[2025-6]

KEASE 기술서

한국형 엑사스케일 응용 SW 개발 환경 프레임워크

메모리/공유자원 접근 최적화 및
응용 SW 워크로드 스케줄링 최적화 기술

2025. 3. 10

성 균 관 대 학 교

[개 정 이 력]

개정 일자	버전	설명	개정페이지	작성자	확인자
2025.3.10	1.0	적응형 OpenMP 글로벌 스케줄러 개발	전체	이현수	서의성

목 차

1. 개요 ····· 1

2. 적응형 OpenMP 글로벌 스케줄러 ····· 1

3. 실험 및 분석 ····· 4

4 결론 및 기대효과 ····· 5

1. 개요

본 프로젝트는 Kokkos 기반 환경에서 다수의 HPC(고성능 컴퓨팅) 응용이 동시에 실행될 때 CPU 활용 및 성능 특성을 극대화하기 위한 최적화 스케줄링 기법을 개발하는 것을 목표로 한다. 이를 위해, Kokkos에서 CPU 기반 병렬 프로그래밍을 위해 사용하는 OpenMP 라이브러리를 확장하고, 해당 라이브러리에 글로벌 스케줄링 모듈(Adaptive Global Scheduler) 및 적응형(Adaptive) OpenMP를 통합하여, 여러 응용 간의 CPU 자원 사용을 효율적으로 제어하는 방법을 제시한다.

2. 적응형 OpenMP 글로벌 스케줄러

1) OpenMP 라이브러리

Kokkos는 하드웨어에 상관없이 동일한 소스 코드를 작성하면 이를 다양한 이기종 연산 장치(CPU, GPU 등)에 맞추어 컴파일해 주는 병렬 프로그래밍 프레임워크이다. CPU 기반 프로그램을 생성할 때는 내부적으로 OpenMP 라이브러리를 사용하며, 결과적으로 컴파일된 실행 파일은 OpenMP의 스레딩 및 스케줄링 정책에 의해 동작한다.

OpenMP의 대표적인 병렬화 방식은 #pragma omp parallel for와 같은 어노테이션을 사용해 for 반복문을 병렬 처리하는 것이다. 일반적인 GCC 컴파일러 환경을 가정하면, 다음과 같은 과정을 거쳐 병렬 코드가 생성된다.

[소스 변환]

컴파일러는 #pragma omp parallel for와 같은 어노테이션을 인지하고, 반복문 블록을 OpenMP 진입 함수인 GOMP_parallel()로 변경한다. 원래의 for 반복문 블록은 Compute Function(fn(...)) 형태로 묶여 함수 포인터로 변환되고, 이 함수 포인터가 GOMP_parallel()로 전달된다.

[OpenMP 스레드 구조 생성]

GOMP_parallel() 함수가 호출되면, 병렬화 수행을 위한 Thread Pool, Team, 그리고 Pthread들이 초기화된다. 여기서 Thread Pool은 스레드 확장 및 축소 시 메타데이터 생성 시간을 단축하기 위해 사전에 여러 스레드를 준비하고 관리하기 위한 구조다. 그리고 Team의 경우 하나의 병렬 영역을 담당하는 스레드를 관리하기 위한 논리적 구조체이며, 실제 계산은 여러 개의 Pthread를 통해 이루어진다.

[스레드의 무한 반복 루프 구성]

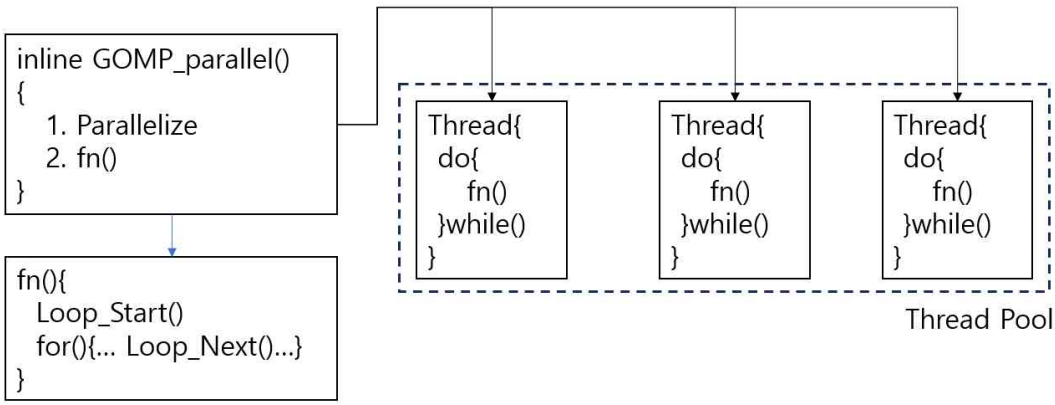
생성된 Pthread들은 한 번의 병렬 처리를 위해 생겼다가 사라지는 것이 아니라, 반복해서 사용할 수 있도록 무한 루프 형태로 동작한다.

이때 fn() 포인터는 OpenMP의 핵심 함수인 Loop_Start(), Loop_Next() 등을 활용하여 반복문 작업을 스레드별로 분할하여 처리하도록 설계된다.

[Work Share와 Chunk 구조]

단일 for 반복문을 여러 스레드에 분산하기 위해 Work Share와 Chunk 개념을 사용한다.

작업량을 나누는 스케줄링 정책에는 Static, Dynamic, Guided 등이 있으며, 특히 Dynamic은 스레드가 작업을 끝낼 때마다 새로운 Chunk를 할당받아 CPU 유휴를 최소화하고, Guided는 Chunk 크기를 점진적으로 줄여 계산량 불균형을 완화한다.



[그림 1] OpenMP 스레딩 구조

이와 같은 OpenMP 구조 덕분에 개발자는 간단한 어노테이션만으로도 CPU 병렬화를 구현할 수 있다. 다만, 이는 단일 응용 내에서만 동작 최적화가 이루어지며, 여러 응용이 동시에 실행될 때는 자원 간섭 문제가 발생할 수 있다.

OpenMP 라이브러리의 한계점은 코어 수 확장 시 성능 포화와 복수 응용 실행 시 자원 경쟁으로 볼 수 있다. CPU 코어와 스레드를 과도하게 늘리면 메모리 접근 병목이나 캐시 결함(Cache miss) 증가 등으로 인해 성능 향상이 제한될 수 있다. 예를 들어, Sparse Matrix 연산 등은 데이터 접근 패턴의 불규칙성으로 인해 스레드가 많아도 성능이 크게 향상되지 않는 케이스가 존재한다.

또한 OpenMP는 자기가 동작하는 단일 응용 내에서 최적화에 집중할 뿐, 여러 응용을 동시에 고려한 자원 분할이나 스케줄링을 지원하지 않는다. 따라서 복수의 OpenMP 응용이 같은 CPU를 공유하면, 각 응용이 자체적으로 최대 스레드를 생성하여 CPU를 독점하려 하고, 이로 인해 성능 간섭과 자원 중첩 사용이 심화될 수 있다.

특히 최근 HPC 환경에서는 CPU와 GPU가 혼합된 시스템 구성이 일반적이어서, CPU 유휴 자원을 여러 응용이 효율적으로 활용하는 전략이 중요하다. 그러나 기존의 OpenMP만으로는 이러한 통합 자원 활용 시나리오를 제대로 지원하기 어렵다.

2) HPC 응용의 메모리·공유자원 접근 성능 개선 스케줄링 기법

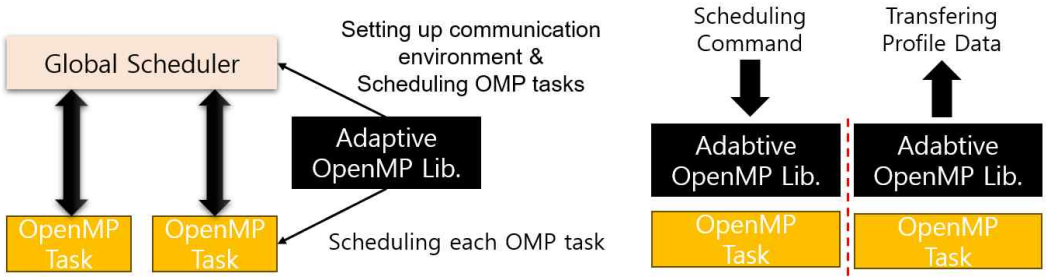
본 프로젝트는 위에서 언급한 OpenMP의 한계를 극복하기 위해, 글로벌 스케줄링 모듈(Global Scheduler)과 적응형 OpenMP(Adaptive OpenMP)를 개발하여 복수의 HPC 응용에 대해 효율적으로 작업을 배치하고 CPU 자원을 동적으로 제어하는 기술을 제시한다.

[OpenMP 기반 글로벌 스케줄링 모듈]

시스템에서 독립적으로 실행되는 모든 OpenMP 응용을 하나의 모듈에서 통합 제어하고, 각 응용의 프로파일링 정보를 기반으로 자원 배분 및 스레드 수를 결정한다. 또한 응용 간 CPU 코어 분할, 스레드 배치, 메모리 대역폭 사용량 등을 종합적으로 조정해 자원 간섭을 최소화하는 역할을 수행한다.

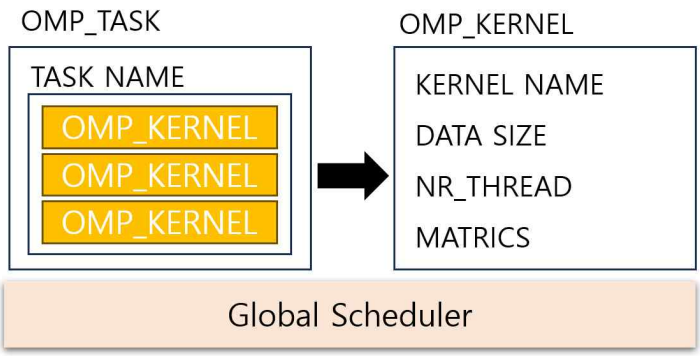
적응형 OpenMP 라이브러리를 사용하는 응용은 시작 시 글로벌 스케줄러와 통신 채널을 마련한다. 각 응용은 프로파일링 데이터(스레드 개수, 데이터 사이즈, cache miss rate, IPC, 메모리 대역폭

사용량, Throughput, FLOPS 등)를 글로벌 스케줄러로 전송한다. 글로벌 스케줄러는 이 정보를 취합·분석하여 선형 회귀 기반의 성능 모델을 만든 뒤, 최적 스레드 개수나 코어 배치를 결정해 응용에 다시 전달한다.



[그림 2] 적응형 OpenMP 라이브러리를 통한 글로벌 스케줄러와 HPC 응용간 통신 구조

글로벌 스케줄링 모듈은 응용별로 OMP_TASK 구조를 할당하여, 해당 응용의 병렬 커널(연산 함수)을 관리·추적한다. 각 커널의 프로파일링 데이터(예: Dense, Sparse, FFT, Graph, Memory Load 연산 등)를 종합적으로 기록해 실시간 혹은 주기적으로 스케줄링 결정을 내린다.



[그림 3] 글로벌 스케줄링 모듈의 응용 및 커널 관리 구조

스레드 개수 증가에 따른 Throughput(초당 반복 수) 변화와 FLOPS(초당 부동소수점 연산 수)의 상관관계를 통해 각 응용에 대해 최적 스레드 수를 찾는다. FLOPS가 지나치게 떨어지지 않으면서도 응용 자체의 처리 속도(Throughput)를 최대화할 수 있는 지점을 추정한다. 메모리 접근 병목이나 cache miss 비율도 종합 고려하여, 불필요하게 많은 스레드를 할당하지 않도록 제어한다.

3) 적응형 OpenMP(Adaptive OpenMP)

기존 OpenMP 라이브러리와 동일한 사용자 프로그래밍 모델을 제공하되, 내부적으로 글로벌 스케줄러와 연동해 스레드 배치·확장 전략을 동적으로 조정할 수 있도록 한다. 복수의 응용을 동시에 지원하기 위해, 모든 코어에 대한 스레드를 초기화하고, 필요 시 스레드를 Active/Sleep 상태로 전환함으로써 빠른 재활용과 오버헤드 축소를 달성한다.

적응형 OpenMP는 Performance Counter Lib.(Perf)와 Intel의 Performance Counter Monitor(PMC) 등을 활용해 각 Chunk 단위로 캐시 미스율, IPC, 메모리 대역폭 사용량 등을 측정한다. 주기적으로 글로벌 스케줄러에 성능 데이터를 전달해, 스케줄러가 최적화 결정을 내릴 수 있도록 한다.

글로벌 스케줄러로부터 “코어 할당 변경” 지시가 오면, 해당 응용은 일부 스레드를 Sleep 모드로

전환하거나 Wake-up하여 스레드 수를 조정한다. Dynamic, Guided 등의 기존 OpenMP 스케줄링 정책과 연동하되, 상위 레벨에서 배정된 CPU 자원 범위를 벗어나지 않도록 Adaptive OpenMP가 재조정한다. 이러한 구조를 통해, 복수 응용이 서로 간섭 없이 CPU 자원을 공유하고, 각 응용은 자체 특성(메모리 집약도, 계산 집약도, 데이터 접근 패턴 등)에 맞추어 성능을 극대화할 수 있다.

4) CPU 기반 HPC 응용 최적화 스케줄링과 Kokkos 통합

Kokkos는 CPU용 응용을 생성할 때 내부적으로 OpenMP를 사용한다. 따라서 본 프로젝트에서 확장한 적응형 OpenMP 라이브러리를 사용하면, 별도의 Kokkos 수정 없이 Kokkos 응용 전체에 동일한 스케줄링 기능을 적용할 수 있다. 예컨대 Kokkos로 작성된 Dense/Sparse/Memory Load 중심의 HPC 응용이라도, 컴파일 시점에 Adaptive OpenMP가 링크되어 실행 단계에서 글로벌 스케줄러와 통신하게 된다. 본 기술은 CPU 코어가 많은 매니코어(Many-Core) 환경에서 특히 유용하며, CPU-GPU 혼합 환경에서도 CPU 측 병렬 자원 활용도를 높이는 데 기여한다. 향후에는 스케줄러가 GPU 워크로드 역시 통합 관리하도록 확장할 수 있으며, CUDA 등 GPU 측 병렬 라이브러리와의 연동도 고려할 수 있다.

3. 실험 및 분석

1) 실험 환경

CPU	Knights Landing (KNL): 2nd Generation Intel® Xeon Phi™ Processor(64core-256T)
Memory	DDR4 96GB
Workload	Dense Matrix, Spars Matrix, Parallel Memory Load base on OpenMP(Kokkos)

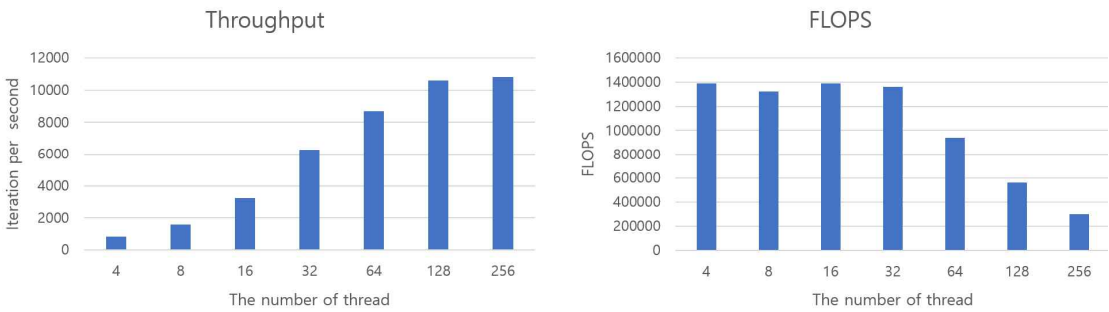
1) 주요 HPC 커널 특성

Dense Matrix 연산: N×N 행렬의 곱을 예로 들면, 계산량뿐만 아니라 메모리 접근량도 많아 스레드 수가 늘면 Throughput은 올라가지만, 메모리 대역폭 병목으로 FLOPS가 떨어질 수 있다.

Sparse Matrix 연산: 메모리 접근량 자체는 작을 수 있으나, 불규칙한 인덱싱과 데이터 구조로 인해 병렬화 효율이 낮아지기 쉽다.

Memory Load 커널: 실질적 산술 연산보다는 대량의 메모리 로드가 주가 되므로 메모리 병목이 발생하기 쉽다.

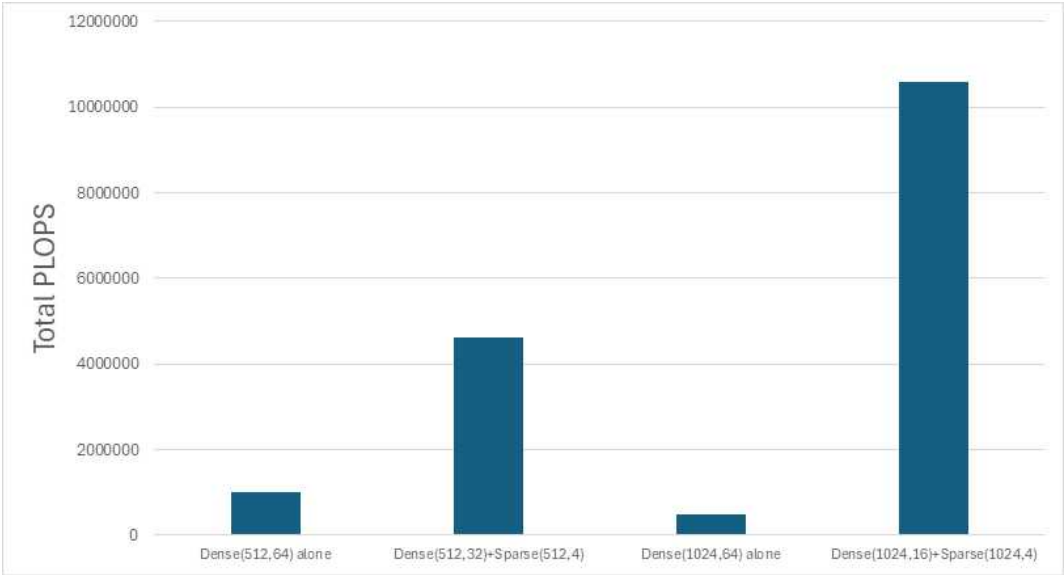
2) 스레드 증가에 따른 성능 변화



[그림 4] Dense Matrix 연산의 스레드 개수에 따른 Throughput과 FLOPS

일정 스레드 수까지는 Throughput이 빠르게 향상되지만, 그 이상은 메모리 병목 혹은 캐시 충돌 등으로 성능이 포화된다. FLOPS는 스레드 확장과 함께 줄어드는 경향이 있을 수 있는데, 이는 캐시 미스 증가, 메모리 접근 Stall 등으로 실제 유효 계산이 감소하기 때문이다.

각 커널이 가진 성능 특성을 학습한 뒤, 전체 시스템 차원에서 CPU 코어와 스레드를 분배한다. Throughput이 더 이상 크게 향상되지 않으면서 FLOPS 손실이 커지는 시점을 찾아, 해당 응용에 불필요하게 할당된 코어를 다른 응용으로 재할당하여 시스템 전체 처리량을 극대화한다.



[그림 5] 동시 배치 실험 결과

그래프에서 각각 단일 워크로드로 코어를 늘려서 사용하였을 경우와, 여러 워크로드의 최적 스레드 조합을 통해 얻어진 FLOPS 합산을 보여준다. 그래프에서 보이는 바와 같이 개별 응용이 스레드 개수를 늘려 많은 코어수를 점유하는 경우와 여러 응용이 동시에 많은 수의 코어를 점유하여 동작하는 것이 전체 시스템 관점에서 그래프상 4.6배에서 21.1배까지 더 이득임을 알 수 있다. 이와 같이 동시에 배치 한다 하더라도, 개별 응용의 성능 간섭으로 인한 성능 저하는 10~30% 수준으로 나타났다.

4. 결론 및 기대 효과

본 과제에서 제안하는 글로벌 스케줄링 모듈 + 적응형 OpenMP 구조는 기존 OpenMP의 단일 응용 최적화 한계를 뛰어넘어, 복수의 HPC 응용이 동시에 실행되는 환경에서 CPU 자원을 효율적으로 관리하는 기술을 제공한다.

메모리 대역폭 및 캐시 사용을 고려하여 스레드 수를 동적으로 조정함으로써, 활용도가 낮은 유휴 코어를 타 응용에 재할당해 전체 시스템 처리량을 높일 수 있다. 글로벌 스케줄러가 각 응용의 프로파일링 데이터를 기반으로 최적화된 배치를 수행하므로, 병렬 응용끼리 서로 자원을 과도하게 점유하는 문제가 줄어든다. Kokkos 환경에서 추가적인 수정 없이 간단히 컴파일 옵션만 변경해도 적용할 수 있기 때문에, HPC 분야에서 보편적으로 사용되는 Kokkos 기반 응용에 폭넓게 확산될 가능성이 높다.

CPU-GPU 혼합 노드뿐 아니라, 초대형 슈퍼컴퓨팅 환경에서 자원 공유 효율을 향상시키는 기본 모듈로 발전할 수 있다. HPC 환경 전반에서 CPU 병렬 응용의 고도화를 유도할 수 있어 생산성 및 성능 양 측면에서 큰 기여를 기대한다.