CUDA programming environment을 활용한 Path-Integral Monte Carlo Simulation의 구현

이화영 * 임은진**

(Hwayoung Lee *, Eun-Jin Im**)

요 약 높아지는 Graphic Processing Unit (GPU)의 연산 성능과 GPU에서의 범용 프로그래밍을 위한 개발 환경의 개발, 보급으로 인해 GPU를 일반연산에 활용하는 연구가 활발히 진행되고 있다. 이와 같이 일반 연산에 활용되고 있는 GPU 로 nVidia Tesla 와 AMD/ATI 의 FireStream 들이 있다. 특수 목적 연산 장치인 GPU를 일반 연산을 위해 프로그래밍하기 위해서는 그에 맞는 프로그램 개발 환경이 필요한데 nVidia에서 개발한 CUDA (Compute Unified Device Architecture) 환경은 자사의 GPU 프로그램 개발을 위해 제공되는 개발 환경이다. CUDA 개발 환경은 nVidia GPU 프로그래밍 뿐만 아니라 차세대 이종 병렬 프로그램 개발 환경의 공개 표준으로 논의되고 있는 OpenCL (Open Computing Language) 와 유사한 특징을 보일 것으로 예상되기 때문에 그 중요성은 특정 GPU 에만 국한되지 않는다. 본 논문에서는 경로 적분 몬테 카를로 (Path Integral Monte Carlo) 방법을 CUDA 개발 환경을 사용하여 nVidia GPU 상에서 병렬화한 결과를 제시하였다.

핵심주제어 : 그래픽 프로세싱 유닛(GPU), CUDA, 병렬 프로그램, 경로적분 몬테카를로 방법

Key Words : GPU, CUDA, parallel program, Path-Integral Monte Carlo Method

Ⅰ. 서 론

높아지는 Graphic Processing Unit (GPU)의 연산 성능과 GPU에서의 범용 프로그래밍을 위한 개발 환경의 개발, 보급으로 인해 GPU를 일반연산에 활용하는 연구가 활발히 진행되고 있다.[1] 이와 같이 일반 연산에 활용되고 있는 GPU 로 nVidia Tesla[2] 와 AMD/ATI 의 FireStream[3], Intel 의 Larrabee[4] 들이 있다. 이와 같은 추세는 Sony/Toshiba/IBM 이 공동 개발하여 PlayStation3에 널리 사용되고 있는 Cell[5] type 프로세서와 더불어 이종 병렬 프로그램 (heterogeneous parallel system)으로 구성되는 미래의 컴퓨팅 환경을 제시하고 있다. 따라서 빠른 속도로 개발되는 병렬 하드웨어를 수용할 수 있는 소프트웨어 개발 환경의 필요성과 중요성은 이전보다 더 중요해지고 있는 것이다. 이러한 이종 병렬 프로그램 개발 환경의 한 예로써 nVidia에서 제공하는 CUDA (Compute Unified Device Architecture) 개발 환경[6]을 들 수 있다. CUDA 개발 환경은 nVidia GPU 프로그래밍 뿐만 아니라 차세대 이종 병렬 프로그램 개발 환경의 공개 표준으로 논의되고 있는 OpenCL (Open Computing Language)[7]과 유사한 특징을 보일 것으로 예상되기 때문에 그 중요성은 특정 GPU 에만 국한되지 않는다. 그 외에 GPU 개발 환경으로는 AMD/ATI GPU 프로그램 개발 환경으로 제공되는 Stream SDK[8]가 있다. 여기에는 공개된 고레벨 언어인 Brook+[9]을 이용하거나 CAL(Compute Abstraction Layer)[8]이 포함되어 있다. Brook+은 C와 비슷한 프로그래밍

* 국민대학교 컴퓨터공학과 석사과정
** 국민대학교 컴퓨터공학과 교수 (교신저자)

환경을 제공해줌으로써 좀더 쉽고 빠르게 개발할 수 있다. 반면에 CAL은 Brook+보다 최적화에 장점을 가지지만 하드웨어 기반 지식이 필요하다.

몬테 카를로 시뮬레이션은 대표적인 embarrassingly parallel application 으로써 multi-core GPU 상에서 병렬화하기에 적합한 응용으로 이미 GPU 를 이용한 여러 종류의 몬테 카를로 시뮬레이션 적용 연구가 보고되고 있다 [10]. 본 연구자는 고체 수소의 양자적 회전자들의 열역학적 성질을 규명하기 위해 사용되는 경로 적분 몬테 카를로 (Path Integral Monte Carlo) 방법[11]을 GPU 상에서 병렬화하여 연산을 가속화하는 연구를 행하고 있다. 이 방법은, 다중의 샘플에 대하여 서로 독립적인 단순한 연산을 하는 대부분의 Monte Carlo 방법과 달리 각각의 샘플에 대한 연산량이 대단히 크기 때문에 각 샘플 내부의 연산 자체를 병렬화할 필요가 있다는 점에서, 연산량과 병렬화 방법에 있어서 근본적인 차이가 있다. 본 논문에서는 경로 적분 몬테 카를로 (Path Integral Monte Carlo) 방법을 CUDA 개발 환경을 사용하여 nVidia GPU 상에서 병렬화한 결과를 제시하였다.

II. nVidia GPU

nVidia GPU 들은 GeForce, Quadro 와 일반 연산 전용의 Tesla 시리즈가 있다.

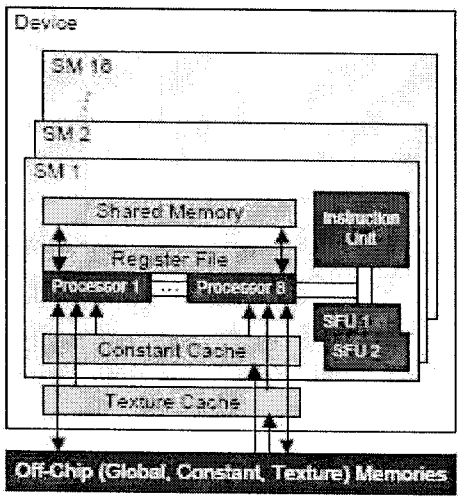

그림 1. nVidia GPU 구조 [2]

그림 1은 Ryoo et. al.[2] 이 도시한 nVidia GPU 의 구조이다. 그림에서 보는 바와 같이 GPU 내부에는 다수개의 스트림 멀티프로세서가 있으며 각 스트림 멀티프로세서에는 보통 8개의 스트림 멀티프로세서 (코어)들이 있어서 32bit 부동 소수점 및 정수 연산 장치를 통해 SIMD(Single-Instruction Multiple-Data) 형식으로 쓰레드들을 동시 수행하며 이들은 스트림 멀티프로세서 내에서 고속의 레지스터 파일과 공유 메모리, constant 및 texture cache, 그리고 스페셜 펑셔널 유니트(SFU) 들을 공유한다. SFU는 삼각함수나 제곱근 계산과 같이 복잡한 부동소수점 연산을 행하는 장치이다.

GPU 사용에 있어서 효율적인 메모리 사용이 중요한데 제한된 on-chip memory를 다수의 쓰레드들 간에 공유해야 하기 때문이다. 더 용량이 큰 off-chip 의 디바이스 메모리는 글로벌 메모리, 로컬 메모리, constant 및 texture 메모리로 사용될 수 있는데 접근 속도가 많이 느리다. [2]

constant memory와 texture memory는 read-only memory로써 device 상에선 데이터를 쓸 수 없다. 또한 이 두 메모리에는 cache가 존재하여 접근 속도를 향상시킨다.

III. CUDA 프로그램 개발 환경

CUDA (Compute Unified Device Architecture)[6] 는 nVidia GPU를 그래픽 API를 통하지 않고 data parallel 프로그램하기 위한 소프트웨어 구조를 의미한다. CUDA 언어는 C 언어를 확장한 것으로 그림 2에서 보는 바와 같이 CUDA 프로그램 개발 환경은 GPU 디바이스 드라이버와, 컴파일러 및 런타임 시스템과 라이브러리로 이루어져 있다.

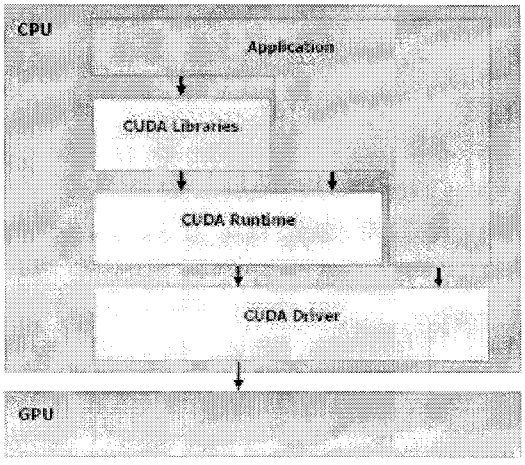

그림 2 CUDA 소프트웨어 스택[6]

CUDA 프로그램은 kernel 함수라고 불리우는 함수를 GPU 상에서 수행하도록 스케쥴하며 이 커널 함수가 호출될 때 인자를 사용하여 스케쥴할 쓰레드 개수를 정할 수 있고 이 쓰레드 개수는 편의에 따라 3차원으로 정의할 수 있다. 예를 들면 다음과 같은 CUDA 문장을 이용하여 kernel 함수가 호출된다.

foo <<< Dg, Db, Ns S>>> (arg1)

CUDA thread들은 계층구조를 이룬다. 여러 개의 쓰레드들이 모여서 하나의 쓰레드 블록을 형성하고, 다수개의 쓰레드 블록들은 다시 쓰레드 그리드를 형성한다. GPU에서 쓰레드들은 쓰레드 블록 단위로 스트림 멀티프로세서에 할당된다. 위의 문장에서 Dg 는 쓰레드 그리드의 크기를 (x,y,z) 의 형태로 최대 3차원으로 나타내고, Db 는 쓰레드 블록의 크기를 역시 최대 3차원으로 나타내며, Ns 는 쓰레드 블록에 동적으로 할당되는 공유 메모리의 크기를 바이트 단위로 나타내며 S 는 optional argument 로 특정 스트림에 연관시키고자 할 때 쓰인다.

함수들을 커널 함수로 선언하기 위해서는 __device__ (커널 함수에서만 호출 가능) 와 __global__ (CPU함수에 호출 가능) 지시자를 사용하고, 변수들을 다양한 종류의 메모리에 할당하기 위해서는 __shared__, __constqant__, __device__ 지시자들을 사용한다. kernel 함수는 nVidia C 컴파일러에 의해 PTX 어셈블리 코드라는 중간 코드로 생성되며 CUDA 런타임 시스템에 의해 수행된다.

IV. 경로 적분 몬테 카를로 방법 (PIMC)

대표적인 양자 고체인 고체 수소에서 양자적 회전자들의 열역학적 성질을 계산하기 위해 사용되는 경로 적분 몬테 카를로 방법[11]은 양자 입자들의 경로 적분이 상호작용하는 고전적인 고분자로 일대일 대응될 수 있다는 Feynman의 아이디어에 바탕을 둔 방법으로 그동안 양자 다체계의 물리적 성질을 정확히 이해하고 예측하는데 있어 주목할 만한 성과를 보였다.

Monte Carlo method 란 "반복적인 무작위 (random) 변수에 대한 샘플링에 의해 결과를 계산하는 알고리즘"을 총칭하는 말로써 무작위 샘플들에 대한 각각의 연산이 독립적이기 때문에 대표적인 embarrassingly parallel application 으로 널리 알려져 있으며 multi-core GPU 상에서 병렬화하기에 적합한 응용으로 이미 GPU를 이용한 여러 종류의 몬테 카를로 시뮬레이션 적용 연구가 보고[10]되고 있다. 그러나 경로 적분 몬테 카를로 시뮬레이션은, 다중의 샘플에 대하여 서로 독립적인 단순한 연산을 하는 대부분의 Monte Carlo 방법과 달리 각각의 샘플에 대한 연산량이 대단히 크기 때문에 각 샘플 내부의 연산 자체를 병렬화할 필요가 있다는 점에서, 연산량과 병렬화 방법에 있어서 근본적인 차이가 있다. 본 연구에서는 이러한 경로 적분 몬테 카를로 방법을 nVidia GPU 상에서 병렬화하여 연산을 가속화하는 연구를 수행하였다.

IV. PIMC 의 구현

본 연구에서는 경로 적분 몬테 카를로 방법의 (1) Fortran 순차 코드를 먼저 C언어의 순차코드로 이식하고 (2) 이 코드를 CUDA 환경을 이용하여 병렬화하였다.

```
function sweep(axis)

for ( j = 0; j < nslice; j++ )
{
  for ( i = 0; i < nparts; i++ )
  {
    Generate the trial move in axis[j][i]
    Calculate the probability to accept the trial move
  }
}
```

그림 3. PIMC 순차코드에서의 sweep함수

그림 3은 PIMC코드에서 병렬화하고자 하는 sweep 함수에 대한 pseudo code이다. sweep 함수는 확률에 따른 회전자의 움직임을 나타낸다. sweep 함수는 각 iteration마다 의존성이 없기 때문에 병렬처리가 가능하다. sweep 함수를 병렬코드로 구현을 위해 각 thread가 담당하는 nslice의 구간을 구한다. 이때 회전자 움직임에 연속성을 위해서 각 구간들은 겹치도록 하고, 겹치는 부분은 계산을 하지 않는다. 그림 4는 각각의 thread가 4개의 slice을 담당하는 경우를 나타내고. 가로 안의 수는 처리하는 thread id을 뜻한다.

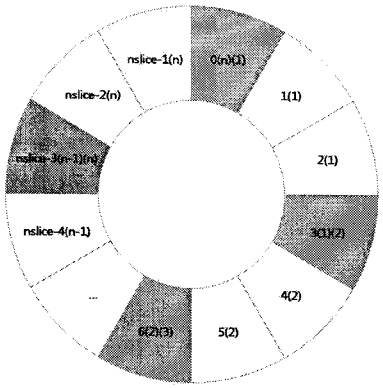

그림 4 PIMC 병렬코드에서 nslice 분할

겹치는 부분의 계산을 위해서 다음 sweep 함수를 호출할 때 각각의 thread가 담당하는 slice 구간을 1칸씩 이동하여 계산한다.

VI. 결 론

본 논문에서 경로 적분 몬테 카를로 방법을 CUDA 환경을 사용하여 병렬화한 결과, C언어로 이식된 순차 코드의 경우 3GHz Intel Core2 CPU에서 49000초가 걸리던 수행 시간이 CUDA 환경을 이용해 병렬화된 코드를 nVidia C870 Tesla에서 수행한 결과 3480초로 단축하여 14배의 speedup을 얻었다.

PIMC 의 병렬화 연구는 CUDA 코드를 fine-tuning하여 더 높은 성능을 얻을 수 있을 것으로 기대된다. 현재 구현된 sweep 함수에서는 회전자 axis의 상태 벡터를 update 하는데 이 axis 의 상태 벡터를 글로벌 메모리에 저장하였으므로 고속의 on-chip 메모리를 사용하여 속도 향상을 도모할 수 있다.

참 고 문 헌

[1] John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn, and Tim Purcell, "A Survey of General-Purpose Computation on Graphics Hardware", Computer Graphics Forum, 26(1):80-113, March 2007.

[2] Shane Ryoo, Christopher Rodrigues, Sara Baghsorkhi, Sam Stone, David Kirk, Wen-mei Hwu, "Optimization Principles and Application Performance Evaluation Of a Multithreaded GPU Using CUDA", Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 73-82, February, 2008.

[3] "AMD stream computing whitepaper", http://ati.amd.com/technology/streamcomputing/firestream-sdk-whitepaper.pdf, Nov. 2007.

[4] Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Abrash, M., Dubey, P., Junkins, S., Lake, A., Sugerman, J., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Hanrahan, P. 2008. "Larrabee: A Many-Core x86 Architecture for Visual Computing". ACM Trans. Graph. 27, 3, Article 18, August 2008.

[5] Michael Gschwind, H. Peter Hofstee, Brian Flachs, Martin Hopkins, Yukio Watanabe and Takeshi Yamazaki. Synergistic processing in Cell's multicore architecture. IEEE Micro 26(2):10-24, March 2006.

[6] CUDA 2.1 Programming Guide", http://developer.download.nvidia.com/compute/cuda/2_1/toolkit/docs/nVidia_CUDA_Programming_Guide_2.1.pdf

[7] "The OpenCL Specification", http://www.khronos.org/registry/cl/specs/opencl-1.0.33.pdf, Feb. 2009.

[8] "ATI Stream SDK User Guide", http://developer.amd.com/gpu_assets/Stream_Computing_User_Guide.pdf

[9] Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston, and Pat Hanrahan, "Brook for GPUs: Stream Computing on Graphics Hardware", In Proceedings of International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH2004) pp. 777-786, August 2004

[10] Roger D. Chamberlain, Joseph M. Lancaster, and Ron K. Cytron, "Visions for Application Development on Hybrid Computing Systems," Parallel Computing, 34(4-5):201-216, May 2008.

[11] Bernard Bernu and David M. Ceperley, "Path Integral Monte Carlo", in Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms, John von Neumann Institute for Computing, J¨ ulich, NIC Series, Vol. 10, ISBN 3-00-009057-6, pp. 51-61, 2002.