04	27.27	30.72	25.37	29.26
openmpi	57.14	34.77	31.56	27.08	23.10	24.74	21.50	26.46
openmpi	58.90	34.99	30.48	27.27	24.55	24.37	21.88	25.87
openmpi	57.96	35.44	33.12	27.86	23.56	24.78	21.90	27.21

<그림> Intel 컴파일러의 라이브러리 별 성능 (PFFTW3 : 병렬버전의 FFTW3, Intel FFT : Intel FFTW3 interface)

· SCALAPACK

LAPACK의 MPI 버전인 SCALAPACK 성능을 테스트. LAPACK 보다 MKL, ATLAS 등의 최적화 라이브러리의 성능이 월등히 좋기 때문에 SCALAPACK 은 성능향상에 도움이 되지 않음

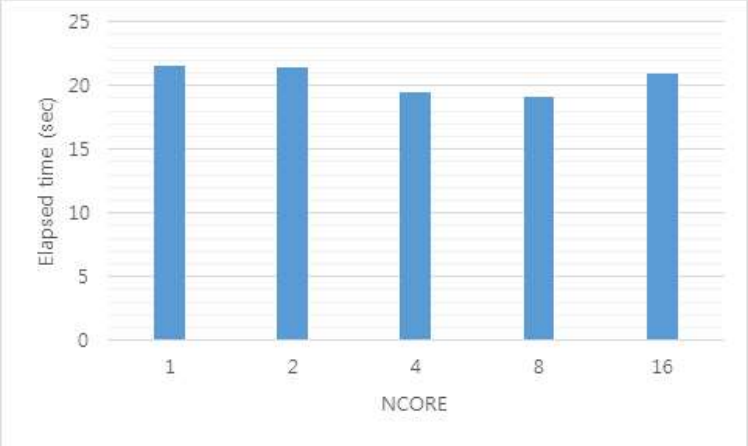


	mvapich2	openmpi
MKL	17.98	27.21
SCALAPACK	19.31	27.84

<그림> SCALAPACK 병렬 성능

· VASP 병렬 설정

VASP 입력 값 중에 병렬 계산과 관련된 설정들이 존재. NCORE, NPAR, KPAR 등의 값을 직접 입력할 수 있음. 이 값들의 의미는 VASP 매뉴얼에 나와있어 권장하는 값으로 설정하는 것이 나음. 노드 당 코어수로 설정하도록 권장하는 NCORE 값에 대하여 테스트한 결과 8로 설정했을 때 성능이 가장 좋음



<그림> NCORE 값에 따른 계산 시간 비교

- 결론

* SCF 계산은 동일한 계산을 반복하여 수렴시키는 것으로 반복 계산은 동시에 계산이 불가능. 따라서 본 문서에서는 하나의 반복 계산에 대한 성능을 기준으로 작성. 테스트 모델은 매우 잘 알려진 나노 구조 (탄소나노튜브)에 대하여 수행하였고, 계산 파라미터는 실제 연구에서 사용하는 일반적은 값들로 정함

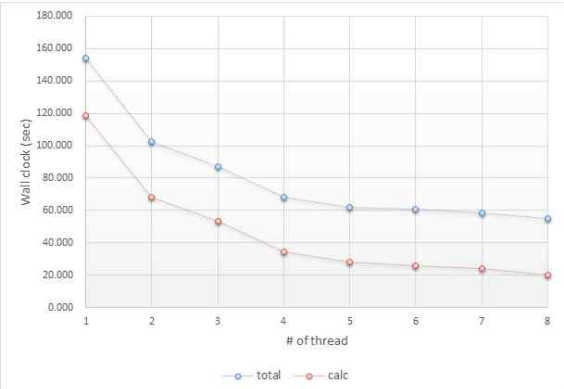
* 최적화 라이브러리를 사용하지 않은 순차계산은 PGI 컴파일러를 사용한 경우 성능이 가장 좋지만 MKL과 FFTW3를 사용하면 인텔 컴파일러를 사용하는 경우가 순차계산 성능이 가장 좋게 나옴. Thread 병렬화된 라이브러리를 사용하는 경우 약간의 성능향상 효과를 얻을 수 있지만 사용하는 CPU 자원에 비해 효율이 떨어짐. 부득이하게 메모리 문제로 MPI 프로세스를 전부 할당하지 못하는 경우 추가로 성능향상을 얻는 정도로 사용하는 것이 좋음

* MPI 병렬 성능은 openmpi 보다는 mvapich2 라이브러리를 사용하는 경우가 성능이 더 좋음. 또한 병렬계산 성능은 Intel 컴파일러, MKL,IntelFFT를 사용하는 경우 최대 성능. 또한 FFT 라이브러리는 병렬 계산의 경우 thread나 mpi 병렬 버전의 성능이 좋지 않음. SCALAPACK은 최적화 라이브러리 (MKL, ATLAS) 등을 사용하는 것에 비하여 병렬화 효과가 크지 않아 성능 향상에 도움이 되지 않음. 다만 코어를 매우 많이 사용하는 경우에는 도움이 될 여지가 남아 있음

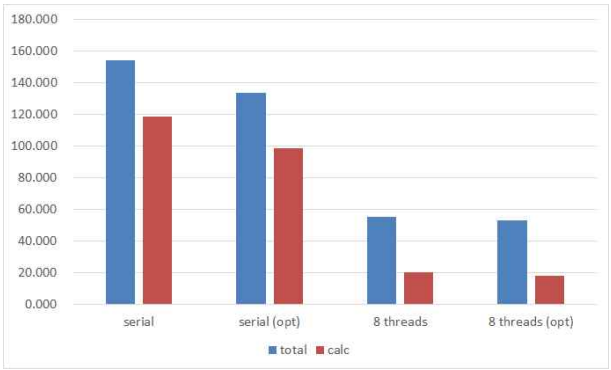
* VASP 입력 값 중에 병렬화에 관련된 값들이 있음. 이 값들을 매뉴얼에 나온 권장 값으로 설정하는 것이 성능에 도움이 됨

5. SPH(Smoothed Particle Hydrodynamics)

- 코드 개요
  - C++ 언어 사용
  - 각 시간 간격에 대한 결과 파일이 다른 이름으로 저장되고, 계산 결과는 다음 계산에 영향을 주기 때문에 병렬화가 불가함
  - 시간 간격에 대한 결과 파일을 2000 개에서 10 개로 축소하고, 입자 개수를 기존 11033 개에서 789600 개로 변경하여 테스트 수행
  - 입자 개수는 계산 영역의 크기에 의해 정해지므로 계산 영역을 확장함
- 개발 내용
  - 순차코드 프로파일링 결과 이웃 입자를 선정하는 calc_amount, 힘과 압력을 계산하는 calc_force 루틴이 각각 전체 계산 시간의 36.8%, 40.1%로 많은 부분을 차지하고 있음 (입자의 개수와 관련이 있으며 입자의 개수가 많아지면 각 계산 루틴이 차지하는 비율이 많음)
  - 입자의 위치와 속도를 파일로 저장하는 output_particles 루틴은 20.3%를 차지하고 있지만, 파일 I/O 특성상 OpenMP 병렬화가 어려울 것으로 판단하여 병렬화에서 제외
  - 루프(loop) 문을 OpenMP 병렬화가 가능하도록 수정 후 OpenMP 지시어 삽입
  - 최적화는 나눗셈을 곱셈으로 변경하여 얻음
- 결과
  - 병렬 코드는 순차 코드 대비 2.8배 정도의 성능 향상이 있음 (주요 계산 부분(calc)의 성능 향상은 5.9배 성능 향상)
  - 최종적으로 최적화를 통해 8코어를 사용할 경우 원본 코드 대비 2.92배 성능 향상(주요 계산 부분에 대해서는 8.67배 성능 향상)



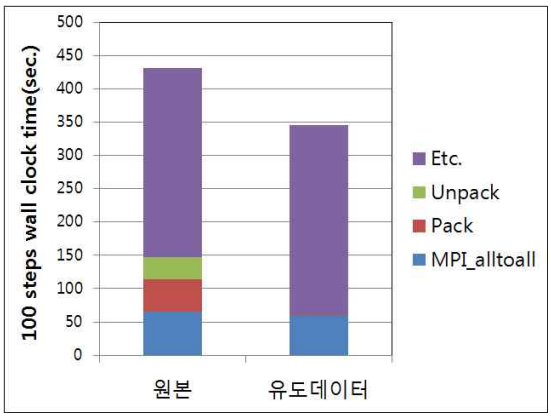
<그림> SPH 병렬코드 쓰레드별 실행시간



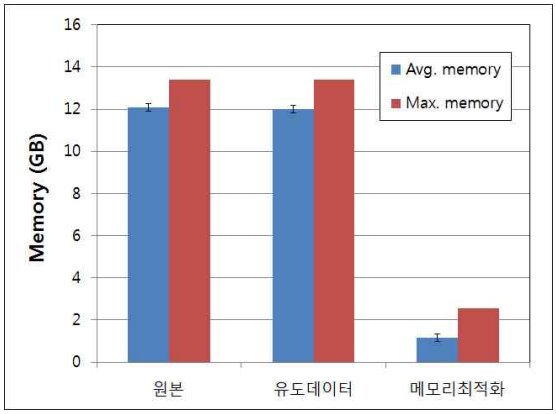
<그림> SPH 최적화코드 실행시간

6. Free-stream 난류경계층해석

- 코드 개요
  - 난류 경계층 유동에 대한 DNS 해석을 통해 매우 높은 레이놀즈수 (Ret≃2,000) 에서의 난류유동을 규명
  - 현재 세계 최대 규모의 문제는 2013년 Jiménez group이 수행한 Ret≃2,000 에서의 수치해석으로 3.37×10^{10}개의 격자와 32,768개의 코어를 사용 (Sillero et al., 2013)
  - 본 연구에서는 Tachyon2에서 비슷한 레이놀즈수의 해석을 위해 보다 현실적인 유입 경계조건을 적용하여 훨씬 적은 격자수(5×10^{9})와 계산코어(4,000~8,000)에서의 해석 수행
- 개발 내용
  - 자유흐름 유입경계조건 처리를 위해 새로 적용된 교란방정식 솔버 모듈의 병렬 코드 개발
  - 1,000코어 이상에서의 통신 성능 향상을 위해 순수 통신시간의 1.5배에 달하는 통신버퍼치환 과정을 제거할 수 있는 MPI 유도데이터 타입 구현 및 통신 모듈 개발
  - 목표 규모인 Ret≃2,000, 5×10^{9}개의 격자보다 더 큰 대규모 시뮬레이션을 위해 동적변수 할당과 메모리 교차할당을 통한 메모리 최적화 수행

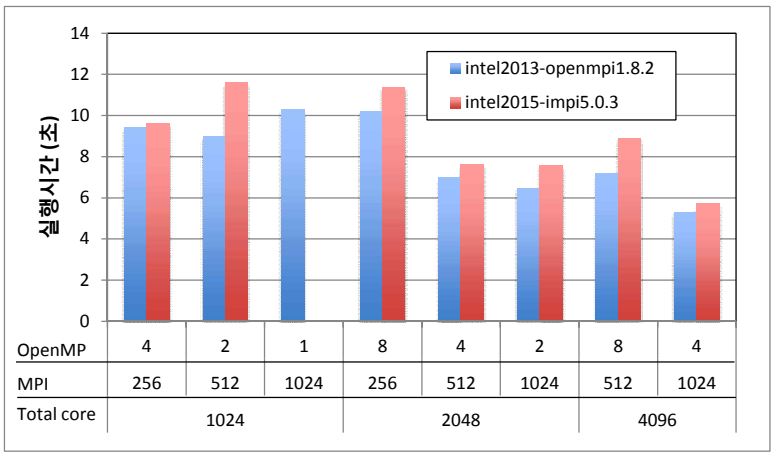


<그림> 유도데이터 이용 통신최적화
(19% 성능 향상)



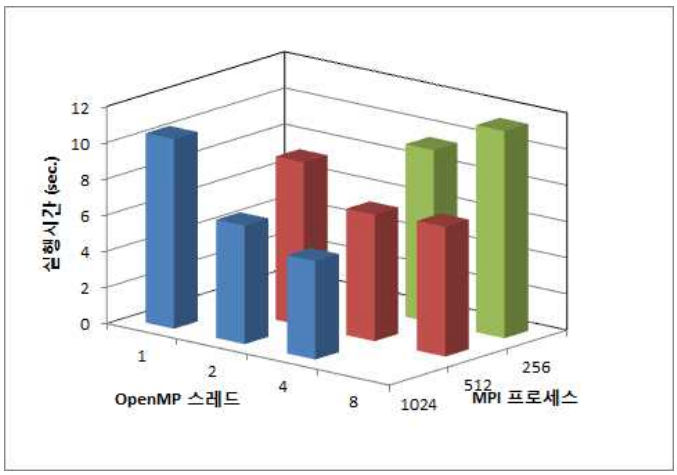
<그림> 메모리 최적화 결과
(평균 10% 이하로 절감)

- 결과
  - 5×10^9개의 격자를 갖는 목표 문제에 대해 Hybrid 병렬화를 통한 4,096코어까지 Strong-scalability의 병렬확장성을 보임(1,024코어 대비 4,096코어에서 1.93배성능 향상)



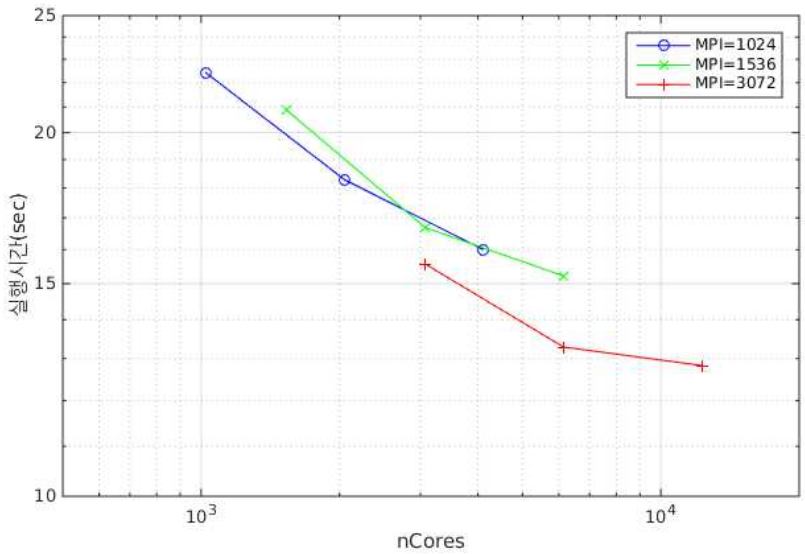
<그림>코어수별 성능테스트 (격자수 = 5x109)

  - 5×10^9개의 격자를 갖는 목표 문제에 대해 메모리 소모량을 기존 12 TByte에서 2.2TByte로 81% 절감함에 따라 MPI 프로세스 수에 제한 없이 다양한 조건으로 실행 가능하며 MPI 수에 따라 OpenMP 스레드를 2개 또는 4개로 했을 때 최고의 성능을 보임



<그림> MPI-OpenMP 조합별 성능테스트

- 메모리 절감에 따르는 문제 규모 확대 (4배 이상) 가능 - 목표 문제 대비 3배 큰 1.5×10^{10}개의 격자를 갖는 문제에 대해 테스트하여 6,144코어까지 strong-scalability의 병렬확장성을 보임



<그림> 코어수별 성능테스트 (격자수 = 1.5x1010)

7. 채널코드 후처리 병렬화

- 코드 개요
  - 2013년에 개발한 채널 난류코드에 대한 후처리가 병렬화 필요
  - Serial 형태로의 Fortran 90 코드로 구현
- 개발 내용
  - 순차 코드에서는 메모리 부족으로 x와 z방향으로 2차원 영역 분할로 병렬 설계
  - 각 단계 모두 x와 z방향으로 2D FFT수행이 필요하므로, 나눠진 영역에 대해서 각각 전치 통신(MPI_Alltoall)을 한 뒤 해당 축에 대해서 1D FFT 수행
  - 메모리 최적화를 위해서 전치시 메모리 구조를 MPI DDT(Derived Data Type)을 통해서 최적화
  - 도메인 nx=192,ny=129,nz=160에 대해서 테스트 후 nx=1536,ny=257,nz=1536 규모까지 확장해서 2048코어까지 테스트
- 결과
  - 순차 코드에서는 메모리 부족으로 수행할 수 없었지만 병렬화를 통해서 2048코어에서 수행 성공
  - MPI DDT 최적화로 병렬화 대비 약 10% 성능향상

8. 다체계에서 핵의성질 병렬화

· 코드 지원 개요

- 핵물리 코드로서 기존에 OpenMP 병렬화만 되어 있는 코드를 MPI 병렬화로 요청

· 수행방법

- 테스트 시스템 : 타키온2
- 컴파일러 및 라이브러리

* GNU compiler 4.9.3
* mvapich2-2.1
* lapack-3.5.0

- 병렬화 (MPI) 방법

* Read input data

계산 관련 설정 값을 파일로부터 읽어 오는 부분은 0번 프로세스에서 수행하고 MPI_BCAST 함수를 사용하여 다른 프로세스로 전달

* Loop 병렬화

· 서브루틴 GMAT과 DIAG1에 있는 이중 loop 문이 메인 계산이고, 병렬화가 필요함

· 각 인덱스 별 계산은 독립적으로 수행이 가능하지만, 대부분 전체 행렬의 정보를 가지고 있어야 하므로 모든 프로세스가 동일한 데이터를 가지고 있어야 함

· 따라서 인덱스를 나누어 프로세스 별로 수행하고 MPI_ALLREDUCE를 이용하여 취합이 가능함

· 인덱스를 프로세스가 나누어 가지는 방식은 라운드 로빈 방식을 택하는 것이 성능이 가장 좋음 (행렬이 대칭이므로 대각 위쪽 성분만 계산함)

* 메모리 이슈

· 노드 당 약 12GB 정도의 메모리를 사용하고 있는데, MPI_ALLREDUCE 루틴으로 1170*1170개의 double precision 데이터를 전송할 경우 메모리 에러 발생.

· MPI_ALLREDUCE 대신 MPI_SEND, MPI_RECV, MPI_BCAST로 변경하여 에러 회피함

* Write output data

· 파일에 결과를 출력하는 부분은 0번 프로세스가 취합하여 하도록 작성

· 결과

- 병렬 성능

* 64 core 사용 시 30분 소요 (icaldmat 값을 1로 했을 때 순차 실행의 경우 2-3일 소요)
* 순차 실행 시간이 정확 치 않아 성능 향상을 수치로 나타내기 어려움
* 병렬화 한 GMAT 함수의 경우 코어 수(<64)에 따라 거의 linear 하게 성능이 증가
* 병렬 성능 및 확장성 확인을 위해 추가 테스트가 필요

- 문제점

* Common block 변수로 대부분의 data를 저장하고 있기 때문에 stack memory를 과도하게 사용
* if, goto 문이 많아서 계산 도중에 특정 조건에 의해 루프를 벗어나 MPI 오류가 생길 수 있음

9. WRF OPENACC 최적화

· 코드 지원 개요

- WRF(Weather Research and Forecasting)는 대기과학 연구와 현업 예보를 위하여 만들어진 중규모 수치 예보이며 미국 기상연구소(NCAR: National Center for Atmospheric Research)에서 개발한 오픈소스 프로그램
- 기존에 연구자들과 현업기관에서 널리 쓰이던 MM5모델을 대체하기 위한 차세대 수치모델로 개발되었으며, 지구를 3차원의 격차체계로 나누어 대기의 상태를 물리법칙에 따라 수식화한 모델
- 역학코어와 여러 가지 물리과정을 포함하며 KISTI 4호기 슈퍼컴퓨터뿐만 아니라 전세계적으로 빈번하게 사용됨
- WRF는 공식적으로 130여개의 나라에서 25,000여명이 이용하고 있고, 매년 workshop과 tutorial이 열리고 있음. 다음 웹사이트(wrf-model.org)에서 WRF에 대한 자세한 정보를 얻을 수 있음
- CPU에서도 MPI 병렬화를 통해서 고해상도에 대해서 수행이 가능하지만, 계산시간이 많이 소요되는 물리과정에 대해서 계산가속기로 이식하여 전체 실행성능을 향상시킬 수 있음
- NVIDIA GPU에서 물리과정 수행을 위해 CUDA와 OpenACC 언어 기반으로 비공개로 개발중

- 기상 모델인 OpenACC 기반 WRF에 대해서 최적화 요청

• 수행 내용

- NVIDIA GPU 벤치마크
  * OpenACC 언어로 구현된 개발용 버전을 테스트 수행 (WRF 3.6)
  * CONUS 18km(80x80 격자)를 다음 역학 및 물리 설정으로 실험

NVIDIA GPU 테스트를 위한 역학 설정

역학 (Dynamics) Scheme
Fully compressible and nonhydrostatic model
Terrain-following hydrostatic pressure
Arakawa C-grid staggering
Runge-Kutta 2nd and 3rd order time integeration scheme
2nd and 6th order advection scheme

NVIDIA GPU 테스트를 위한 물리 설정

물리(Physics) Scheme	NVIDIA GPU
Microphysics	Thompson
Cumulus Parameterization	Kain-Fritsch
Longwave Radiation	RRTMG
Shortwave Radiation	Goddard
Surface Layer	MM5 similarity
Land surface	Noah-MP
PBL	YSU

· NVIDIA GPU 물리과정

- 공기 중에 존재하는 수상체들을 표현하는 microphysics으로는 Thompson
- convection 과정을 표현하는 cumulus으로는 Kain-Fritsch
- Longwave 및 shortwave Radiation을 표현하는 RRTMG와 Goddard
- 대기와 지표간의 상호작용을 표현하는 Surface Layer와 Land surface으로는 MM5 similarity와 Noah-MP
- 지표면 경계층 모수화로 YSU를 사용

• 결과

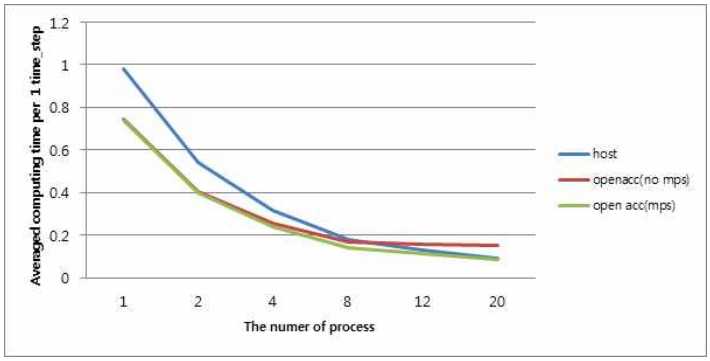
- NVIDIA GPU 테스트

* 단일노드에서 CPU(2cores)대비 GPU(MPS 동작, CPU(2cores)+1GPU) 사용시 성능 1.36배 향상

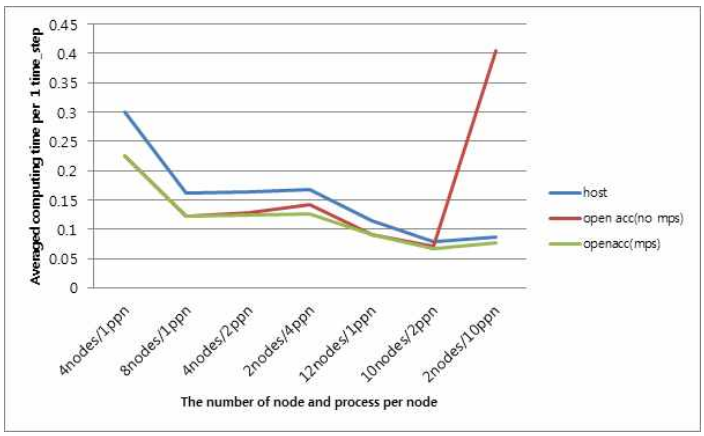
- MPS를 작동시킬 경우에 코어수가 4개 이상인 경우에도 병렬확장성 보장이 됨

* 멀티노드에서 CPU(4nodes/1ppn)대비 GPU(MPS 동작, CPU(4nodes/1ppn)+4GPUs) 사용시 성능 1.34배 향상

- 8nodes/1ppn, 4nodes/2ppn, 2nodes/4ppn는 모두 8개의 프로세스를 MPI로 동작한 경우인데, MPS를 동작시키지 않으면 4ppn인 경우에 성능저하가 목격됨 -> 하지만 MPS를 동작시키면 모두 비슷한 성능이 보장됨을 알 수 있음
- 10nodes/2ppn, 2nodes/10ppn는 모두 20개의 프로세스를 MPI로 동작한 경우인데, MPS를 동작시키지 않으면 10ppn인 경우에 성능저하가 현저하게 목격됨 -> 하지만 MPS를 동작시키면 모두 비슷한 성능이 보장됨을 알 수 있음



단일노드에서 WRF의 GPU 성능



멀티노드에서 WRF의 GPU 성능

10. NCEP CFS 기후모델 Tachyon2 최적화

· 코드 지원 개요

- 기후 모델로서 GAIA시스템에서 사용하던 코드를 Tachyon2로 이식 및 최적화 요청

· 수행방법

- 사용환경: pgi-2014 + mvapich2-2.1
- cfs/sorc 아래 29개 소스 디렉터리 중 cfs_coupa2o.fd.org와 cfs_global_atmos.fd.old 제외한 27개 디렉터리 소스 빌드 시도
- 위 디렉터리 소스 빌드를 위해 필요한 라이브러리 중 위 소스 디렉터리에 없는 ip, bufr, sigio 라이브러리 소스를 nwprod/lib/sorc에서 찾아 추가 빌드 하였음(3 of 58)

* w3 불완전 빌드 (AIX: sys/proc.h)

· cfs_global_atmos.fd.e130skh 빌드 실패 (ESSL + w3)

· global_postgp.fd 빌드 실패(ESSL + w3 + xlf 고유함수)

· cfs_avg_grib.fd 빌드 실패(ESSL)

· cfs_global_atmos.fd 빌드 실패(ESSL + w3)

* w3 라이브러리

· 리눅스에 sys/proc.h가 없어 start, summary 함수 빌드 실패

* Xlf 고유함수 (libsp_4에서 호출하는 num_parthds 함수 링크 오류)

· OpenMP 함수로 대체 가능

* ESSL 함수

· drcft: FFT real to complex

· dcrft: FFT complex to real

· scrft: FFT complex to real

· srcft: FFT real to complex

· sbsrch: Binary search for elements

· sgetmo: general matrix transpose

· dgef: general matrix factorization

· dges: general matrix, conjugate transpose solve

- 환경설정

* module add compiler/pgi-2014 mpi/mvapich2-2.1 applic/hdf5 applic/grads-2.0.a9 applic/ncarg-6.0.0 applic/netcdf4-4.1.3 applic/lapack-3.5.0
* NETCDF 경로: /applic/compilers/pgi/linux86-64/2014/applib1/NETCDF4/4.1.3/lib

- 기준 경로: /scratch/suncode2/CFS/

- 기준 경로 아래 cfs/sorc 29 dirs & nwprod/lib/sorc 58 dirs

* cfs/sorc 아래 27개 디렉터리 컴파일 시도, 이 과정에서 필요한 경우 nwprod/lib/sorc의 소스 컴파일 시도

1 cfs/sorc 컴파일

1.1 cfs/sorc/bacio

- makebacio.sh 수정
- clib.h, clib4.h, clib8.h 수정 (#define LINUX)
- lib/ 아래 libbacio_4.a, libbacio_8.a 생성

1.2 cfs/sorc/w3

- makelibw3.sh 수정
- mova2i.c 수정, 아래 코드 추가

```
#ifdef PGI
   #define mova2i mova2i_
#endif
```

- errexit.f (EXIT_ à EXIT)
- lib/incmod/w3_4, w3_8, w3_d 디렉터리 생성: module 파일 이동
- lib/아래 libw3_4.a, libw3_8.a 생성,
- libw3_d.a 생성 과정 오류(linux에 proc.h 없음)

```
pgcc -c  -O3 summary.c
PGC-F-0206-Can't find include file sys/proc.h (summary.c: 27)
PGC/x86-64 Linux 9.0-4: compilation aborted
make: *** [../../../lib/libw3_d.a(summary.o)] Error 2
```

(W3TAGB(w3tagb.f)에서 start, summary 함수 호출)

1.3 cfs/sorc/sp

- makelibsp.sh 수정
- lib/아래 libsp_4.a, libsp_8.a, libsp_d.a 생성

1.4 cfs/sorc/cfs_global_atmos.fd.e130skh/

- Makefile 수정
- gloopb_omp.f 수정(179 line comment out, random_seed(generator=2))
- rascnvv2.f 수정(596 line comment out, random_seed(generator=2))
- wrtsfc_comm.f(1066,1759 라인 implicit none 위치 변경)
- 링크 오류 (ESSL + W3TAGB)

```
grid2fln_b.o: In function `grid2four_':
/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd.e130skh/./grid2fln_b.f:41:
undefined reference to `drcft_'
```

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd.e130skh/./grid2fln_b.f:41: undefined reference to `drcft_'

grid2fln_b.o: In function `grid2fouropt_':

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd.e130skh/./grid2fln_b.f:91: undefined reference to `drcft_'

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd.e130skh/./grid2fln_b.f:91: undefined reference to `drcft_'

fln2grid_b.o: In function `four2grid_':

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd.e130skh/./fln2grid_b.f:226: undefined reference to `dcrft_'

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd.e130skh/./fln2grid_b.f:226: undefined reference to `dcrft_'

fln2grid_b.o: In function `four2gridopt_':

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd.e130skh/./fln2grid_b.f:280: undefined reference to `dcrft_'

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd.e130skh/./fln2grid_b.f:280: undefined reference to `dcrft_'

/scratch/suncode2/CFS/lib/libw3_d.a(w3tagb.o): In function `w3tagb_':

/scratch/suncode2/CFS/cfs/sorc/w3/./w3tagb.f:79: undefined reference to `start_'

/scratch/suncode2/CFS/cfs/sorc/w3/./w3tagb.f:101: undefined reference to `summary_'

1.5 cfs/sorc/cfs_cnvsig.fd/

- makefile 수정
- cfs_convsigexec 생성

1.6 cfs/sorc/cfs_mom3_etadaily2.5.fd/

- makefile 수정
- cfs_mom3_etadaily2.5 생성

1.7 cfs/sorc/cfs_mom3_daily_proc.fd/

- makefile 수정
- cfs_mom3_daily_proc 생성

1.8 cfs/sorc/cfs_grib_eta_daily.fd/

- libip_4.a, libbufr_4_32.a 필요 (nwprod/lib/sorc에 위치)
- cfs_grib_eta_daily 생성

1.9 cfs/sorc/cfs_global_cycle_forR2.fd/

- makefile 수정
- cfs_global_cycle_R2 생성

1.10 cfs/sorc/global_sighdr.fd/

- libsigio_4.a 필요 (nwprod/lib/sorc에 위치)
- makefile 수정

- global_sighdr 생성

1.11 cfs/sorc/global_sigavg.fd/

- makefile 수정
- global_sigavg 생성

1.12 cfs/sorc/global_postgp.fd/

- makefile 수정
- mptran.f (609, 828 라인 수정)

MPI_LOGICAL1 à MPI_LOGICAL

- postgp.f 수정 (sigio_r_module의 private 선언으로 sigio_module의 선언 변수 처리 안됨)

PGF90-S-0087-Non-constant expression where constant expression required (postgp.f: 200)

use sigio_module 추가(174,1424 라인)해 해결

- Link 오류 (ESSL + w3tagb + XLF Intrinsic(num_parthds))

/scratch/suncode2/CFS/cfs/sorc/global_postgp.fd/./postgp.f:4292: undefined reference to `sbsrch_'

/scratch/suncode2/CFS/cfs/sorc/global_postgp.fd/./postgp.f:4305: undefined reference to `sbsrch_'

mptran.o: In function `mptran1r4_':

/scratch/suncode2/CFS/cfs/sorc/global_postgp.fd/./mptran.f:715: undefined reference to `sgetmo_'

/scratch/suncode2/CFS/cfs/sorc/global_postgp.fd/./mptran.f:734: undefined reference to `sgetmo_'

/scratch/suncode2/CFS/lib/libw3_4.a(w3tagb.o): In function `w3tagb_':

/scratch/suncode2/CFS/cfs/sorc/w3/./w3tagb.f:79: undefined reference to `start_'

/scratch/suncode2/CFS/cfs/sorc/w3/./w3tagb.f:105: undefined reference to `summary_'

/scratch/suncode2/CFS/lib/libsp_4.a(splat.o): In function `splat_':

/scratch/suncode2/CFS/cfs/sorc/sp/./splat.f:137: undefined reference to `dgef_'

/scratch/suncode2/CFS/cfs/sorc/sp/./splat.f:137: undefined reference to `dges_'

/scratch/suncode2/CFS/cfs/sorc/sp/./splat.f:171: undefined reference to `dgef_'

/scratch/suncode2/CFS/cfs/sorc/sp/./splat.f:171: undefined reference to `dges_'

/scratch/suncode2/CFS/lib/libsp_4.a(ncpus.o): In function `ncpus_':

/scratch/suncode2/CFS/cfs/sorc/sp/./ncpus.f:28: undefined reference to `num_parthds_'

/scratch/suncode2/CFS/lib/libsp_4.a(spffte.o): In function `spffte_':

/scratch/suncode2/CFS/cfs/sorc/sp/./spffte.f:83: undefined reference to `scrft_'

/scratch/suncode2/CFS/cfs/sorc/sp/./spffte.f:105: undefined reference to `srcft_'

/scratch/suncode2/CFS/cfs/sorc/sp/./spffte.f:114: undefined reference to `scrft_'

/scratch/suncode2/CFS/cfs/sorc/sp/./spffte.f:139: undefined reference to `srcft_'

1.13 cfs/sorc/cfs_pseudo.fd/

- makefile 수정
- cfs_pseudo 생성

1.14 cfs/sorc/cfs_ocean_mom3_1x1.fd/

- makefile 수정
- libpatch.f 컴파일 제외
- cfs_ocean_mom3_1x1 생성

1.15 cfs/sorc/cfs_pseudo.fd.han/

- makefile 수정
- cfs_pseudo 생성

1.16 cfs/sorc/cfs_coupa2o.fd.org/

1.17 cfs/sorc/cfs_global_atmos.fd.old/

1.18 cfs/sorc/cfs_coupa2o.fd/

- makefile, makefileA2O 수정
- cfs_coupa2o, coupA2O_fix.x 생성

1.19 cfs/sorc/cfs_grib_ocean.fd/

- makefile 수정
- cfs_grib_ocean 생성

1.20 cfs/sorc/cfs_getfac.fd/

- makefile 수정
- cfs_getfac 생성

1.21 cfs/sorc/cfs_force_grib_date_y2k.fd/

- makefile 수정
- cfs_force_grib_date_y2k 생성

1.22 cfs/sorc/cfs_avg_grib.fd/

- makefile 수정
- LINK 오류(ESSL)

/scratch/suncode2/CFS/lib/libsp_4.a(splat.o): In function `splat_':

/scratch/suncode2/CFS/cfs/sorc/sp/./splat.f:137: undefined reference to `dgef_'

/scratch/suncode2/CFS/cfs/sorc/sp/./splat.f:137: undefined reference to `dges_'

/scratch/suncode2/CFS/cfs/sorc/sp/./splat.f:171: undefined reference to `dgef_'

/scratch/suncode2/CFS/cfs/sorc/sp/./splat.f:171: undefined reference to `dges_'

1.23 cfs/sorc/cfs_mppnccombine.fd/

- makefile 수정

- cfs_mppnccombine 생성

1.24 cfs/sorc/nhour.fd/

- makefile 수정
- nhour 생성

1.25 cfs/sorc/ndate.fd/

- makefile 수정
- ndate 생성

1.26 cfs/sorc/cfs_convert_restart.fd/

- makefile 수정
- cfs_convert_restart 생성

1.27 cfs/sorc/cfs_skim_sst.fd/

- makefile 수정
- cfs_skim_sst 생성

1.28 cfs/sorc/cfs_coupo2a.fd/

- makefile 수정
- coupO2A_fix.x 생성

1.29 cfs/sorc/cfs_global_atmos.fd/

- Makefile 수정
- LINK 오류(ESSL+w3tagb)

grid2fln_b.o: In function `grid2four_':

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd/./grid2fln_b.f:41: undefined reference to `drcft_'

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd/./grid2fln_b.f:41: undefined reference to `drcft_'

grid2fln_b.o: In function `grid2fouropt_':

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd/./grid2fln_b.f:91: undefined reference to `drcft_'

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd/./grid2fln_b.f:91: undefined reference to `drcft_'

fln2grid_b.o: In function `four2grid_':

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd/./fln2grid_b.f:226: undefined reference to `dcrft_'

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd/./fln2grid_b.f:226: undefined reference to `dcrft_'

fln2grid_b.o: In function `four2gridopt_':

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd/./fln2grid_b.f:280: undefined reference to `dcrft_'

/scratch/suncode2/CFS/cfs/sorc/cfs_global_atmos.fd/./fln2grid_b.f:280: undefined reference to `dcrft_'

/scratch/suncode2/CFS/lib/libw3_d.a(w3tagb.o): In function `w3tagb_':

/scratch/suncode2/CFS/cfs/sorc/w3/./w3tagb.f:79: undefined reference to `start_'

/scratch/suncode2/CFS/cfs/sorc/w3/./w3tagb.f:105: undefined reference to `summary_'

2 nwprod/lib/sorc 컴파일

2.1 nwprod/lib/sorc/ip/

- makelibip.sh 수정

- lib/ 아래 libip_4.a, libip_8.a, libip_d.a 생성

2.2 nwprod/lib/sorc/bufr (32bit 생성 안함, tachyon2에 32bit 시스템 라이브러리 없음)

- makebufrlib.sh 수정(preprocessing 수정)

- lib/ 아래 libbufr_4_64.a, libbufr_8_64.a, libbufr_d_64.a, libbufr_4_32.a(64bit), libbufr_s_64.a 생성

2.3 nwprod/lib/sorc/sigio

- makefile_4 수정

- lib/libsigio_4.a 생성

- lib/incmod/sigio_4/ sigio_module.mod, sigio_r_module.mod 생성