CUDA 다중 프로세스 실행 서비스를 이용한 과학 응용 실행 패턴 분석

*김세진, 오지선, 김윤희
숙명여자대학교 컴퓨터과학과
{*wonder960702, jsoh8088}@gmail.com, yulan@sookmyung.ac.kr

Execution Pattern Analysis of Scientific Applications using CUDA Multi-Process Service

*Sejin Kim, Jisun Oh, Yoonhee Kim
Dept. of Computer Science, Sookmyung Women's University

요 약

Graphical Processing Units(GPUs)는 연산을 병렬적으로 처리함으로써 높은 성능을 사용자에게 제공해왔다. 기술의 발전에 따라 GPU의 하드웨어 자원이 증가함으로써, 하나의 응용만을 실행하면 GPU 자원을 모두 사용하지 못하는 문제가 생겨났다. 이로 인해 새로운 GPU 구조들은 커널의 동시 실행을 지원한다. 하지만 이는 같은 응용 안에서만 동시 실행이 가능하다는 문제점이 있어 NVIDIA는 Multi-Process Service(MPS)를 소개하였다. MPS는 다른 응용에 속한 커널도 동시 실행할 수 있도록 서비스하지만, 응용의 실행 특성이 미리 파악되어 있지 않으면 MPS 장점을 최대한으로 취할 수 없다. 본 논문에서는 응용 프로파일링을 통해 응용의 특성을 파악하고, 응용의 동시 실행을 통해 MPS에서의 커널 실행과 동시에 실행되는 응용의 조합의 중요성을 확인하였다. GPU 커널이 실행되는 비율이 높은 응용들을 동시 실행하여 여러 커널이 중첩 실행되었을 때, MPS를 사용하는 장점을 최대화할 수 있다.

Ⅰ. 서 론

Graphics Processing Units(GPUs)의 사용을 통해 다양한 응용들이 강력한 처리 성능을 받음으로써 여러 분야에서 이를 사용하는 것이 점점 더 보편화 되고 있다. 이는 응용의 연산 집약적인 부분을 병렬적으로 처리함으로써 처리 속도를 높일 수 있다.

특히 NVIDIA GPU의 Compute Unified Device Architecture(CUDA) 프로그래밍 모델은 커널(kernel)을 GPU에서 연산을 수행하기 위해 동작하는 함수로 정의한다. 이는 여러 개의 스레드가 프로그래머의 정의에 따라 스레드 블록을 이루며 각 스레드 블록은 하나의 Streaming Multiprocessor(SM)에서 실행된다. GPU는 수천 개의 코어에서 스레드를 병렬(parallel) 실행하여 응용들을 가속해왔다[5]. 하지만 GPU의 하드웨어 자원이 증가하면서 스레드 병렬성만으로는 GPU 자원들을 충분히 활용하지 못하는 문제점이 발생하였다[1].

이를 위해 최근의 GPU 구조들은 프로세스(응용)의 독립적인 커널들을 다른 스트림으로 지정함으로써 동시 실행(concurrent)을 제공한다. 하지만 이는 같은 프로세스 안에서만 동시 실행이 가능하다는 한계가 있다. 따라서 응용이 충분한 자원을 사용하지 않는다면 GPU의 자원 활용률이 떨어질 수 있다. 이의 한계를 해결하기 위해서 NVIDIA는 Multi-Process Service(MPS)를 제공한다. MPS는 다른 프로세스의 커널을 하나의 프로세스처럼 보이도록 함으로써 동시 수행이 가능하게 한다.

그러나 여러 커널의 실행 및 커널 특성이 미리 파악되어 있지 않으면 MPS의 장점을 최대한으로 활용할 수 없다. 그러므로 응용의 프로파일링을 통해서 응용의 성격을 파악하고 서비스를 이용하는 것이 중요하다. 본 연구에서는 MPS를 사용한 응용의 스케줄링 알고리즘을 개발하기 위해서 MPS를 사용한 CUDA 과학 응용의 실행 패턴을 분석하고자 한다. LAMMPS[3], GROMACS[4] 응용의 동시 실행을 통해 MPS의 커널 수행 방식과 동시에 실행되는 커널 간 조합의 중요성을 확인한다.

본 논문의 구성은 다음과 같다. 2장에서는 MPS의 구성과 기능을 설명한다. 3장에서는 과학 응용인 LAMMPS와 GROMACS의 실행을 통해 MPS의 성능 확인 및 분석을 진행하며, 4장에서는 결론 및 향후 연구에 관해 기술한다.

Ⅱ. Multi-Process Service

CUDA 응용은 사용할 하드웨어 자원이 담긴 CUDA context를 생성하며 시작된다. Hyper-Q 기술은 하나의 커널이 모든 GPU 자원을 사용하지 않는다면, 같은 CUDA context에 속한 다른 커널을 동시에 실행시킬 수 있도록 한다[6]. CUDA 모델에서 스레드 병렬성과 함께 동시에 실행될 수 있는 CUDA 작업은 커널, 디바이스-호스트간 메모리 복사 작업, CPU에서의 작업이다. Hyper-Q 기술을 가진 GPU의 동시 실행 스케쥴러(concurrent scheduler)는 이 작업들을 중첩시켜 같은 context에 속한 여러 프로세스가 GPU 자원을 동시에 활용할 수 있도록 한다. 동시 실행 스케쥴러는 커널과 디바이스-호스트 간 메모리 복사작업을 가지는 세 개의 작업이 제출되었을 때, 그림 1과 같이 커널과 메모리 복사 작업을 중첩한다. 하지만 다른 context에 속한 커널들은 time-sliced 스케줄러를 통해 그림 1과 같이 순차적으로 수행됨으로써 실행시간이 늘어남을 확인할 수 있다. 그러므로 하나의 context가 충분한 자원을 사용하지 않는다면 GPU의 자원이 낭비될 수 있다.

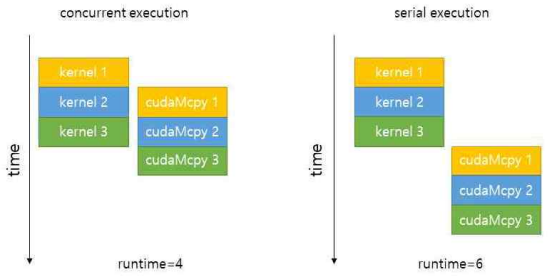

그림 1 동시 실행과 순차 실행

MPS는 다른 CUDA context로부터 온 여러 커널이 MPS 서버를 통해 작업을 제출함으로써 하나의 context로 관리되어 Hyper-Q 기술을 최대한으로 활용할 수 있도록 제공하는 CUDA API이다. GPU 남는 자원을 다른 프로세스의 커널이 사용할 수 있도록 하여 MPI(Message Passing Interface) 작업들을 실행하기에 적합하다[2]. MPS의 구조는 그림 2와 같이 서버 프로세스, 클라이언트 프로세스, MPS 컨트롤 데몬 프로세스로 구성되어 작동한다.

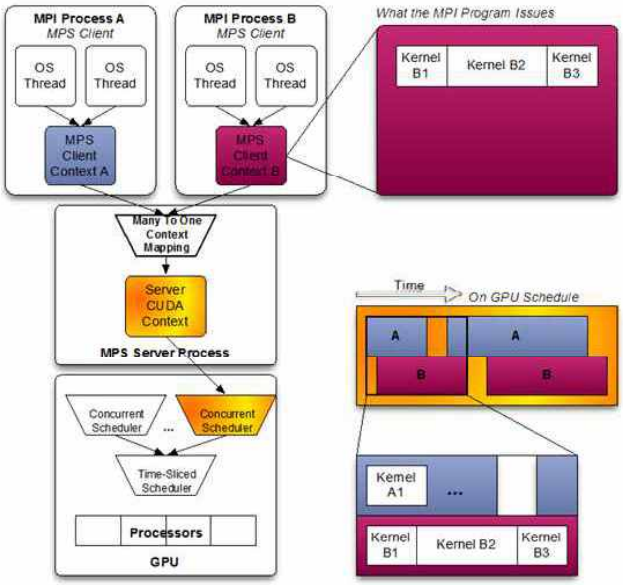

그림2 MPS 클라이언트-서버 구조[2]

단일 프로세스는 GPU에서 사용 가능한 자원을 모두 사용하지 않는다. MPS는 서로 다른 프로세스(응용)의 커널과 호스트와 디바이스간의 메모리 복사 작업을 중첩해서 시간을 단축하고, GPU 활용률을 높일 수 있다. GPU는 각 프로세스에 저장공간과 스케줄링 자원을 할당한다. 따라서 여러 프로세스가 GPU에서 실행되면 스케줄링 자원이 GPU에서 스와핑되어야 한다. 하지만 MPS 서버는 모든 MPS 클라이언트가 GPU의 저장공간과 스케줄링 자원을 공유하게 하므로 클라이언트를 스케줄링할 때 스와핑 오버헤드를 제거한다.

Ⅲ. 실험 및 분석

본 논문에서는 MPS의 프로세스 실행 방식의 진행과 성능을 확인하기 위해 과학 계산 응용인 LAMMPS[3], GROMACS[4] 응용을 대상으로 실행하였다.

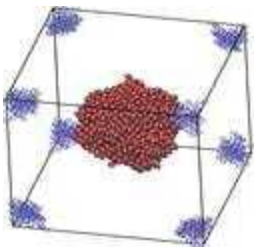

그림 3-(a) LAMMPS[3]

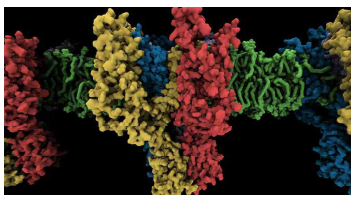

그림 3-(b) GROMACS[7]

그림 3-(a)와 같이 LAMMPS 응용은 분자 동역학 시뮬레이션을 위한 소프트웨어이며, 그림 3-(b)와 같이 GROMACS는 물 분자 데이터 셋을 사용하여 뉴턴의 운동 방정식을 시뮬레이션 한다. 본 실험 환경은 Intel(R) Core(TM) i7-5820K, Nvidia Titan XP이며 Pascal 구조이다. 각 응용 프로세스의 커널 실행 흐름을 파악하기 위해 Nvidia visual profiler(nvprof)를 사용하여 확인하였다.

LAMMPS 응용의 총 수행시간은 1196초이며 수행 동안 363MB~8.3GB의 메모리를 사용한다. 이 응용은 호스트와 디바이스간 메모리 복사가 빈번하게 일어나서 GPU 커널이 실행되는 비율을 나타내는 GPU의 평균 활용도(utilization)가 약 25%이다. GROMACS 응용의 경우, 총 수행시간은 636초이며 수행 동안 171MB~537MB의 메모리를 사용한다. 이 응용은 수행시간 동안 빈번히 연산이 실행되어 평균 활용도(utilization)가 69%를 보이며 계산 집약적인 형태를 확인할 수 있다.

MPS를 사용하여 LAMMPS와 GROMACS를 단독적으로 수행하였을 때 수행시간은 각각 1208초, 641초였다. MPS를 사용하지 않았을 때와 비교하여 수행시간이 각각 12초, 5초 늘어난 것을 확인하였다. 이는 MPS 서버와 클라이언트 간의 통신 오버헤드로 인하여 수행시간이 증가한 것임을 알 수 있다.

MPS를 사용한 응용의 동시 실행을 통해 MPS의 커널 수행 방식을 확인하기 위해, 동시 작업을 LAMMPS-GROMACS, GROMACS-GROMACS로 구성하여 실험하였다. LAMMPS-GROMACS 작업의 경우 수행시간은 MPS를 사용하지 않고 동시에 실행했을 때의 시간은 833초, MPS를 사용하였을 때의 시간은 809초이다. 이때, 본 논문은 커널이 동시에 실행된 경우의 커널 수행 방식에 초점을 두고 있으므로 두 응용이 동시에 실행되는 시간만 고려하였다. 두 응용의 수행시간이 다르므로 하나의 응용의 수행이 끝나고 다른 응용이 단독으로 수행되는 시간은 고려하지 않았다.

그림 4-(a)는 MPS를 사용하지 않았을 경우의 LAMMPS 응용과 GROMACS 응용의 커널 실행(파란색 계열 작업)과 CPU-GPU 메모리 전송 작업(황토색 작업)을 343초부터 344.5초까지 보여주는 타임라인이다. 두 프로세스는 다른 CUDA context를 생성하여 스케줄링 자원을 할당한다. 343.7초부터 343.9초까지 LAMMPS의 메모리 작업이 실행되는 동안, GROMACS의 커널이 약 155밀리 초 중첩되는 등 GPU의 효율성을 높일 수 있었다. 그러나 343.1초, 343.4초에는 커널과 메모리 전송 작업이 중첩되지 않고, 각 응용의 커널들이 함께 제출된다. 이때. time sliced 스케줄러가 커널들을 시간 단위로 나누어 순차적으로 실행시키므로 문맥교환의 오버헤드가 발생하며 GPU 자원 사용의 효율성이 낮다.

파스칼 구조의 NVIDIA GPU는 명령어 레벨로 문맥 교환이 이루어지기 때문에 블록 단위로 보여주는 nvprof에서는 커널의 문맥 교환을 확인할 수 없다.

그림 4-(b)는 325초부터 326.75초까지 각 응용이 MPS를 사용하여 하나의 CUDA context로 통합되어 두 개의 stream으로 실행되는 타임라인이다. 325.4초부터 325.6초까지의 LAMMPS 메모리 복사 작업이 약 141 밀리 초 동안 GROMACS의 커널과 중첩될 뿐 아니라, 325.7초부터 약 0.1초 간격으로 약 326초까지 LAMMPS의 연산 작업과 GROMACS의 메모리 전송 작업이 중첩되어 효율성을 높일 수 있었다. 또한, GROMACS의 커널에서 LAMMPS의 커널로 문맥이 변경될 때, 스케줄링 자원이 GPU에서 스왑 인, 아웃 되지 않으므로 문맥 교환의 오버헤드를 제거할 수 있다.

MPS를 사용하지만 수행시간에서의 차이는 거의 발생하지 않는데, 이는 LAMMPS와 GROMACS 응용의 커널이 동시에 실행되는 구간이 많이 존재하지 않아 문맥 교환의 오버헤드가 많이 발생하지 않기 때문이다. 또한, 이 두 응용은 MPS를 사용하지 않아도 커널과 CPU-GPU 메모리 복사 작업을 중첩하여 수행시간이 짧고 GPU 활용도가 높다. GROMACS-GROMACS 작업의 경우 MPS를 사용하지 않고 실행했을 때 1391초, MPS를 사용하였을 때는 1233초 수행시간이 소요된다.

그림 4-(c)는 MPS를 사용하지 않았을 때 LAMMPS-GROMACS 실행을 179.5초부터 180.15초까지 프로파일링한 타임라인이다. 두 프로세스는 다른 CUDA context에 속하여 실행된다. 커널(하늘색 작업)들이 동시에 제출이 되어 time sliced 스케줄러가 시간을 분할하여 순차실행이 되고, 문맥이 교환될 때 스케줄링 자원이 GPU에서 스와핑된다. 호스트-디바이스간 메모리 작업(황토색 작업)과 커널이 중첩되어 실행되기보다는, 커널이 동시 실행되므로 GPU 자원을 두고 경쟁이 일어나 효율성이 떨어지며 수행시간도 길어진다.

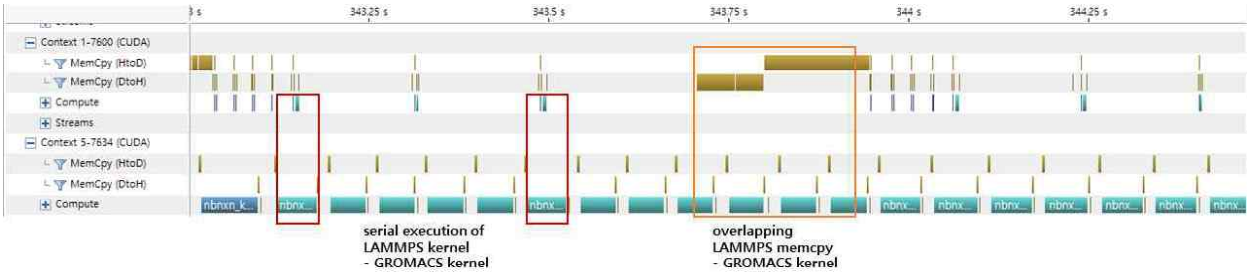

그림 4-(a) MPS를 사용하지 않은 LAMMPS-GROMACS 작업의 수행 프로파일

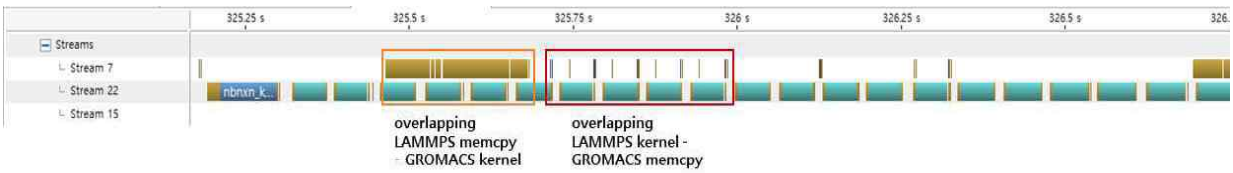

그림 4-(b) MPS를 사용한 LAMMPS-GROMACS 작업의 수행 프로파일

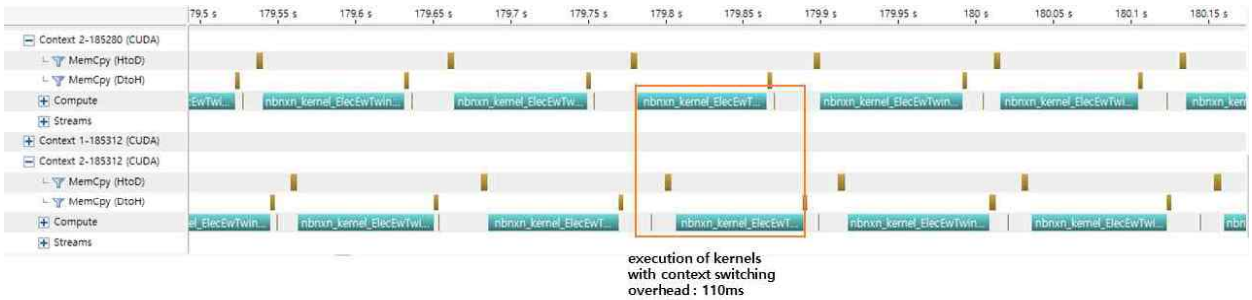

그림 4-(c) MPS를 사용하지 않은 GROMACS-GROMACS 작업의 수행 프로파일

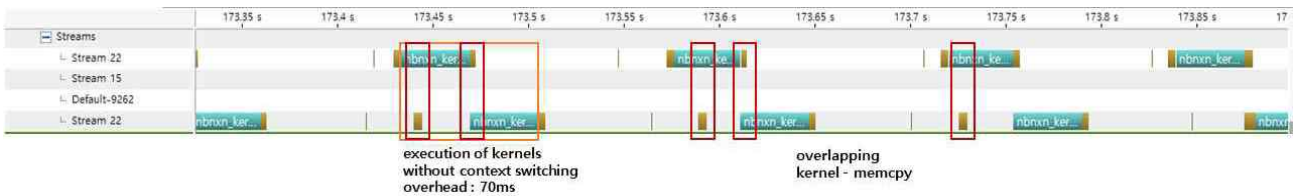

그림 4-(d) MPS를 사용한 GROMACS-GROMACS 작업의 수행 프로파일

그림 4-(d)는 MPS를 통해 두 응용을 수행했을 때 173.3초부터 173.9초까지의 타임라인을 나타낸다. 173.43초부터 약 0.15초 간격으로 두 번째로 제출한 프로세스의 메모리 복사 작업과 첫 번째로 제출한 프로세스의 커널이 중첩되며, 173.46초부터 약 0.15초 간격으로 첫 번째 프로세스의

메모리 복사 작업과 두 번째 프로세스의 커널이 중첩되는 것을 확인할 수 있다. 또한, MPS를 사용하여 각 프로세스의 두 개의 커널을 실행하였을 때는 70밀리 초가 소요되었지만, MPS를 사용하지 않았을 때는 같은 커널의 실행에 대해 110밀리 초가 소요되었다. 이를 통해 MPS는 스케줄링 자원의 GPU 스와핑을 제거하여 문맥 교환의 오버헤드를 축소하고, 커널과 메모리 복사 작업을 중첩하여 수행시간을 단축할 수 있다는 것을 알 수 있다. 따라서 MPS 서버 사용의 장점은 여러 커널이 중첩되는 프로세스일 때 나타나는 것으로 확인할 수 있다.

Ⅳ. 결 론

MPS를 사용해서 여러 응용 및 프로세스들을 동시에 수행할 수 있으며 Hyper-Q 기술을 최대한 활용할 수 있다. 그러므로 이는 GPU 자원 활용률을 높이고, 시간을 단축시킬 수 있다. 본 논문에서 진행한 실험을 통해 여러 커널이 중첩되어 사용되었을 때 MPS를 사용하는 장점을 얻을 수 있다는 것을 확인할 수 있었다. 응용 특성을 파악하여 실험하게 되면 MPS 사용의 장점을 극대화 시킬 수 있으므로 응용의 프로파일링 정보를 미리 파악하는 것이 중요하다.
향후 연구로는 GPU CUDA 응용의 커널 실행 패턴을 파악하여, 프로파일링 정보를 바탕으로 MPS 서버에서 서로 영향을 주는 응용들의 커널 스케줄링 기법에 관해 연구하고자 한다.

ACKNOWLEDGMENT

이 논문은 2015년도 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(NRF-2015M3C4A7065646)

참 고 문 헌

[1] Carvalho, Pablo, et al. "Kernel concurrency opportunities based on GPU benchmarks characterization." Cluster Computing (2019): 1-12.

[2] Multi-Process Service, https://docs.nvidia.com/deploy/mps/index.html

[3] LAMMPS, https://lammps.sandia.gov/.

[4] GROMACS, http://www.gromacs.org/

[5] CUDA C Programming Guide, https://docs.nvidia.com/cuda/cuda-c-programming-guide/

[6] HyperQ, http://developer.download.nvidia.com/compute/DevZone/C/html_x64/6_Advanced/simpleHyperQ/doc/HyperQ.pdf

[7] GPU-accelerated GROMACS, https://www.nvidia.com/en-us/data-center/gpu-accelerated-applications/gromacs/