[2024-2]

KEASE 기술서

한국형 엑사스케일 응용 SW 개발 환경 프레임워크

Kokkos 프레임워크에서 ExaMPM 테스트

2024. 04. 20

숭 실 대 학 교

[개정이력]

개정 일자	버전	설명	개정페이지	작성자	확인자
2024.4.20	1.0	INITIAL DRAFT	전체	김임범	최재영

목 차

1. 연구의 개요

이 문서는 Kokkos documentation에서 다루지 않는 내용을 기술함. Kokkos라는 단어가 기본적으로 모호하게 표현되었기에, 여기서는 Kokkos 대신 Kokkos 프레임워크라는 표현을 사용함. 설명에 앞서, Kokkos라는 단어가 포함되어 있어 다양하게 해석될 여지가 존재하는 단어들을 정의함.

- Kokkos : 병렬 프레임워크 자체를 지칭
- Kokkos Environment : Kokkos 프레임워크 내부에 속한 모듈 전체를 지칭.
  구체적으로는, Kokkos 내부에 속한 모듈인 kokkos-core, kokkos-kernels, kokkos-tools, pykokkos, kokkos-remote-spaces, kokkos-resilience 전체를 포괄적으로 지칭함.
- Kokkos programming model : 일반적인 응용 소프트웨어 소스 코드 내에서 보여지는 Kokkos 프레임워크의 구조
- Kokkos 코드 : Kokkos programming model을 준수하여 작성된 C++ 기반 응용 소프트웨어 소스 코드

2. Kokkos 사용을 위한 Kokkos 프레임워크 분석

```
View<double*> data("data", size);
for (int64_t i = 0; i < size; ++i) {
  data(i) = ...read from file...
}

double sum = 0;
Kokkos:: parallel_reduce(
  "Label",
  RangePolicy<SomeExampleExecutionSpace>(0, size),
  KOKKOS_LAMBDA (const int64_t index, double & valueToUpdate) {
    valueToUpdate += data(index);
  },
  sum
) ;
```

[그림 1] Kokkos 코드 예시

1) Kokkos 프레임워크 기본

Kokkos 프레임워크는 C++ 기반의 독자적인 병렬화 및 메모리 추상화 구문을 통해, 응용 소프트웨어의 높은 성능을 보장하며 높은 이식성을 가지는 단일 노드 병렬 프로그램을 생성하는 라이브러리임. 동일한 Kokkos 코드로 host 병렬화 및 device 병렬화가 가능함. 타겟 하드웨어의 스펙에 맞추어 작업의 병렬화 방식을 조정해야 하므로, Kokkos 프레임워크 내에서 configuration을 통해 각 하드웨어에 최적화된 파라미터를 제공함.

2) Kokkos 코드의 최적화 과정

Kokkos 코드의 최적화 작업은 오직 해당 코드의 컴파일 단계에서만 수행함. 다르게 말하자면, Kokkos 프레임워크는 응용의 실행 이전 환경 설정 단계 및 응용의 런타임 단계에서 최적화를 수행하지 않음. Kokkos 코드의 컴파일은 크게 2단계로 나뉨. 첫 번째 단계에서, Kokkos 코드는 컴파일 시에 C++

template metaprogramming을 통해 C++ 소스 코드 및 GPU 전용 소스 코드(주로 Cuda 코드)로 변환됨. Kokkos 프레임워크가 설치 과정에서 생성한 configuration을 참고하여 타겟 머신에 최적화된 소스 코드로 매핑 작업을 수행함. 두 번째 단계에서, 각 소스 코드는 Kokkos 프레임워크 설치 단계에서 지정한 컴파일러를 통해 바이너리 파일로 컴파일됨. 즉, 컴파일러에서 처리하지 못하는 최적화 기법은 Kokkos를 통해 처리하여야 함. 메모리 관리 및 기본 하이퍼 파라미터 설정을 통한 최적화는 Kokkos에서 수행하며, loop unrolling 및 SIMD instruction 활용과 같은 로우 레벨 최적화는 컴파일러에서 수행함. 최적의 하이퍼 파라미터 설정을 통한 최적화는 응용 소프트웨어 개발자가 수행하며, 미리 최적화된 수치 라이브러리 제공을 통한 최적화는 kokkos-kernels가 수행함. Kokkos Environment 중 kokkos-core에서 적절한 코드 매핑을 통해 컴파일러에서 수행하는 최적화를 Kokkos 프레임워크 수준에서 처리할 수 있으며, kokkos-tools를 활용하면 최적의 하이퍼 파라미터 설정 또한 응용 소프트웨어 개발자가 아닌 Kokkos 프레임워크 내에서 수행할 수 있음.

Kokkos 프레임워크는 응용 소프트웨어에서 수행하고자 하는 작업을 추상화하도록 요구하고, 추상화된 작업을 높은 성능을 보이도록 구체화하는 방식으로 응용 소프트웨어를 최적화함. 이를 위해 Kokkos 프레임워크는 하드웨어 추상화 개념을 도입하여 임의의 아키텍처에 대한 최적화를 지원하도록 구현되었으며, Kokkos programming model에서는 코드 추상화를 지원하여 응용 소프트웨어 개발자가 시스템에 독립적인 소프트웨어 코드를 작성할 수 있도록 함. Kokkos 프레임워크에서 채용한 하드웨어 추상화 모델은 단일 노드 환경에서 On-Chip network를 통해 core, on-package memory, DRAM, NVRAM, accelerator 간 통신하는 구조임. 유의할 점으로, 하드웨어 추상화 모델이 단일 노드 환경이기에 응용 소프트웨어 개발자는 별도로 MPI를 사용하여 멀티 노드 환경의 코드로 변환하거나 Kokkos 프레임워크의 한정적인 멀티 노드 지원에 의존해야 하고, accelerator 동작은 해당 accelerator 컴파일러가 결정하기에 Kokkos 프레임워크의 최적화 지원 범위는 멀티 코어에서의 공유 메모리 관리 및 device 간 통신 최적화로 한정됨. Kokkos programming model에서 지원하는 코드 추상화 방식은 응용 소프트웨어 개발자가 응용 작업이 어떻게 동작하는지를 기술하도록 하는 것이 아닌, 어떻게 동작해야 하는지를 기술하도록 유도함. 코드 추상화는 데이터 추상화 및 작업 추상화로 나뉨. 데이터 추상화는 데이터가 어디 위치하였는지 (즉, 데이터가 host 메모리에 위치하였는지, 혹은 device 메모리에 위치하였는지), 데이터가 어떻게 저장되어 있는지 (즉, sparse matrix의 경우 데이터 포맷이 Bsr, Ccs, Coo, Crs 등 중 무엇인지), 데이터 접근이 주로 어떻게 이루어지는지(순차적으로 접근하는지 혹은 랜덤 인덱스 접근인지, 데이터를 아토믹하게 접근해야 하는지) 기술함. 작업 추상화는 작업을 어디서 수행할 것인지(CPU에서 혹은 GPU에서 실행할 것인지), 병렬 작업이 어떤 작업인지 (즉, parallel인지, reduction인지, scan인지), 작업 수행에서의 정책이 무엇인지 (즉, range 혹은 team 단위 수행인지, scheduling은 static인지 dynamic인지) 기술함. 작업 추상화는 공식적으로 위 3종류로 나뉘나, kokkos-kernels의 연산을 호출하는 행위 또한 어떤 작업을 수행할 것인지에 대한 작업 추상화로 볼 수 있음.

3) Kokkos Environment의 모듈 및 관계

Kokkos Environment의 주요 3개 모듈은 kokkos-core, kokkos-kernels, kokkos-tools임. 각 모듈을 간략히 요약하자면 kokkos-core는 template metaprogramming을 통한 최적화를 수행하는 최적화 베이스 모듈이고, kokkos-kernels는 응용 소프트웨어 개발자가 수치 라이브러리를 호출한 경우 kokkos-kernels 자체 수치 라이브러리 혹은 Kokkos 프레임워크 설치 과정에서 명시한 외부 수치 라이브러리 호출을 수행하는 모듈임. 그리고 kokkos-tools는 kokkos-core 및 kokkos-kernels에서 수행하지 않으나 Kokkos 프

레임워크 내에서 지원해야 하는 다른 작업(프로파일링, 디버깅, 오토 튜닝 등)을 수행하는 extension이 포함된 모듈임.

kokkos-cores는 Kokkos programming model을 준수한 코드를 적절한 C++ 코드와 cuda 코드로 변환하는 작업을 수행하기 위한 내부 구현 및 Kokkos 프레임워크 기반 응용 소프트웨어 동작의 event tracking 구현 및 kokkos-tools를 위한 인터페이스를 포함한 모듈임. 코드의 변환은 최적화 관련 변환 및 이식성 관련 변환으로 나뉨. 최적화 관련 변환은 작업을 블록 형태로 분해한 후 블록 크기와 같은 튜닝 파라미터를 기본 값으로 설정하는 작업 및 Kokkos 프레임워크 설치 당시 설정한 타겟 아키텍처에 맞는 컴파일 옵션을 컴파일러에 전달하는 작업으로 나뉘며, 튜닝 파라미터의 기본 값 설정은 독립적인 configuration 파일이 존재하는 것이 아닌 <exec_space>/Kokkos_<exec_space>_<exec_pattern>.hpp에서 독자적으로 결정한다. 그리고 컴파일 옵션 설정은 Kokkos 프레임워크의 cmake 파일을 통해 단순 매핑 형식으로 결정함.

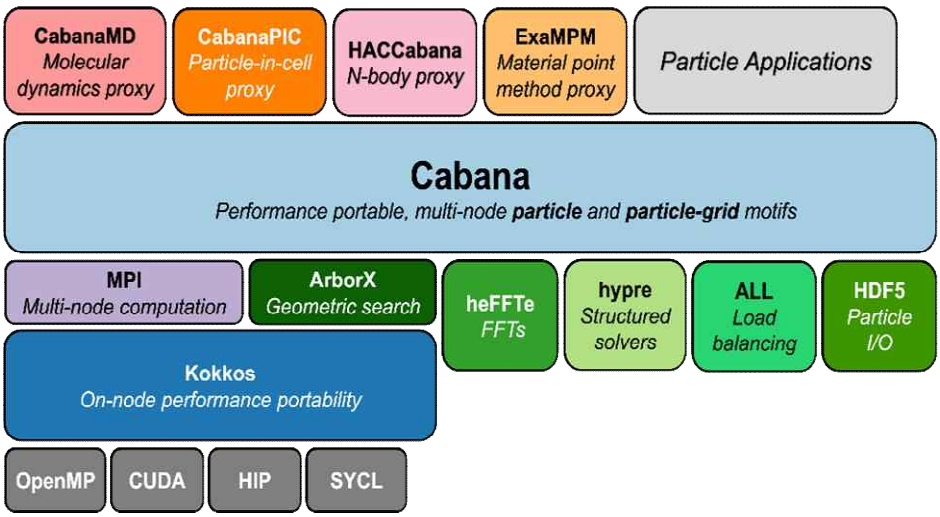
이식성 관련 변환은 kokkos-core 코드 전반에서 template metaprogramming을 통해 수행되며, View 간 데이터 이동은 모두 kokkos-core의 View 내부 커널 함수로 치환하는 형태로 데이터 이동 최적화를 Kokkos 프레임워크 내에서 수행할 수 있음. kokkos-tools를 위한 인터페이스는 Kokkos_Profiling_Interface.hpp에 정의되어 있음. Kokkos 프레임워크에서 지원하는 event tracking이 충분하지 않을 경우에 impl/Kokkos_Profiling.cpp에서 기능을 추가해야 하지만, 이 작업은 Kokkos 프레임워크에서 제공하는 정식 기능이 아님에 주의해야 함.

kokkos-kernels는 응용 소프트웨어에서의 수치 라이브러리 호출을 kokos-kernels 자체 수치 라이브러리 호출로 연결하거나 외부 수치 라이브러리 호출로 변경함. 본 연구는 수치 라이브러리 개발도 포함되어 있기에 외부 수치 라이브러리 호출은 무시하여도 됨. 다르게 말하자면, kokkos-kernels는 임의의 머신에 최적화된 수치 라이브러리를 제공하는 모듈로 봐도 무방함. kokkos-kernels은 크게 Dense Linear Algebra, Sparse Linear Algebra를 지원함. Dense Linear Algebra에서는 BLAS, LAPACK, Batched BLAS, Batched LAPACK을 지원하며, batched 작업은 accelerator 환경에서 최적화되어야 함. Sparse Linear Algebra에서는 sparse matrix/vector를 담는 자료구조인 Sparse Container 및 sparse matrix에서 주로 사용되는 연산인 SpMV, SpADD, SpGEMM을 지원함. Kokkos에서 자체적으로 지원하는 Sparse Container의 종류는 Bsr matrix, Ccs matrix, Coo matrix, Crs matrix임. kokkos-kernels의 Sparse Linear Algebra는 Sparse Container의 포맷을 handle 형태로 전달받는 형식으로, kokkos-core의 sparse matrix와는 독립적으로 구현되어 있음.

kokkos-tools는 Kokkos 프레임워크 기반 응용 소프트웨어 개발에 필요한 외부 도구를 내장한 모듈임. 서드 파티 extension 개발을 위한 프레임워크는 kokkos-core에 위치해 있음. kokkos-tools를 위하여 kernels, regions, metadata, memory alloc/free, dualview operation 등 여러 종류의 event tracking을 지원함. kokkos-tools를 위한 event logging은 오버헤드가 0에 가까울 정도로 매우 작음. Kokkos 프레임워크 기반의 응용 소프트웨어는 모두 tool-enable 상태로 컴파일되기에, 툴 사용을 위한 재컴파일이 필요하지 않음. 기본적으로 메모리 프로파일링 도구, 커널 동작 시간 정보 제공 도구, 휴리스틱 혹은 decision tree 기반 오토 튜닝 도구를 제공함.

3. ExaMPM

ExaMPM은 MPM (Material Point Method)을 exascale로 동작할 수 있도록 Kokkos 프레임워크를 기반으로 하여 작성된 응용 소프트웨어임. MPM은 메쉬 기반으로 계산하는 방식이 아닌 연속체 기반의 파티클 계산으로 한 유체 역학 시뮬레이션임. ExaMPM은 Kokkos 프레임워크로 작성된 파티클 기반 시뮬레이션 라이브러리인 Cabana를 통해 동작함.



[그림 2] Cabana 개요도

ExaMPM의 내부 연산은 매 time step마다 4종의 커널을 수행하며, 해당 커널 내에서 병렬화가 수행됨. 4종의 커널 중 p2g 및 g2p 커널 수행 시간이 전체 시간 대비 60% 이상 차지함. p2g 커널에서는 수치 라이브러리 호출을 통해 명시적으로 dense matrix & vector 연산을 수행하지만, g2p 커널에서는 수치 라이브러리 호출 없이 자체적으로 코드를 작성하여 연산을 수행함. 연산의 크기는 그리드 크기, 셀 크기, 셀 당 입자 개수로 인해 정해지지만, 해당 수치들은 모두 사용자의 입력으로 결정되기 때문에 작업의 특징에 따른 특별한 최적화 기법 적용이 까다로워 보임.

4. Kokkos 프레임워크에서 ExaMPM 테스트

KNL 단일 노드에서 환경 설정에 따른 ExaMPM의 수행 속도를 실험하였음. Execution Space로 openmp를 지정하였으며, kokkos-tools의 SimpleKernelTimer로 프로파일링을 수행하였음. Physical cell size가 1이고, cell당 particle이 100개이고, time step이 1000번인 케이스에 대한 프로파일링 결과는 다음과 같음.

```
(Type) Total Time, Call Count, Avg. Time per Call, %Total Time in Kernels,
%Total Program Time

-----------------------------------------------------------------------

Regions:

- solve

(REGION) 25.456093 1 25.456093 106.197430 96.154491

- Cabana::Grid::scatter

(REGION) 0.004629 3000 0.000002 0.019313 0.017486

- Cabana::Grid::gather

(REGION) 0.002796 2000 0.000001 0.011666 0.010563

-----------------------------------------------------------------------

Kernels:

- p2g

(ParFor) 8.435860 1000 0.008436 35.192622 31.864507

- g2p

(ParFor) 7.243255 1000 0.007243 30.217326 27.359717

- correct_particles

(ParFor) 2.752771 1000 0.002753 11.483979 10.397956

- Kokkos::ScatterView::ResetDuplicates [duplicated_position_correction]

(ParFor) 1.103923 1000 0.001104 4.605334 4.169814

- Kokkos::ScatterView::ResetDuplicates [duplicated_force]

(ParFor) 0.956391 1000 0.000956 3.989859 3.612544

- init_particles_uniform

(ParRed) 0.801744 1 0.801744 3.344706 3.028402

- momentum_cfl_timestep

(ParRed) 0.482111 1000 0.000482 2.011266 1.821063

- Cabana::Grid::ParticleGridMigrate::count
```

```
(ParRed) 0.322937 1000 0.000323 1.347225 1.219820
- Kokkos::ScatterView::ReduceDuplicates [duplicated_mass]
(ParFor) 0.256879 1000 0.000257 1.071644 0.970300

... (생략) ...

- Kokkos::View::destruction []
(ParFor) 0.000218 3 0.000073 0.000910 0.000824
----------------------------------------------------------------------
Summary:

Total Execution Time (incl. Kokkos + non-Kokkos): 26.47416 seconds
Total Time in Kokkos kernels: 23.97054 seconds
-> Time outside Kokkos kernels: 2.50362 seconds
-> Percentage in Kokkos kernels: 90.54 %
Total Calls to Kokkos Kernels: 21007
----------------------------------------------------------------------
```

스레드 수 변화에 따른 ExaMPM 수행 시간을 측정하였음. 스레드 개수가 늘어남에 따라 수행 시간이 개선되나 동시에 데이터 복사 커널로 인하여 성능이 큰 폭으로 개선되지 못함. 스레드 개수가 34개일 경우의 데이터 복사 커널의 수행 시간은 2.76초이나, 스레드 개수가 68개인 경우 데이터 복사 커널의 수행 시간이 7.00초로 큰 폭으로 상승함.