[2024-5]

KEASE 기술서

한국형 엑사스케일 응용 SW 개발 환경 프레임워크

CPU 기반 HPC 응용을 위한 최적 자원 설정 기술

2024. 04. 20

성 균 관 대 학 교

[개 정 이 력]

개정 일자	버전	설명	개정페이지	작성자	확인자
2024.4.20	1.0	Kokkos 기반 통합 프로파일러 개발	전체	이현수	서의성

목 차

1. 연구의 개요

본 문서는 Kokkos 기반 HPC 응용에 대한 시스템 관점에서의 프로파일링을 위한 통합 프로파일러 개발 및 분석에 관한 문서이다. 최신 HPC 환경 애플리케이션들은 GPU와 같은 병렬 컴퓨팅 연산 가속 장치를 활용하는 방향으로 넘어감에 따라 실제 CPU 자원 사용량이 미미한 수준으로 보고 되고 있다. 본 연구에서는 이러한 사실에 착안하여 HPC 환경에서 GPU 뿐만 아니라 CPU 자원을 적극적으로 활용하여 HPC 응용들을 지원할 수 있는 스케줄링 기법 개발을 목표로 하고 있다. 이러한 목표를 달성하기 위해서는 우선적으로 Kokkos 기반으로 개발된 응용들에 대한 시스템 자원 사용량에 대한 세밀한 분석이 필요하다. 이를 위해서 본 연구에서는 우선적으로 이기종 연산 장치 환경에서 병렬 컴퓨팅을 통합적으로 지원 가능한 프로그래밍 모델을 제공하는 Kokkos를 기반으로 Linux Perf 기능을 통합한 프로파일러 개발 및 HPC 응용에 대한 CPU 실행에 대한 특성 분석을 진행하였다.

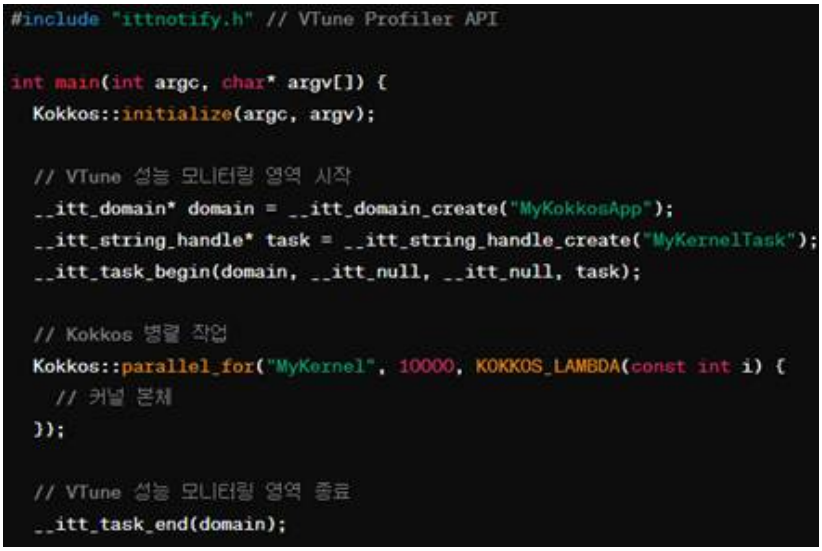
2. Kokkos 기반 시스템 프로파일링 툴

1) Kokkos Tools

Kokkos는 이기종 컴퓨팅 유닛을 활용하기 위한 통합 프로그래밍 모델을 제공하며, 제공되는 병렬 컴퓨팅을 위한 인터페이스를 활용하여 개발할 경우 컴파일 타임에 각 디바이스에 최적화된 코드로 변환된다. 기본적으로 Kokkos에서는 parallel_for, parallel_scan, parallel_reduce와 같은병렬 컴퓨팅 인터페이스를 제공하며, 해당 인터페이스에 반복 횟수와 단일 스레드가 수행할 동작에 대한 정의를 하면, input과 주어진 스레드 풀을 고려하여 병렬 컴퓨팅 연산을 완성해준다.

Kokkos는 이렇게 개발된 병렬 컴퓨팅 응용에 대해서 유저가 최적화할 수 있도록 도와주는 유용한 툴들을 제공한다. 해당 툴에는 각 parallel 인터페이스에 대해서 유저 계층에서 프로파일링 할 수 있는 Kokkos Profiling 기능이 포함되어 있다. 이러한 프로파일링 기능은 유저 코드에 직접 명시적으로 삽입되어 사용되는 방식과 Kokkos에서 미리 정의되어 제공되는 Utilization, Memory Analysis Hooking 라이브러리를 Kokkos 응용을 수행할 때 삽입하여 실행하는 두 가지 방식을 제공한다.

기본적으로 Kokkos Profiling 기능은 응용 계층에서의 정보만 획득 가능하기 때문에, 각 병렬 컴퓨팅 유닛에 최적화된 프로파일링 기능을 확장 가능한 인터페이스도 제공하고 있다. 예로 들어 다음 [그림 1]은 Intel CPU 기반 프로파일링 툴인 Vtune을 활용하여 Kokkos 응용을 프로파일링 하기 위한 방법을 보여준다.



```
#include "ittnotify.h" // VTune Profiler API

int main(int argc, char* argv[]) {
  Kokkos::initialize(argc, argv);

  // VTune 성능 모니터링 영역 시작
  __itt_domain* domain = __itt_domain_create("MyKokkosApp");
  __itt_string_handle* task = __itt_string_handle_create("MyKernelTask");
  __itt_task_begin(domain, __itt_null, __itt_null, task);

  // Kokkos 병렬 작업
  Kokkos::parallel_for("MyKernel", 10000, KOKKOS_LAMBDA(const int i) {
    // 커널 본체
  });

  // VTune 성능 모니터링 영역 종료
  __itt_task_end(domain);
```

[그림 1] Intel Vtune을 활용한 Kokkos Profiling

기본적인 Kokkos의 명시적인 프로파일링 기능 삽입을 기반으로 내부에서 호출된 프로파일링 기능을 Vtune 라이브러리를 통해 구현한 것이다. 이와 같이 Utilization, Memory Analsysis와 같이 유저 계층 정보를 얻기 위해 미리 정의된 라이브러리를 Hook 형태로 사용하지 않는다면, 기존 응용에 대한 소스 코드를 수정할 필요가 있다.

2) AutoThread Executor

본 문서에서는 Kokkos 기반으로 개발된 HPC 응용에 대해서 추가적인 소스 코드 변경 없이 시스템 수준의 프로파일링 기능을 삽입하기 위한 AutoThread Executor를 제안한다. AutoThread Executor는 파리미터로 HPC 응용을 받아서 실행되며, 전술한 Kokkos의 병렬 연산 인터페이스인 parallel_for, parallel_sccan, parallel_reduce 함수의 호출 전과 호출 후에 프로파일링 기능을 실행 중에 Hook 형태로 삽입한다.

또한 AutoThread Executor는 Kokkos에서 제공하는 응용에서 수행되는 병렬 연산에 대한 스레드 개수 조정 옵션을 이용하여 스레드 개수를 1개부터 시스템의 사용 가능한 Core 개수까지 증가시키면서 샘플링을 수행한다. 이때 기본적으로 샘플링 횟수는 각 스레드 개수 옵션 당 10회로 설정되어 있다. 이러한 작업을 통해 병렬 연산에 대해서 스레드 개수 증가에 따른 성능 메트릭의 변화를 관찰이 가능하다.

이와 같이 샘플링된 데이터는 AutoThread Executor가 in-memory 형태로 관리되며 프로그램 종료 전에 Json 파일 타입과 DB 파일 타입으로 출력된다. [그림 2]는 실제 출력된 데이터의 예시를 보여주며, 데이터는 샘플링 횟수, 스레드 개수, 커널 종류 별로 구성된된다. 이러한 Json과 DB 형태의 파일은 프로파일링되어 나온 많은 데이터를 시각화 및 유의미한 정보를 추출하기에 적합하다.

```
kokkosautothreads.db
kokkosautothreads.json
kokkosautothreads.summary.json
```


```
{
  "exec_time": 13058,
  "hook_type": "parallel_for",
  "hw_cache_misses": 0,
  "hw_cache_references": 5,
  "kernel_id": 16,
  "kernel_name": "Kokkos::View::initialization [b_lev_ind] via memset",
  "sw_page_faults": 0,
  "sw_page_faults_maj": 0,
  "sw_page_faults_min": 0
},
{
  "exec_time": 19246,
  "hook_type": "parallel_for",
  "hw_cache_misses": 70,
  "hw_cache_references": 147,
  "kernel_id": 12,
  "kernel_name": "13fillColorsMap",
  "sw_page_faults": 0,
  "sw_page_faults_maj": 0,
  "sw_page_faults_min": 0
},
```


[그림 2] 프로파일링 데이터 추출 및 구조

3) MPerf

```
#include "mperf/LinuxPerf.hpp"
using HLMType = MPerf::HLMeasureType;

// Measurements
using tracerType = MPerf::Tracers::LinuxPerf::Tracer;
using measureType = std::unique_ptr<MPerf::Measure>;
tracerType linuxTracer;
measureType hwCacheMeasure, pgFaultMeasure;

hwCacheMeasure = linuxTracer.MakeMeasure({
  HLMType::HWCacheReferences,
  HLMType::HWCacheMisses,
});

pgFaultMeasure = linuxTracer.MakeMeasure({
  HLMType::SWPageFaults,
  HLMType::SWPageFaultsMaj,
  HLMType::SWPageFaultsMin,
});


Insert Point{
  hwCacheMeasure->DoMeasure();
  pgFaultMeasure->DoMeasure();

  kernelCacheTick = hwCacheMeasure->GetJSON();
  kernelPgFaultTick = pgFaultMeasure->GetJSON();
}
```

[그림 3] MPerf Interface

MPerf는 실제 시스템 수준에서 HPC의 응용의 성능 메트릭을 프로파일링 하는 핵심 기능으로써 기본적으로 Linux Perf 라이브러리를 이용하여 구현되어 있다. MPerf는 본 연구 목적에 따라 HPC 응용의 메모리/캐시 접근 패턴을 파악하기 위해서 Page Fault, Cache Reference, Cache Miss 성능 메트릭을 기본적으로 제공하고 있다. AutoThread Executor를 통해 각 병렬 연산 인터페이스에 삽입되는 프로파일링 기능은 MPerf를 이용하여 구현된다. MPerf는 [그림 3]과 같이 간단하게 사용 가능한 인터페이스를 제공하고 있다.

먼저 MPerf를 이용하기 위해서 MPerf 네임스페이스에 LinuxPerf와 Measure 타입의 변수를 정의한다. 그리고 MakeMeasure를 통해 MSR 기반의 성능 메트릭 타입을 지정하여 프로파일링 시에 추출된 정보를 지정한다. 마지막으로 정의된 성능 메트릭에 대해서 프로파일링이 실제로 동작되길 바라는 시점에 DoMearsure() 함수 호출을 통해 실행한다. AutoThread Executor는 Kokkos의 병렬 연산 커널들에 진입점과 반환점에서 DoMearsure()가 호출되도록 개발되었다.

3. 실험을 통한 테스트

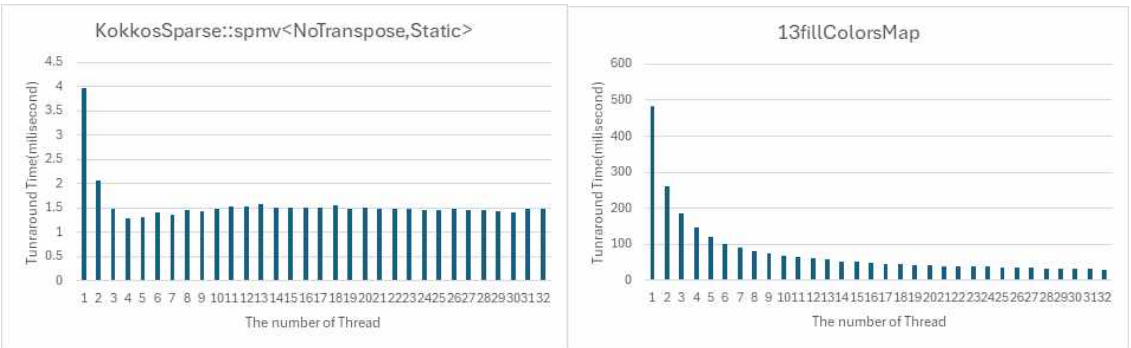
1) 실험 환경

CPU	Intel(R) Xeon(R) Gold 6130(16 core, 2 processor)
Memory	DDR4 60GB
Workload	HPCG 3.0 based on Kokkos

2) HPCG 워크로드 분석

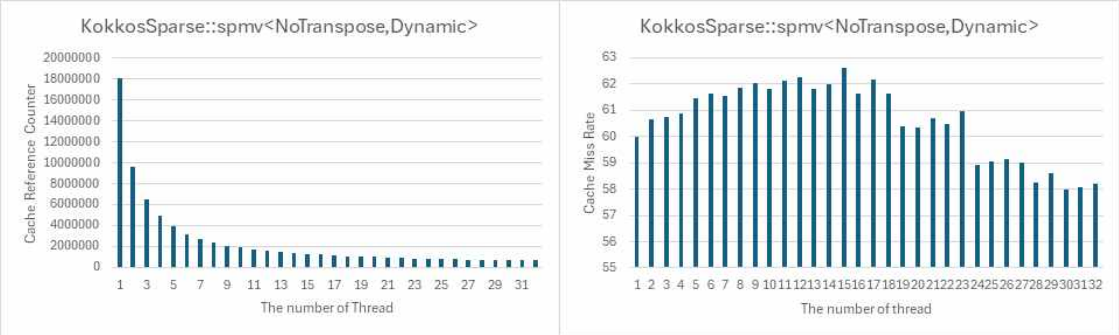
본 문서에서 설명한 프로파일링 도구를 이용하여 CPU 환경에서 병렬 연산 벤치마크 워크로드로 이용되는 HPCG를 Kokkos 기반으로 컴파일하고 Input size를 128로 고정하고 프로파일링하였다.

HPCG에서 주요 커널은 Colormap과 SPMV 커널로 본 문서에서는 해당 커널에 대해서 중점적으로 분석한다. [그림 4]의 실험 그래프에서 보이는 바와 같이 두 커널 모두 스레드를 증가 시킬수록 성능이 포화되는 현상을 보이며, 특히 SPMV 커널과 같이 항상 스레드가 클수록 최적의 성능을 보장하지 않는 것을 알 수 있다.



[그림 4] 스레드 증가에 따른 실행 시간

다음 [그림 5]는 HPCG의 SPMV 커널의 스레드 증가에 따른 Cache Reference Count와 Cache Miss Rate 결과를 보여주고 있다.



[그림 5] 스레드 개수 변화에 따른 캐시 접근

동일한 작업량을 기준으로 스레드 개수만 늘렸을 경우 완전히 병렬화된 작업의 경우 메모리 접근 총량(Cache Reference)가 같아야 하지만, SPMV와 같은 희소 행렬 벡터 곱연산과 같이 오히려 메모리 접근 수가 줄어드는 것을 확인할 수 있다. 또한 Cache miss rate 결과를 통해 SPMV는 Cache 자원에 대한 경쟁은 있지만 스레드가 증가하였다고 해서 Cache에 대한 경합이 항상 발생하는 것이 아님을 알 수 있다. 이와 같은 결과를 통해 병렬 연산 커널들의 경우 다양한 메모리 접근 패턴 결과를 가질 수 있으며 이러한 결과를 각 커널마다 성능 최적화된 자원할당 모델 수립을 하는데 이용될 수 있다. 따라서, 각 커널은 보다 세밀한 자원 접근 패턴 분석을 추가적으로 수행하여 메모리 및 공유 자원에 대해서 경합이 발생하는지 혹은 데이터 지역성이 증가되는지에 따라 각 커널의 최적화된 스레드 결정과 자원 할당을 해야한다.

[그림 6]은 AutoThread Executor로부터 추출된 RAW 데이터를 기반으로 각 커널의 스레드 증가에 따른 각 성능 메트릭과의 상관계수를 출력한 결과다. HPCG는 총 25개의 커널로 구성되어 있으며, 그 중 주요 연산은 Colormap과 SPMV 연산으로 초기화를 제외하고 전체 연산 시간의 대부분을 차지하고 있다. 전술한 바와 같이 HPCG의 Colormap과 SPMV의 경우 스레드 개수가 증가할수록 성능이 포화되며, Cache 효율성이 증가하는 특성을 가지고 있다. mapscan의 경우 양상이 정 반대로 스레드 개수가 증가할수록 성능이 감소하며, cache 효율성 또한 같이 떨어지는 특성을 보이며, ProlongationFunctor는 성능은 증가하지만 cache 효율성은 감소하는 특성을 보이고 있다.

	exec_time	hw_cache_misses	hw_cache_references	sw_page_faults
Kokkos::View::initialization [CrsMatrix: Values...	0.426576	-0.307161	-0.388052	0.396786
Kokkos::View::initialization [CrsMatrix: Global...	0.353203	-0.313424	-0.272168	0.294984
Kokkos::View::initialization [CrsMatrix: LocalI...	0.243292	-0.334176	-0.076283	0.100142
Kokkos::View::initialization [CrsMatrix: RowMap...	0.304076	-0.440072	-0.194269	0.192856
Kokkos::View::initialization [Matrix Diagonal] ...	0.384204	-0.202365	-0.191995	0.228145
Kokkos::View::initialization [f2cOperator] via ...	0.323188	-0.203600	-0.081474	0.117252
Kokkos::View::initialization [Vector: Values] v...	0.114923	0.276032	0.194075	-0.174998
Kokkos::View::initialization [f_row_level] via ...	0.300702	-0.101707	0.352670	-0.343236
Kokkos::View::initialization [b_row_level] via ...	0.326547	0.307799	0.341659	-0.300983
Kokkos::View::initialization [CrsMatrix: RowMap...	0.279261	0.173574	0.236106	-0.187934
Kokkos::View::initialization [f_lev_map] via me...	0.645150	-0.594131	0.299905	-0.090159
Kokkos::View::initialization [f_lev_ind] via me...	0.284056	-0.247987	0.140574	-0.147442
13fillColorsMap	-0.665890	-0.646355	-0.623488	-0.343434
7mapScan	0.477132	0.894442	0.734853	-0.305395
13fillColorsInd	-0.640746	-0.536846	-0.601490	-0.329579
Kokkos::View::initialization [b_lev_map] via me...	0.930793	-0.374292	-0.154444	-0.078213
Kokkos::View::initialization [b_lev_ind] via me...	0.635824	0.030629	0.092538	-0.150219
KokkosSparse::spmv<NoTranspose,Dynamic>	0.028530	-0.654706	-0.646788	-0.319858
8AlphaOne	-0.592467	-0.643912	-0.634590	-0.371531
10Dotproduct	-0.561042	-0.642701	-0.645056	-0.133087
Kokkos::View::initialization [z] via memset	0.666531	-0.610854	-0.035741	-0.099135
12LeveledSweep	0.534570	0.784707	0.842827	-0.219355
18RestrictionFunctor	-0.034798	-0.652678	-0.650720	-0.352542
KokkosSparse::spmv<NoTranspose,Static>	-0.331512	-0.652031	-0.632522	-0.371785
19ProlongationFunctor	0.603211	-0.630212	-0.646495	-0.188684
Kokkos::Impl::host_space_deepcopy_double	0.543590	-0.753672	-0.748276	-0.090734

[그림 6] HPCG의 각 커널에 대해서 스레드 개수와 성능 메트릭간의 상관계수

HPCG 벤치마크 분석 결과, 스레드 증가에 따른 동기화 및 공유 자원 사용으로 성능 포화 현상이 발생함을 확인. 스레드 간 성능 간섭이 주요 요인이 된다. 따라서 본 문서에서 분석한 내용을 기반으로, 개별 커널의 성능을 최적화하기 위한 각 커널의 캐시 및 메모리 사용 특성을 고려하여 최적 스레드 개수와 배치를 결정하는 기법 고안할 필요성이 있다.