GPU 성능 분석을 위한 코드 삽입 도구 활용

김현준[O1] 홍성인[1] 한환수[2]
[1]성균관대학교 정보통신대학
[2]성균관대학교 소프트웨어대학
{hjunkim, sungin.h, hhan}@skku.edu

Utilizing Code Instrumentation Tools for GPU Performance Analysis

Hyunjun Kim[O1] Sungin Hong[1] Hwansoo Han[2]
[1]College of Information and Communication Engineering, Sungkyunkwan University
[2]College of Software, Sungkyunkwan University

요 약

대량의 쓰레드가 동시에 하나의 명령어를 수행하는 GPU에서 성능을 분석하고 최적화를 적용하는 것은 병렬 애플리케이션 개발자에게 시간 소모가 큰 일이다. 따라서 본 논문에서는 성능 분석을 위한 GPU 코드 삽입 도구인 SASSI를 이용하여 1) 쓰레드 단위의 메모리 접근 패턴 정보와 2) 워프 단위의 스케줄링 정보를 추출하는 도구를 개발하였다. 두 정보를 추출하는데 프로파일링 정보를 추출하지 않는 프로그램 실행 시간 대비 평균 36.4배와 58.5배의 오버헤드가 발생하였다. 본 도구에서 추출한 정보를 활용하여 GPU 애플리케이션의 성능을 분석하고, 메모리 접근 최적화와 같은 최적화 기법에 활용할 수 있도록 한다.

1. 서 론

수천에서 수만 개의 쓰레드가 병렬적으로 연산을 동시에 수행하는 GPU가 범용 계산 분야에서도 성공적으로 적용 가능해지면서, 모바일 디바이스부터 초고성능 컴퓨터의 가속기까지 넓은 범위에서 활용되고 있다. GPU 프로그래밍을 위한 대표적인 모델로는 CUDA와 OpenCL이 있으며, OpenMP와 OpenACC와 같은 지시어-기반의 프로그래밍 모델 또한 사용 가능하다. GPU를 이용한 애플리케이션의 가속화는 병렬 프로그래밍의 높은 복잡도 때문에 이를 효과적으로 활용하여 최대 성능을 내는 것은 쉽지 않다. 이를 위해 GPU의 성능 분석 및 자동 최적화 도구들에 대한 연구가 진행되었다[1].

GPU의 메모리는 온/오프-칩 메모리로 구성되어 있으며, 두 메모리 소자 간에는 10-100배의 지연시간 차이가 존재한다. GPU의 성능 향상을 위해서는 오프-칩 메모리보다 상대적으로 빠른 온-칩 메모리를 활용하는 것이 중요하다. 하지만 다수의 쓰레드가 동시에 하나의 명령어를 수행하는 GPU의 SIMT (Single Instruction Multiple Threads) 구조에서, 각 쓰레드에서 요청하는 메모리 요청을 분석하고 관리하는 것은 쉽지 않다. 또한, 스케줄링 알고리즘에 따라 다르게 수행하는 쓰레드들의 메모리 접근 패턴을 분석하는 것은 쉽지 않다.

SASSI[2]는 사용자가 정의한 코드를 GPU 커널 코드에 삽입하여 원하는 정보를 런-타임에 추출하는 코드 삽입 도구이다. 본 논문에서는 SASSI를 이용하여 GPU 커널의 1) 쓰레드 단위의 메모리 접근 패턴 정보와 2) 워프 단위의(NVIDIA에서 정의한, 명령어를 수행하는 기본 단위로 일반적으로는 32개의 쓰레드로 구성되어 있음.) 스케줄링 정보를 추출할 수 있는 도구를 개발하였다.

본 논문에서 개발한 메모리 접근 패턴 정보 추출 도구를 GPU 벤치마크 애플리케이션들을 대상으로 NVIDIA GTX980 GPU에서 수행한 결과 기존 프로파일 정보를 추출하지 않는 프로그램 실행 시간 대비 평균 36.4배의 오버헤드가 발생하였다. 반면, GPUOcelot[3]에서는 동일한 정보를 추출하는데 평균 2,459.3배의 오버헤드가 발생한 것을 확인하였다. 따라서 본 논문에서 제작한 도구를 이용하면 상대적으로 적은 오버헤드로 런-타임에 GPU 디바이스에 대한 성능 분석 정보를 추출하는 것이 가능하다. 이 도구에서 추출한 정보를 활용하여 GPU 커널의 메모리 사용 패턴을 분석할 수 있다. 예를 들어, 온-칩 내의 L1/L2 캐시의 사용 패턴을 분석하는 것이 가능하며, 이를 활용하여 효과적인 최적화가 가능하다.

본 논문의 구성은 다음과 같다. 2장에서 현재 공개되어 있는 GPU를 대상으로 프로파일링이 가능한 도구들에 대해 설명하고, 3장에서는 GPU의 메모리 구조와 실행모델, 그리고 4장에서는 SASSI를 활용한 프로파일 도구에 대해 설명한다. 5장에서는 실험을 통한 정보 추출 오버헤드에 대해 보인다. 마지막으로 6장에서 본 논문의 결론을 내린다.

2. 관련연구

본 장에서는 연구에 사용되고 있는 GPU 커널들을 대상으로 성능분석이 가능한 도구인 SASSI와 GPUOcelot에 대해 소개한다.

2.1 SASSI

SASSI는 NVIDIA의 NVLab에서 공개한 GPU 커널 코드 삽입 도구이다. 이 도구를 사용하면 사용자가 정의한 성능 분석 함수를 GPU의 특정 명령어의 전/후에 삽입하여 런

본 논문은 한국 연구재단의 지원을 받아 수행된 연구결과임 (NRF-2016M3C4A7952587)

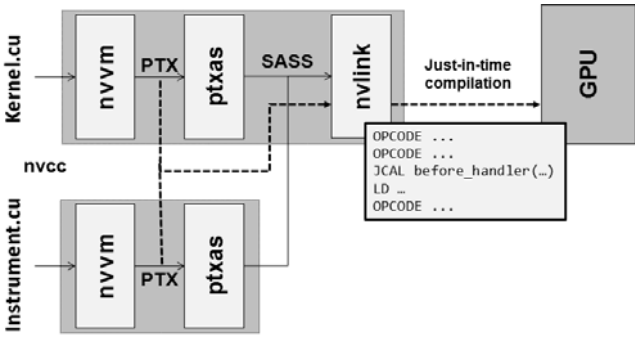

그림 3 SASSI 성능분석 코드 삽입 과정

-타임에 성능 분석 정보를 추출하는 것이 가능하다. NVIDIA의 Kepler와 Maxwell 아키텍처를 대상으로 성능 분석이 가능하며, 두 아키텍처보다 이전에 공개된 Fermi 아키텍처를 대상으로는 일부 기능만을 제공하고 있다.

2.2 GPUOcelot

GPUOcelot을 이용하면 GPU의 중간 언어(IR)인 PTX를 대상으로 GPU 에뮬레이션이 가능하며, PTX를 대상으로 코드를 삽입하여 성능분석 정보를 추출하는 것 또한 가능하다. NVIDIA의 Fermi 아키텍처 모델을 지원하고 있으며 이후 아키텍처들을 대상으로 하는 에뮬레이션은 지원하고 있지 않다.

3. GPU의 메모리 구조와 실행모델

본 장에서는 NVIDIA GPU의 메모리 구조와 실행모델에 대해 설명한다.

3.1 메모리 구조

GPU는 온/오프-칩 메모리로 크게 구분된다. 오프-칩 메모리에는 전역 메모리, 상수 메모리, 지역 메모리, 그리고 텍스처 메모리가 있다. 상수 메모리와 텍스처 메모리는 읽기 전용이며, 전역 메모리에 존재하는 데이터에 대해서는 읽기/쓰기가 가능하다. GPU 애플리케이션의 실행과 함께 호스트 메모리의 데이터는 전역 메모리로 복사되어 사용된다. 일반적으로 온-칩 메모리는 오프-칩 메모리에 비해 10-100배 정도의 짧은 지연시간을 필요로 하지만, 크기가 제한적이다. 따라서 GPU의 성능을 극대화하기 위해서는 온-칩 메모리를 적절히 활용하는 것이 중요하다.

3.2 실행모델

GPU는 일반적으로 긴 메모리 지연시간을 숨기기 위해 1,000-10,000개 이상의 쓰레드가 동시에 동일한 명령어를 수행하는 SIMT (Single Instruction Multiple Thread) 모델을 따른다. NVIDIA에서는 기본적으로 32개의 쓰레드 그룹을 워프라고 부르고, 다수개의 워프가 모여 TB (Thread Block)라고 부르며, 이는 GPU의 SM (Streaming Multiprocessor)에 스케줄링되는 기본 단위가 된다. 하나의 TB는 특정 SM에 할당되고, 워프 단위로 워프 스케줄러에 할당되어 수행된다. 상용화된 GPU 디바이스의 워프 스케줄러에서 사용하는 스케줄링 알고리즘은 공개되어 있지 않다. 하지만, 비교적 GPU의 동작을 실제와 유사하게 구현한 GPGPU-Sim 시뮬레이터[4]에서는 RR (Round Robin)과 GTO (Gready-Then Oldest) 스케줄링 모델을 제공하고 있다.

4. 프로파일러

SASSI 코드 삽입 도구는 GPU 커널의 명령어 (메모리, 제어 흐름, 읽기/쓰기 레지스터의 값), 베이직 블록, 커널 함수 호출의 전과 후에 대해 사용자가 정의한 코드를 삽입하는 것이 가능하다. 그림 1과 같이 Instrument.cu 코드 내에 정의된 성능분석 코드(before_handler())를 Kernel.cu 내의 커널에 컴파일-시간에 삽입한다. 성능분석 코드가 삽입된 커널은 실제 GPU에서 수행되며, 사용자가 정의한 성능분석 정보들을 추출하게 된다. 대표적인 예로, 특정 명령어가 수행되기 전에 호출할 코드를 삽입하여 해당 명령어가 수행되기 전의 특정 레지스터의 값을 추출하거나 혹은 임의로 수정하는 것이 가능하다. 본 논문에서는 SASSI 코드 삽입 도구를 이용하여 GPU 커널의 동작 원리를 분석하는데 필요한 쓰레드 단위의 메모리 접근 패턴과 워프 단위의 스케줄링 정보를 추출하는 도구를 작성하였다.

4.1 메모리 접근 패턴 추출 도구

본 도구에서는 GPU 내의 쓰레드들이 GPU 커널 내의 메모리 명령어들을 수행하면서 요청하는 메모리 주소들의 정보를 추출하는데 목적이 있다. 이를 구현하기 위해 SASSI 코드 삽입 도구를 사용하였고, 작성한 코드가 GPU 커널의 모든 전역 메모리 읽기/쓰기 명령어 ({ld/st}.global.{data_type})가 호출되기 전에 쓰레드 ID, 접근 메모리 주소 등을 추출하도록 작성하였다.

4.2 워프 스케줄링 정보 추출 도구

본 도구에서는 각 SM에 할당되어 수행되는 워프들의 스케줄링 정보를 추출하는데 목적이 있다. 이를 구현하기 위해 커널함수 호출 전과, 커널함수 종료 전에 수행되는 EXIT 명령어 호출 전에 clock() 함수를 이용하여 정의한 자료구조에 워프의 시작과 끝의 클락-틱 시간을 저장하였다.

5. 성능비교

본 장에서는 SASSI를 이용하여 작성한 GPU 커널 정보 추출 도구의 오버헤드를 측정하기 위해 NVIDIA GTX980 (Maxwell 아키텍처) GPU에서 실험하였다. 오버헤드는 프로파일 정보를 추출하지 않는 프로그램 실행 시간을 대

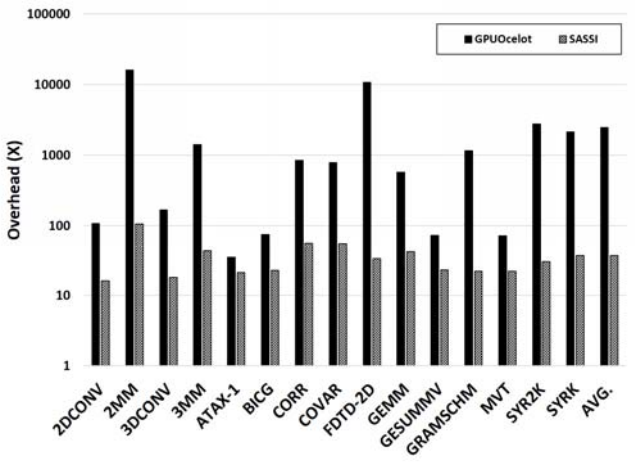

그림 4 메모리 정보 추출 오버헤드

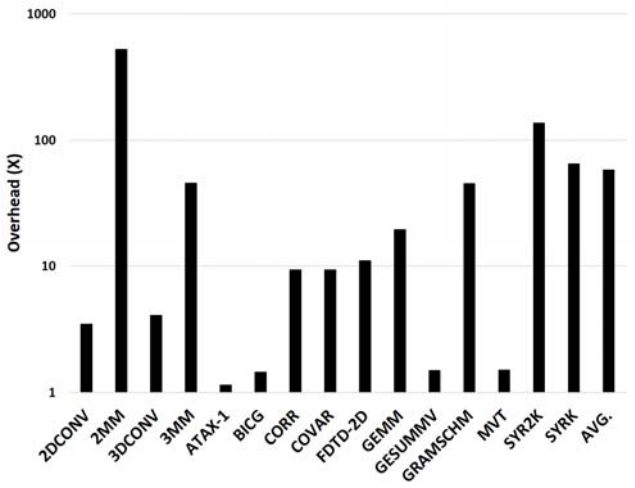

그림 5 워프 스케줄링 정보 추출 오버헤드

비로 나타내었다. 실험은 Polybench/GPU v1.0[5] 벤치마크 애플리케이션들을 대상으로 측정하였다.

먼저 쓰레드 단위의 메모리 정보 추출 도구의 오버헤드 비교를 위해 GPUOcelot에서 동일한 기능을 구현하였고, 이를 벤치마크 애플리케이션들을 대상으로 수행하였다. GPU 내 모든 쓰레드에서 요청하는 정보들을 추출하는 것은 많은 시간을 필요로 하기 때문에, 본 실험에서는 커널 내 하나의 TB가 요청하는 메모리 정보들만을 추출하도록 하였다.

그림 2는 SASSI와 GPUOcelot에서 메모리 정보를 추출하는 오버헤드를 비교한 그래프이다. SASSI와 GPUOcelot을 이용한 정보 추출 오버헤드는 각각 평균 (AVG.) 36.4배와 2,459.3배인 것으로 확인하였으며, SASSI를 이용한 메모리 정보 추출 오버헤드가 적은 것을 확인하였다.

그림 3은 SASSI로 구현한 워프 별 스케줄링 정보 추출 도구의 오버헤드를 그린 그래프로, 하나의 SM 내에서 스케줄링 된 모든 워프들의 시작과 종료 시간을 추출할 때의 오버헤드 결과이다. 실험한 결과, 평균 (AVG.) 58.5배의 오버헤드가 추가된 것을 확인할 수 있었다.

6. 결 론

대단위의 쓰레드가 병렬 연산을 수행하는 GPU를 최적화하기 위해서는 성능 분석이 가능한 도구가 필요하다. 따라서 본 논문에서는 상대적으로 적은 오버헤드로 실제 디바이스에서 프로파일링이 가능한 SASSI GPU 코드 삽입 도구를 이용하여 1) 쓰레드 단위의 메모리 접근 패턴 정보와 2) 워프 단위의 스케줄링 정보 추출 도구를 구현하였다. 두 도구에 대해서 프로파일 정보를 추출하지 않는 프로그램 실행 시간과 비교한 결과 각각 평균 36.4배와 58.5배의 오버헤드가 측정 되었다.

참고문헌

[1] Chao Li, Yi Yang, Zhen Lin, and Huiyang Zhou. 2015. Automatic data placement into GPU on-chip memory resources. In Proceedings of the 13th Annual IEEE/ACM International Symposium on Code Generation and Optimization (CGO), 2015.

[2] Mark Stephenson, Siva Kumar Sastry Hari, Yunsup Lee, Eiman Ebrahimi, Daniel R. Johnson, David Nellans, Mike O'Connor, and Stephen W. Keckler. Flexible software profiling of GPU architectures. In Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA), 2015.

[3] Naila Farooqui, Andrew Kerr, Gregory Diamos, S. Yalamanchili, and K. Schwan. A framework for dynamically instrumenting GPU compute applications within GPU Ocelot. In Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units (GPGPU), 2011.

[4] A. Bakhoda, G. L. Yuan, W. W. L. Fung, H. Wong and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," IEEE International Symposium on Performance Analysis of Systems and Software, Boston, MA, 2009, pp. 163-174.

[5] Scott Grauer-Gray, Lifan Xu, Robert Searles, Sudhee Ayalasomayajula, and John Cavazos. Auto-tuning a High-Level Language Targeted to GPU Codes. Proceedings of Innovative Parallel Computing (InPar), 2012.