한국컴퓨터정보학회 동계학술대회 논문집 제21권 제1호 (2013. 1)

CUDA로 구현한 FDTD알고리즘의 OpenMP기술 적용 및 성능 측정

정복재°, 오승택*, 이철훈*
°*충남대학교 컴퓨터공학과
e-mail: {jbj12, ost2011, clee}@cnu.ac.kr

OpenMP application to implement CUDA for FDTD algorithm and performance measurement

Bok-Jae Jung°, Seung-Take Oh*, Cheol-Hoon Lee*
*Chung-Nam National University

● 요 약 ●

반도체 공정에서 소자의 제조 비용 감소를 위해 제조 공정 검증을 위한 시뮬레이션을 수행하게 된다. 이 시뮬레이션은 반도체 소자 내부의 물리량 계산을 통해 반도체 소자 내부의 불순물의 거동을 해석하게 된다. 이를 위해 사용되는 알고리즘으로 3차원적 형상을 표현하는 물리적 미분 미분방정식을 계산하게 되는데, 정확한 계산을 위해 유한 차분 시간 영역법(이하 FDTD)과 같은 수치해석 기법을 이용한다. 실제적으로 반도체 공정의 시뮬레이션에서 FDTD연산의 실행 시간은 90% 이상을 소요하게 된다. 이러한 연산에서 더욱 빠른 성능을 확보하기 위해 본 논문에서는 기존의 CUDA(Compute Unified Device Architecture)로 구현된 FDTD알고리즘을 OpenMP를 통한 다중 GPU제어를 이용하여 연산 수행시간을 감소하고, 그 결과물을 통하여 성능 향상도를 측정한다.

키워드: FDTD, CUDA, OpenMP

I. 서 론

컴퓨터의 발달 및 사용 목적이 다양해짐에 따라, 어플리케이션에서 처리하는 데이터의 양과 연산 대상들이 점차적으로 증가되고 있는 추세이다. 이에 따라 프로그램 수행 방식 또한 대용량 데이터를 처리하기 위한 기존의 순차 프로그래밍에서 병렬 프로그래밍으로 발전하고 있다.

병렬 프로그래밍은 멀티코어를 시작으로 오늘날의 GPGPU (General-Purpose computing Graphics Processing Units)에 이르기까지 계속적으로 발전하고 있다. GPGPU는 GPU장치를 이용하여 동시에 많은 양의 연산을 수행하는 기술로써, 오늘날 Scalar 연산 병렬 프로그래밍 모델에서 가장 좋은 성능을 보이는 것으로 평가 받고 있으며 기상관측 시스템을 비롯하여 각종 시뮬레이션을 수행하는 고성능 컴퓨팅 환경에서 사용되고 있다. 본 논문에서 구현하고자 하는 FDTD 알고리즘 또한 GPGPU의 활용도가 높은 연산으로써, 반도체 공정의 모의 시뮬레이션을 수행한다. 하지만, GPU에서 가질 수 있는 제한된 자원에 의해 GPGPU단일 기술만으로는 빠른 연산 속도를 기대하기 힘들고 연산을 수행하는 데이터 크기 또한 많은 양의 크기를 처리하지 못하는 문제가 발생하게 된다. 본 논문에서는 이러한 문제를 해결하기 위해 GPGPU의 한 종류로서 NVIDIA사에서 발표한 CUDA와 OpenMP를 이용하여 FDTD 알고리즘을 수행하고, 구현된 결과물을 통해 향상된 성능을 증명한다.

본 논문은 2장에서 관련연구로써 FDTD, CUDA, OpenMP에 대해 소개하고, 3장에서는 OpenMP를 이용한 FDTD알고리즘의 구현을, 4장에서는 테스트 환경과 결과를 보이며, 마지막으로 5장에서는 결론 및 향후 연구과제를 기술한다.

II. 관련 연구

1. FDTD

1966년 Yee가 처음으로 제안한 FDTD는 맥스웰(Maxwel) 방정식을 시간영역 상에서 수치 해석적으로 풀어내는 방법이다. 알고리즘이 간단하며 전파해석과 관련된 다양한 문제에 적용할 수 있는 유연성 때문에 FDTD는 전자기파 산란 해석에 광범위하게 쓰인다. FDTD 방법은 간단한 수치해석으로써 임의의 구조를 갖는 물체의 마이크로파 해석에 유용하고 복잡한 구조를 가지는 광소자의 마이크로파 해석에 적합하다. 또한 Fourier 변환에 의하여 간단히 주파수 영역에서의 변수들을 계산할 수 있다[1].

2. CUDA

CUDA는 GPU에서 수행하는 병렬 처리 알고리즘을 C 프로그래밍 언어를 비롯한 프로그래밍 언어를 사용하여 작성할 수 있도록 하는 GPGPU기술이다. CUDA는 C 프로그래밍 언어를 기본으로 별도의 라이브러리를 추가하여 사용하는 방식이다. 그래픽스 파이프라인의 별도의 이해가 필요없이 C/C++언어를 사용하여 프로그램을 개발할 수 있는 환경을 제공하기 때문에 프로그래머가 보다 손쉽게 병렬처리 프로그램을 개발할 수 있다. CUDA는 2006년 10월 NVIDIA가 웹에 처음 공개하였으며, 이후 지속적으로 발전하고 있으며 슈퍼컴퓨터 제작과 고성능 연산처리에 많이 사용되고 있다. 특히 데이터 사용에 초점을 둔 어플리케이션을 CUDA를 통하여 병렬화 한 뒤, GPU 내부의 코어 프로세서를 사용하여 고속의 부동소수점 연산을 할 수 있다. 이러한 CUDA 아키텍처를 사용하려면 NVIDIA GPU와 특별한 스트림 처리 드라이버가 필요시 된다.[2].

2.1 Hardware Architecture

아래의 <그림 1>은 CUDA의 하드웨어 구조를 설명하는 그림이다. GPU내부는 대량의 코어들이 존재하여 병렬처리 수행 시 코어당 하나의 쓰레드를 처리하게 된다. 쓰레드들이 모여 하나의 블록을 이루며, 블록들이 모여 그리드를 이루는 계층 구조를 가진다. 블록은 하드웨어적으로 하나의 SM에서 처리하게 되고 그리드별로 GPU를 선점하여 연산을 수행한다. 각 SM(Streaming Multiprocessor)은 Core뿐만 아니라, sin이나 exp와 같은 초월 함수를 위해 SFU(Special Function Unit)가 존재하고 빠른 메모리 접근을 위해 Shared Memory, L1 Cache, Register를 SM별로 가진다.

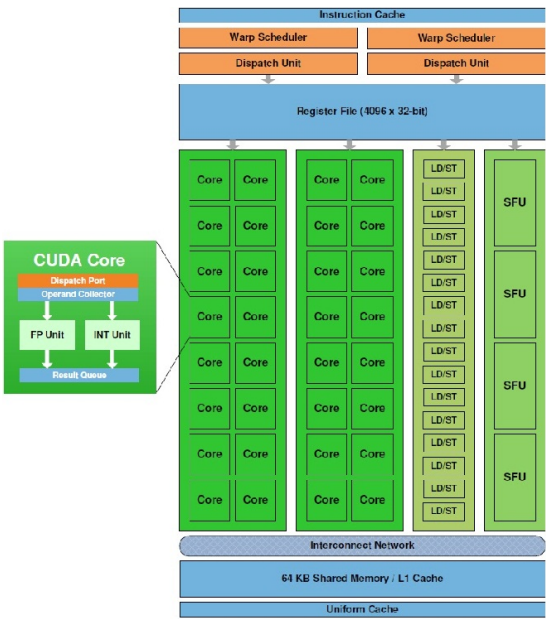

그림 1. CUDA 하드웨어 아키텍처
Fig. 1. CUDA Hardware Architecture

2.2 Memory Architecture

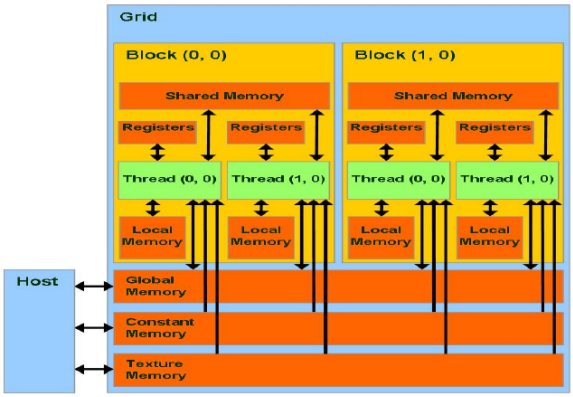

그림 2. CUDA 메모리 아키텍처
Fig. 2. CUDA Memory Architecture

CUDA는 GPU의 메모리를 크게 GPU 내부의 온칩(on-chip)과 GPU 외부의 오프칩(off-chip)으로 나누어서 관리한다. 온칩 메모리에는 레지스터 메모리, 공유 메모리가 있고, 오프칩 메모리에는 전역 메모리, 지역 메모리, 텍스처 메모리, 상수 메모리가 있다. <그림 2>는 CUDA 메모리 구조와 각각의 메모리간의 연산관계를 나타낸 그림이다[3].

3. OpenMP

멀티코어의 장점을 살리기 위해서는 소프트웨어 또한 멀티코어를 지원할 수 있어야한다. 순차코드로 작성된 프로그램으로는 멀티코어라 하여도 병렬로 처리할 수 없기 때문에 멀티코어로 인한 성능향상을 기대하기 어렵다. 따라서 멀티코어를 위한 프로그램이 작성되어야 하는데 알고리즘 복잡성 증가로 인해 이는 매우 어려우며 또한 이미 개발 되어있는 많은 순차 프로그램을 병렬 프로그램으로 수정 하는 것은 많은 시간을 필요로 한다. 따라서 이를 좀 더 쉽게 할 수 있도록 OpenMP가 나오게 되었다. OpenMP는 공유 메모리 형태 멀티프로세서의 병렬 프로그래밍 모델을 위한 API(Application Programming Interface)이다[4]. OpenMP는 SMP(Symmetric MultiProcessing)타입의 컴퓨터 아키텍처에서 C, C++, Fortran 언어를 지원하며, 어플리케이션이 이용할 수 있도록 컴파일러 디렉티브를 삽입함으로써 서브루틴 라이브러리를 덧붙여 병렬 처리가 가능하도록 한다[5].

III. 본 론

본 논문은 CUDA를 이용하여 구현한 FDTD 알고리즘에서 속도 향상과 대용량의 데이터 연산을 수행하기 위해 OpenMP를 통한 다중 GPU제어를 설계 및 구현하였다. 본 장에서는 OpenMP 구현을 통해 변경된 기존의 FDTD알고리즘과 동작방식에 대해서 설명한다.

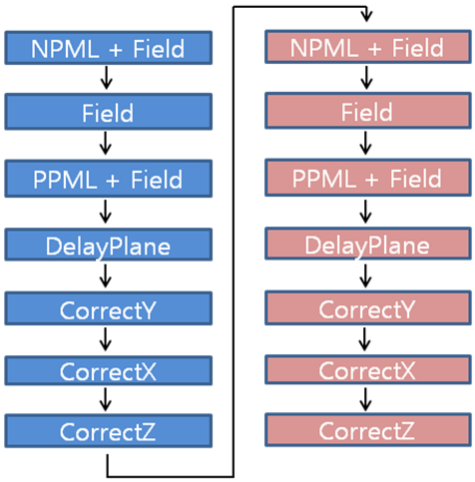

그림 3. CUDA FDTD 알고리즘
Fig. 3. CUDA FDTD Algorithm

<그림 3>은 CUDA에서 FDTD의 연산 순서를 나타낸다. 전기장 영역과 자기장 영역에 대한 결과 값을 얻기 위해 PML, Field, Delay, Correct 연산을 반복적으로 수행한다. 이는 FDTD 알고리즘의 특성상 시간에 대해 순차적인 연산을 수행하기 때문에 연산 순서는 절대적으로 지켜져야만 한다. GPU연산 시 Host와 Device 간의 메모리 복사 오버헤드가 크기 때문에 최초에 GPU에 데이터를 Load하여 연산을 수행하고 연산이 완료된 후 결과 값을 Host로 복사한다.

<그림 4>는 OpenMP를 이용한 연산을 위해 변경된 FDTD 연산 순서를 나타낸 그림이다.

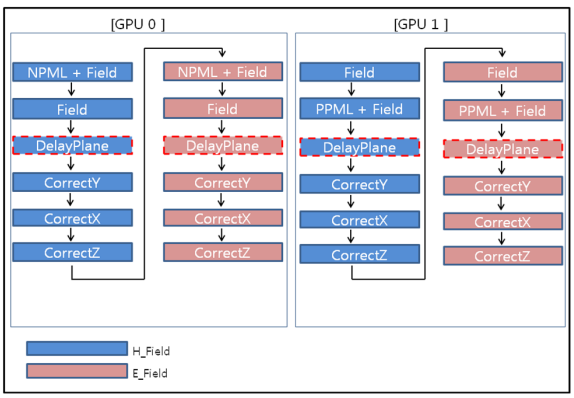

그림 4. CUDA FDTD 알고리즘
Fig. 4. CUDA FDTD Algorithm

실제적 연산의 수행은 GPU 1대를 이용한 것과 크게 다르지 않지만 NPML연산과 PPML연산을 분할하여 하나의 GPU에서 하나씩을 전담하여 수행하도록 연산 수식을 분할하였다. 또한 연산을 수행하는 대상 필드의 크기를 절반씩 분할함으로써 필드 데이터를 계산하기 위한 시간도 절반으로 감소하였다. 필드 데이터를 분할하는 기준은 총 계산할 메모리에서 Z축을 기준으로 절반씩 나누어 각각의 GPU에서 동시에 연산하도록 하였다. 이 때 전기장 영역과 자기장 영역은 각각 이전 Z축의 값이 필요하다. FDTD 알고리즘의 특성상 전기장 필드는 현재 자신의 Z축과 Z+1축에 대한 값들을 참조하게 되는데, Z축이 경계면이라면 다른 GPU로부터 Z축의 값을 얻어와야 한다. 자기장 필드의 연산도 마찬가지로 이전 Z축으로부터 값을 얻어와야 하기 때문에 다른 GPU로부터 결과 값을 얻어와야 한다. FDTD 연산에서 다수의 GPU사용 시 이와 같은 이유로 한 스텝이 이루어 질 때마다 동기화 시간이 필요 시되며, 속도 향상을 위해 최소한의 동기화 시간을 유도하여야 한다. 동기화 시간을 줄이기 위한 방안으로는 동일한 성능을 가지는 하드웨어 장치를 사용함으로써 각 GPU에서 수행하는 연산 시간의 차이를 줄여야 하고, 연산 대상이 되는 필드의 크기를 최대한 동일하게 하여 비슷한 시간에 연산이 완료되도록 하여야 한다.

IV. 실험환경 및 결과

표 1. GPU 환경
Table 1. GPU Environment

항목	값
GPU명	NVIDIA TESLA C2070
Core수	448개
Core clock	1.15GHz
Memory Clock	1.5GHz
Memory Size	6GB

<표 1>은 본 연구의 실험 환경으로써 슈퍼컴퓨터용 모델군인 TESLA C2070을 두 장 탑재하였다. 연산 시간 측정 방식은 CPU Clock을 이용하였으며, 결과 값의 검증은 Text 비교 툴을 이용하였다.

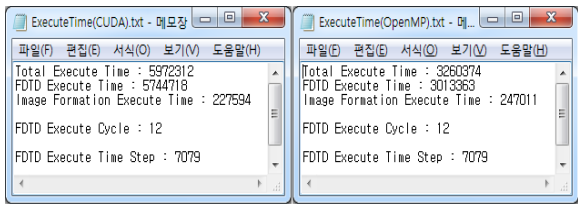

그림 5. 수행 시간 결과 화면
Fig. 5. Execute time result screen

<그림 5>의 결과 화면을 통해 향상된 연산 시간을 확인할 수 있다. <표 2>는 다양한 데이터 크기별 FDTD 연산의 수행시간을 정리한 표이다.

표 2. 수행시간 정리
Table 2. Execute time organize

(단위 : ms)

	GPU	OpenMP+GPU
100*100*220	185749	99434
200*200*220	709317	374980
300*300*320	2278721	1194993
400*400*420	5744718	3013363

FDTD의 수행시간 결과를 통해 OpenMP를 활용하여 연산을 수행하는 경우 약 1.9배의 연산 속도 향상 효과를 확인할 수 있다.

V. 결 론

반도체 공정에서 시뮬레이션을 수행하기 위해서는 매우 방대한 양의 연산을 수행하게 된다. 이를 한 장의 GPU를 이용하여 수행할 경우 제한된 자원으로 인하여 연산 수행에 비교적 오랜 시간을 소비하게 되고 메모리 크기 또한 크지 않아 많은 양의 데이터 연산을 수행하기에는 제한적이다. 본 논문에서는 이러한 문제를 OpenMP를 이용하여 개선하였고, 속도 측정을 통하여 실제적으로 개선된 FDTD의 연산 시간을 증명하였다. OpenMP를 통해 필요한 만큼의 CPU Core를 생성하고 각 Core별로 GPU를 점유해서 사용함으로써 다중의 GPU를 동시적으로 활용하여 FDTD 연산을 수행하였고 이에 따른 충분한 GPU자원 획득을 통해 FDTD 연산 시간이 감소하였다. FDTD 연산은 시간 순차적 연산이기 때문에 병렬 프로그래밍 수행 시 동시적인 연산 수행이 불가능하여 알고리즘을 개선하였다. 또한 GPU메모리에 Load되는 데이터가 분산되어 더 많은 양의 데이터 처리가 가능하게 되었다. 본 논문에서는 이러한 개선들을 통하여 FDTD 연산 속도가 두 장의 GPU사용 시 1.9배 증가한 것을 확인할 수 있었다.

향후 연구과제로는 클러스터간 메시지 패싱 기술인 MPI를 이용하여 다수의 클러스터 환경을 구축해 높은 병렬성을 지원하여 방대한 양의 FDTD 연산을 더욱 빠른 시간에 수행할 수 있도록 한다.

참고문헌

[1] David M. Sheen, Sami M. A., Mohamed D. A., ‘Application of the three-dimensional Finite-Difference Time-Domain method to the analysis of planar microstrip circuits’, IEEE Transantion on Microwave Theory and Techniques, vol. 38, pp 849-857, 1990.

[2] *http://en.wikipedia.org/wiki/CUDA*, “CUDA“

[3] NVIDIA CUDA Programming Guide V2.0, *http://kr.nvidia.com/object/cuda_develop_kr.html*, accessed on 13 April 2009.

[4] Babara Chapman, Gabriele Jost and Ruud van der Pas “Using OpenMP”, 2007.

[5] Rohit Chandra, Leonardo Dagum, Dave Kohr, Dror Maydan, Jeff McDonald, and Ramesh Menon “Parallel Programming in OpenMP” Morgan aufman Publishers, 2001.