디지털산업정보학회 논문지 제11권 제1호-2015년 3월
http://dx.doi.org/10.17662/ksdim.2015.11.1.027

국방정보시스템 성능향상을 위한 효율적인 GPU적용방안 연구*

고 장 혁** · 이 동 호***

The study on the Efficient methodology to apply the GPU for military information system improvement

Kauh Janghyuk · Lee Dongho

<Abstract>

Increasing the number of GPU (Graphic Processor Unit) cores, the studies on High Performance Computing Platform using GPU have actively been made in recent. This trend has led to the development of GPGPU (General Purpose GPU) and CUDA (Compute Unified Device Architecture) Framework.

In this paper, we explain the many benefits of the GPU based system, and propose the ICIDF(Identify Compute-Intensive Data set and Function) methodology to apply GPU technology to legacy military information system for performance improvement. To demonstrate the efficiency of this methodology, we applied this method to AES CPU based program obtained from the Internet web site. Simply changing the data structure made improved the performance of AES program. As a result, the performance of AES based GPU program is improved gradually up to 10 times. Depending on the developer's ability, additional performance improvement can be expected. The problem to be solved is heat issue, but this problem has been much improved by the development of the cooling technology.

Key Words : Multi-core, Parallel Processing, GPGPU, GPU, CUDA

Ⅰ. 서론

CPU의 다중코어(multi-core)와 GPU의 다수코어(many-core) 형태의 프로세서가 보편화됨에 따라 병렬처리 프로그래밍의 중요성은 점점 더 커지고 있다. 그동안 GPU는 그래픽스 계산 시 한 번에 수많은 기하 및 화소 데이터를 병렬적으로 처리하기 위해 코어의 수를 지속적으로 늘려 왔으며, 연산시 발생하는 메모리 접근 지연 시간을 숨기기 위해 쓰레드를 사용한 병렬처리로 계산 능력을 획기적으로 향상시켜 왔다. 이렇게 다수코어 기반 스트리밍 계산에 특화된 GPU의 연산 능력은 현재 CPU 대비 상당한 정도의 성능 우위를 점유하게 되었다. 이러한 특성으로 인하여 GPU는 방대한 데이터에 대해 동일한 계산 패턴을 요구하는 영역에서 사용이 늘어나고 있으며, 특히 과

* 이 논문은 2014년도 광운대학교 교내 학술연구비 지원에 의해 연구되었음.
** 국방과학연구소 선임 연구원
*** 광운대학교 컴퓨터소프트웨어학과 교수

학계산 분야나 그래픽스 분야에서 CPU를 대신하여 HPC(High Performance Computing) 플랫폼으로 각광받고 있다[1-2]. 미국의 NVIDIA사에서는 GPGPU (General Purpose GPU) 아키텍처인 CUDA (Compute Unified Device Arc hitecture) API를 발표하여 프로그래머에게 C/C++언어와 유사한 문법을 제공함으로써 GPU 구조에 대한 특별한 지식이 없이도 프로그래머에게 친숙한 병렬 처리 개발 환경을 제공한다.

국방 분야에서도 GPU가 초기에는 그래픽 기능위주의 영상처리 및 상황도(COP) 등에 적용되었으나 현재는 앞에서 언급한 민간분야와 같이 고성능 컴퓨팅이 필요한 분야에 적용되기 시작하였다. 하지만 아직은 활용이 그다지 높지 않은 실정이다.

본 논문에서는 기존에 개발된 국방 정보시스템의 성능을 향상시키기 위한 효율적인 GPU 적용방안을 제안하고 이에 대한 가능성 및 성능향상을 보이고자 한다.

본 논문의 구성은 Ⅱ장에서 이러한 GPU 기술의 발전 추세에 기술한다. Ⅲ장에서는 기존 레거시 시스템에 GPU기술을 적용하는 ICIDF 방법론을 제안하였고 이에 대한 적용 예로 CPU기반의 AES 암호 알고리즘 프로그램에 적용하는 과정과 이에 대한 적용 사례의 CPU버전 대비 GPU 버전의 성능 향상 결과를 보이며 Ⅳ장에서는 이에 대한 결론을 맺고자 한다.

Ⅱ. GPU 적용 장점 및 관련연구

2.1 GPU 적용 장점

국방 정보시스템에 GPU를 적용한다면 그림1과 같은 장점이 있다. 크기 및 소비전력에서도 우수하여 임베디드 분야에도 많이 사용되고 있다. 이러한 장점들은 단지 프로세서 자체의 이점에 그치지 않고 다음과 같은 이점이 있다[3].

<그림 1> CPU vs GPU[4]

첫째는, 시스템 구축비용의 절감이다. 같은 처리능력을 고려할 경우 <그림1>에 나와 있는 것과 같이 CPU 경우에 비해 구축비용이 크게 절약된다. 범용적인 목적으로 여러 가지를 고려하여 설계된 CPU와 수치 계산만을 목적으로 설계된 GPU는 구조적인 면에서 다르며 또한 그 비용도 차이가 날 수밖에 없다. 또한 GPU는 전력 소비가 적어 여러 개의 GPU를 한 보드에 장착할 수 있다. 현재 GPU 카드를 4~8장까지 탑재할 수 있는 컴퓨터가 출시되어 있으며 <그림1>에서와 같이 성능의 차이를 5배라고 가정할 때 GPU로 구성한 컴퓨터의 경우 CPU 컴퓨터보다 20~40배의 성능을 낼 수가 있다. 현재는 GPU 기술이 더욱 발전하여 더 많은 코어를 가진 제품이 많이 나오고 있어 차이는 그 이상이라고 할 수 있다.

둘째는, 공간 및 유지비용의 절감이다. 앞서 설명한 것과 같이 성능이 20배정도 차이가 난다고 가정한다면 고성능(HPC)이 요구되는 현실에서 성능을 만족시키기 위해 20개의 Rack으로 구축해야 될 경우를 1개의 Rack으로 대체 구현이 가능해 지는 것이다. 이는 공간문제에 그치지 않고 이에 대한 유지비용(전력, 항온항습, 등)의 절감과 운용인력의 절감을 가져 올 수 있다. 시스템을 구축 및 운용해 본 사람이라면 이러한 장점이 얼마나 큰 것인지 알 것이다[3].

2.2 GPU 관련연구

GPU 관련연구는 크게 2가지 방향으로 정리할 수 있는데 병렬화 및 최적화 기술이 한 방향이고, 최근 이슈가 되고 있는 빅 데이터 처리를 위한 기술로 GPU 클러스터링 기술과 MapReduce 프레임워크 기술이 한 방향이다.

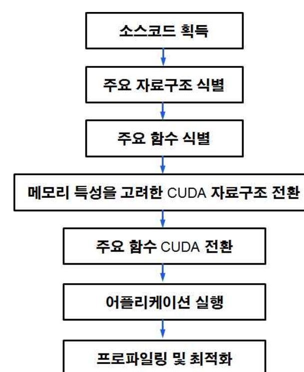
첫 번째, 병렬화 및 최적화 관련 연구도 다양하게 이루어 지고 있다. AntMinerGPU처럼 기존의 알고리즘에 GPU 기술을 적용하기 위해 알고리즘을 변경하여 병렬화 및 최적화를 이루고자 하는 연구와 병렬화 및 최적화를 지원하는 방안에 대한 연구로 나눌 수 있다.[5-6] 기존 알고리즘에 GPU 기술을 적용하는 연구는 이미지 프로세싱, 데이터 마이닝 등 여러 분야에서 다양하게 이루어지고 있다.

병렬화 및 최적화를 지원하는 방안으로는 NVIDIA에서 제안하고 있는 병렬 축소 (Parallel Reduction)가 있다[5]. 이를 통해 약15~30배 이상의 성능향상이 가능하다고 한다. 그러나 이에 대한 적용은 사례에 따라 다르며 정형적인 방법이 있는 것도 아니다. 병렬화를 위한 또 다른 방안으로 기존에 병렬 축소를 제안했던 M. Harris가 제안하는 CUDPP(CUDA Data Parallel Primitives) 라이브러리를 이용한 방법이 있다. 이는 parallel prefix sum (“scan”), parallel sort와 parallel reduction 등과 같은 데이터 병렬 알고리즘의 프리미티브들을 라이브러리로 제공한다. 기존의 알고리즘을 CUDPP에서 제공되는 병렬 프리미티브들로 구현하면 데이터 병렬에 따른 성능향상을 얻을 수 있고, 이에 대한 연구나 프레임워크 개발이 증가하고 있다[6].

두 번째, 빅데이터 처리 기술은 최근에 이슈가 되면서 가장 연구가 활발하게 진행되고 있는 분야이다. GPU clustering, MapReduce based GPU 기술이 여기에 해당되겠다[7-8]. MapReduce 기술은 구글에서 분산 컴퓨팅을 지원하기 위한 목적으로 제작하여 2004년 발표한 소프트웨어 프레임워크로 함수형 프로그래밍에서 일반적으로 사용되는 Map과 Reduce라는 함수 기반으로 주로 구성된다. 여러 Mapper들이 작업을 해서 Reduce에서 통합하는 구조로 Device(=GPU)들이 계산하여 Host(=CPU)에서 통합하는 CUDA의 구조와 유사하다. 초기에 CPU 기반의 멀티 코어 환경에서 MapReduce 모델을 구현한 Phoenix가 제안되었고, 단일 GPU환경에서 MapReduce를 구현한 Mars가 제안되었다. 현재는 멀티 GPU 환경에서 구현한 GPMR, i-MapReduce, MapCG, CellMR 등과 같은 다양한 MapReduce 프레임크들이 개발되고 있다. 이러한 멀티 GPU 환경에서도 데이터 이동, Out of core 데이터, 모든 GPU의 활용, MapReduce 적용을 위한 알고리즘 변경 등이 추가 연구되어야 할 문제이다.

Ⅲ. 효율적인 GPU 적용방안

3.1 ICIDF(Identify Compute-Intensive Data set and Function) methodology



<그림 2> ICIDF 방법론

기존 레거시 시스템의 코드를 CUDA 코드로 쉽게 변환하는 방법으로는 Translator나 Converter를 활용하는 방안이 있다[9]. 병렬 프로그래밍의 어려움을 극복하기 위한 방법으로 이러한 도구에 대한 연구도 많이 이루어 지고 있다. 하지만 이러한 도구들은 배열 연산, 정렬과 같은 간단한 프로그램에 대해 실험적으로 적용되며 성능이나 적용범위에 한계가 있다.

본 논문에서 제안하는 ICIDF(Identify Compute-Intensive Data set and Function) 방법론은 프로그램을 새로 개발하는 것이 아니고, <그림2>와 같이 CPU 기반의 프로그램 소스코드를 획득해서 GPU의 이점인 연산처리에 초점을 두고 연산이 많이 이루어지는 자료구조와 함수를 파악하여 메모리 특성과 병렬처리를 고려한 CUDA 코드로 변환 실행한 후 프로파일링과 최적화를 통해 성능을 한층 더 향상시키는 방안을 제안한다.

(1) 소스 코드 획득 - 기존 레거시 시스템의 소스 코드를 획득한다. 국방에서는 정보시스템 구축 후에 재사용성 및 상호 운용성을 높이기 위해 활용성이 높은 컴포넌트를 관리하기 위해 오래전부터 레포지토리(Repository)를 구축/운용하고 있다. 그래서 손쉽게 CPU 환경에서 개발된 소스 코드를 획득할 수 있다.
(2) 주요 자료구조 식별 - 프로그램에서 많이 사용되는 자료구조를 식별하는 것으로 일반적으로는 구조체 및 배열의 형태이다. 참조 회수가 높은 자료구조를 식별한다. 기본적으로 CUDA는 CPU 프로그램대비 자료를 GPU로 복사하여 계산하고, 다시 복사해 오는 추가적인 작업이 요구됨으로 데이터에 대한 참조회수가 적을 경우에는 비효율적일 수 있다.
(3) 주요 함수 식별 - 처리량이 많거나 자주 호출되는 기능을 식별한다. 이를 위해서는 분석이 필요하며 CUDA에서 제공하는 분석도구인 Visual Profiler 및 NSight를 이용하면 도움이 된다. (2)단계 주요 자료구조에도 이러한 분석도구들은 도움이 된다. 이렇게 식별된 기능들은 컴포넌트로 개발, 관리되면 좋다.
(4) 메모리 특성을 고려한 CUDA자료구조 전환 - (2)단계에서 식별한 자료구조에 대해 CUDA 메모리 특성을 고려한 자료구조로 전환한다. 여기서 결정한 메모리 종류는 어플리케이션 실행 및 최적화 단계에서 변경될 수 있다.
(5) 주요 함수 CUDA 전환 - (3)단계에서 식별된 함수를 CUDA로 전환한다. (4)단계에서 식별된 자료구조에 따라 함수의 구현 형태가 달라진다.
(6) 어플리케이션 실행 - 프로그램을 병렬적으로 재구성하여 어플리케이션 실행한다.
(7) 프로파일링 및 최적화 - 어플리케이션이 실행한 후에도 (4)단계에서 적용한 자료구조의 메모리 종류를 변경하고, 일반적인 프로그램 병렬기법을 적용하여 최적화를 이룬다.

성능향상은 (4)단계 메모리 특성을 고려한 적절한 자료구조를 정의하는 것만으로도 데이터 양이 클 경우 높은 성능향상을 얻을 수 있다. 또한 (4), (5), (7)단계는 개발자의 기술 수준에 좌우되는 부분으로 크게는 10배이상의 많은 성능 차이가 날 수도 있다.

3.2 AES암호화 적용사례

앞서 설명한 것과 같이 국방 정보시스템의 경우, 산출물관리와 컴포넌트 레포지토리 등을 통해 쉽게 소스 코드를 획득할 수 있으나 본 논문에서는 보안상의 이유로 제안한 방안의 적절성을 보이기 위하여 민간에서 쉽게 구할 수 있는 표준 암호알고리즘인 AES 암호알고리즘을 획득하여 적용하였다. AESS 암호 알고리즘은 대칭키 기반의 128, 192, 256비트 블록 암호

알고리즘으로 non-Feistel 구조에 속한다[10]. 키 확장 알고리즘으로부터 생성되는 라운드 키 크기는 평문(Plaintext)과 암호문(Ciphertext)의 크기와 동일한 128 비트로 출력하고, Subytes, ShiftRows, MixColumns, AddRoundKey 단계의 함수 연산을 수행하여 암호화 한다.

(1) 소스코드 획득 - 본 논문에 기반이 되는 CPU 버전의 AES 암호 알고리즘[10]은 이런 구조로 구현 되었다. S-Box와 같은 table 형태의 배열구조가 병렬화하기 좋은 구조로 GPU 전환시 큰 성능의 향상을 보일 수 있다.

(2) 주요 자료 구조 식별 - AES 프로그램에서 주로 사용되는 자료구조는 S-BOX를 구현한 자료 구조와 연산이 이루어지는 공간인 버퍼로 나눌 수 있다.

성능향상을 위해 S-Box에 의한 연산들을 미리 계산하여 table형태로 구성하고 있다. 실제로 S-box에 대한 정보는 다음과 같은 9개의 테이블로 구성된다. 이러한 table 형태의 연산에 GPU는 장점이 있다.

```
uchar Sbox[256]
uchar InvSbox[256]
uchar Xtime2Sbox[256]
uchar Xtime3Sbox[256]
uchar Xtime2[256]
uchar Xtime9[256]
uchar XtimeB[256]
uchar XtimeD[256]
uchar XtimeE[256]
```

계산이 이루어지는 자료구조는 다음과 같다.

```
uchar state[256];
```

(3) 주요 함수 식별 - Encrypt/Decrypt 함수가 주요 함수이며 내부적으로는 앞서 설명한 Subytes, ShiftRows, MixColumns, AddRoundKey 등의 함수가 되겠으며 출력을 위한 함수도 해당된다.

(4) 메모리 특성을 고려한 CUDA 자료구조 전환 - (2)단계에서 식별된 테이블에 관련된 자료구조는 일단 디바이스 메모리로 정의하며 테이블과 같이 값을 수정하지 않고 참조만 하는 경우에는 일반적으로 상수 메모리로 정의하며 여기서는 상수 메모리로 정의한다.

```
__device__ __constant__ uchar Sbox[256]
__device__ __constant__ uchar InvSbox[256]
…
```

state[256] 자료구조는 공유 메모리를 이용하여 _Encrypt()와 _Decrypt()함수내에 다음과 같이 정의하였다.

```
__shared__ uchar buffer[256];
```

(5) 주요 함수 CUDA 전환 - (3)단계에서 식별된 함수들을 CUDA로 구현하기 위해서는 필요한 데이터를 Host(CPU)에서 받아야 하는 부분과 계산 후 Host로 돌려 주어야 하는 부분, 이에 따른 메모리 선언과 메모리 종류에 따른 함수 구현이 있겠다.

(6) 어플리케이션 실행 - (4), (5)단계에서 조치한 부분이 정상적으로 구현되었는지 실행을 해야 한다. 이를 위해서 자료를 검증하는 루틴을 포함하는 것을 잊지 말아야 한다.

(7) 프로파일링 및 최적화 - 최적화를 위한 병렬처리 기법을 적용하고, 다음과 같은 CUDA에서 제공하는 Visual Profiler 및 NSight를 사용하여 자료구조 및 함수들의 성능 및 개선점을 파악하여 부족한 부분을 보완한다.

성능향상을 위한 주된 부분도 역시 자료구조와 함수이다. 메모리들에 대한 특성은 어디까지나 일반적인 이야기이며, 실제 CUDA 컴파일러의 발전과 프로

그램의 다양한 경우에 따라 그 성능은 차이가 발생한다. (4)단계에서 정의한 테이블에 대한 자료구조를 우리는 상수 메모리로 선언하였으나 실제로는 디바이스 메모리로 선언한 경우가 더 성능이 향상되었고, 공유메모리를 이용하는 경우는 메모리 복사에 따른 추가 소요시간으로 성능이 더 나쁘게 나왔다. 그래서 최종적으로는 테이블정보들은 디바이스 메모리를 사용하고 버퍼는 공유 메모리를 사용하였다. (7)단계에서 다음과 같은 병렬 축소 (Parallel Reduction)기법 등을 통해 구조변경을 이루어 성능향상을 이룰 수 있다.

- 메모리 결합 전송(Memory coalescing)
- 뱅크 충돌(Bank conflicts)
- 분기 발산(Divergent branching)
- 지연시간 숨기기(Latency Hiding)
- 병목지점 파악
- 루프 언롤링(Unrolling)
- 알고리즘의 최적화

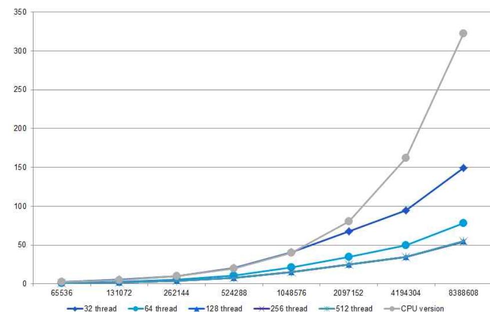
```
for ( int i = 0; i < 4*Nb; ++i )
state[i] = tmp[i];

state[0] = tmp[0];
state[1] = tmp[1];
state[2] = tmp[2];
.........................................
state[14] = tmp[14];
state[15] = tmp[15];
```

<그림 3> 루프 언롤링(Loop Unrolling)

<그림3>은 병렬축소 방법 중 하나인 루프 언롤링(Loop Unrolling)을 적용한 예로 이와 같이 for문들을 분해하여 나열하는 것만으로도 성능향상을 보인다. 많이 반복되는 루틴의 for문일수록 더 높은 성능향상을 보이며 여러 개의 for문을 한 개의 for문으로 합칠 수도 있다. 또한 분기 발산(Divergent branching)은 do ~while문이나 if문 등과 같이 분기를 하는 명령구조를 단순구조로 변경함으로 성능향상을 보일 수 있으며 CUDA에서 제공하는 cudaMemcpyAsync()와 같은 비동기 전송함수를 이용함으로서 지연시간 숨기기(Latency Hiding) 성능향상을 이룰 수 있다.

3.3 실험 결과

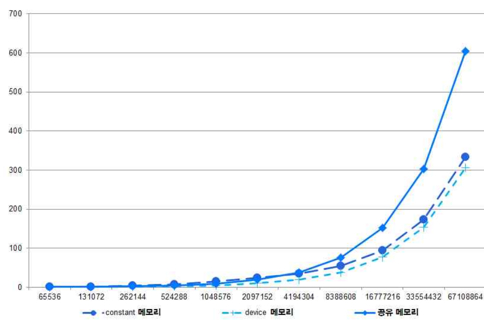
실험환경으로 CPU는 Intel(R) Xeon(R), 2.93GHz, 메모리는 4GB, 그래픽 카드(GPU)는 NVIDA Quadro 600 환경에서 수행하였다. 블록당 스레드의 수를 32개에서부터 2배씩 증가하며 시험을 진행했으며 CPU 버전과의 성능차이를 보기 위해 같이 도시하였다. AES의 CPU 버전과 GPU 버전의 성능과 그 비교치이다. 이는 CUDA의 SIMT (Single Instruction Multiple Thread) 특성을 잘 볼 수 있는 결과이다. 성능은 16KB부터 64MB까지 측정하였으며 CPU 버전 AES는 메시지 크기가 증가함에 따라 처리시간도 지수분포를 이루며 급격하게 증가하는 반면 GPU 버전 AES의 경우는 거의 선형적으로 증가함을 알 수 있다. 이러한 특징은 요즘과 같은 고성능이 요구되는 어플리케이션이 많은 상황에서 시스템을 설계할 때 확장성이라는 측면에서 매우 중요한 요소가 아닐 수 없다.



<그림 4> 블록당 쓰레드 수 증가에 따른 성능변화

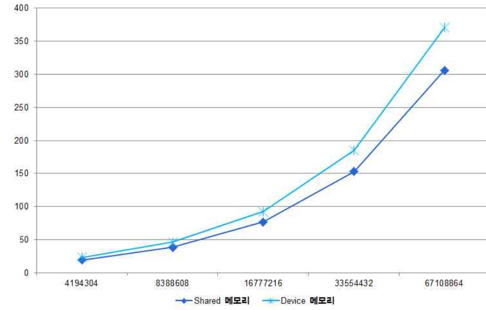
CPU 버전의 경우 메시지가 커짐에 따라 <그림4>와 같이 성능의 한계를 가지게 된다. 하지만 GPU 버전의 경우 병렬화에 따른 효과로 성능이 거의 비례적으로 증가하는 특성을 보이므로 GPU Device의 성능향상이나 GPU 노드의 증가, 즉 디바이스의 다중화를 통해 원하는 성능을 만족할 수 있을 것이다. 또한 메시지 크기가 클 경우에는 GPU 버전에 의한 성능향상이 크지만, 메시지 크기가 작다면 GPU를 통한 성능향상이 미비하다는 것을 알 수 있다. 이는 CPU와 GPU 사이의 데이터 이동이라는 추가적인 오버헤드가 있으므로 이를 극복할 만큼 데이터 량이 많거나 큰 처리능력을 요구하는 경우에는 성능이 우수하지만, 데이터 량이 작거나 작은 계산능력을 요구하는 일에는 적절하지 않다는 것이다. 물론 이런 경우에도 작은 데이터들을 모아 큰 데이터로 만드는 chunking 기법을 통해 처리할 데이터량을 증가시킨 후 GPU를 적용하는 기법을 사용한다면 역시 우수한 성능을 보일 것이며, 이에 대한 연구도 많이 이루어 지고 있다.

S-Box 관련 자료구조의 메모리별 성능차이와 버퍼 자료구조의 메모리별 성능차이는 <그림5>와 같다. 디바이스 메모리에서 가장 우수한 성능을 보였고, 공유 메모리의 경우가 가장 성능이 낮았다.



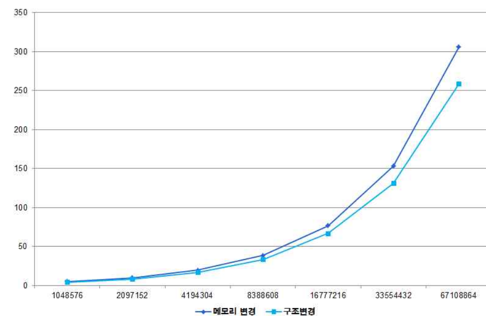
<그림 5> S-Box 메모리 변화에 따른 성능 차이

공유 메모리의 경우, 추가적인 복사를 해야 하는 점이 있어 모든 경우에 가장 우수한 성능을 보이는 것은 아니다. 블록내 공유가 가능할 때 우수한 성능을 보인다고 하겠다.



<그림 6> 버퍼 메모리의 메모리별 성능 차이

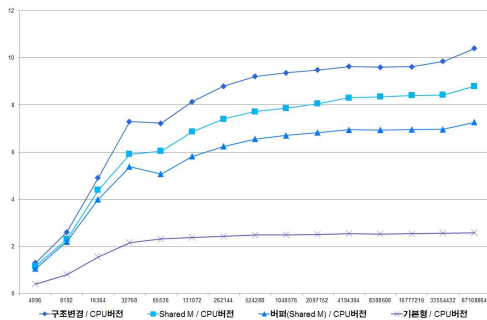
이러한 특성은 <그림6>에 잘 나오는데 블록내 작업을 하는 버퍼 공간에 대해서는 역시 공유 메모리가 좋은 성능을 보였다. 주목할 점은 참조만 하는 경우 상수 메모리가 디바이스 메모리보다 우수하다고 알려져 있지만 실제 실행해 본 결과는 크게 차이가 나지는 않았지만 반대의 결과가 나왔다. 이는 CUDA 컴파일러의 발전에 따라 컴파일러에서 자동으로 처리해 주는 부분이 많아져서 이러한 성능이 나온 것으로 보인다.



<그림 7> 메모리 변경과 구조변경으로 이룬 성능차이

<그림7>은 메모리 특성을 이용한 성능향상, 즉 S-Box관련 정보 테이블을 디바이스 메모리로 하고 버퍼 공간을 공유 메모리로 정의하여 이룬 성능향상과 여기에 다양한 병렬기법을 적용하여 구조를 변경함으로써 이룬 성능향상의 차이를 보여 주는 그래프이다.

3.4 실험 분석



<그림 8> CPU대비 성능 차이

최종적으로는 <그림8>은 본 논문에서 제안한 방법론을 적용하여 이룬 성능 차이를 CPU시스템 대비 성능 배수로 단계별로 보여 주고 있다. 메시지 크기 32768byte에서 그래프의 변화가 보이는 것은 버퍼의 크기와 관련이 있는 것으로 판단된다. 기본형 버전은, CPU 프로그램을 GPU의 CUDA 프로그램으로 변경한 것으로 메모리 특성 및 함수의 특성을 고려하지 않고, 처리할 데이터량이 많은 부분은 GPU 디바이스로 전송하여 처리하고 다시 돌려받는 부분을 처리한 버전이다. 이렇게 함으로 얻어진 성능 향상은 약 2배이다. 버퍼 버전은, 기본형 버전의 버퍼 자료구조의 메모리 종류를 디바이스 메모리에서 공유 메모리로 바꾼 버전으로 이에 따라 버퍼를 공유할 수 있는 형태로 함수를 변경하였다. 이렇게 얻어진 성능 향상은 약6배이상이다. Shared Memory버전은 버퍼 버전에 S-Box 및 이에 관련된 자료구조들을 디바이스 메모리로 선언하여 구현한 버전으로 약 8배이상 성능이 향상되었다.

마지막으로 Shared Memory버전에 앞서 설명한 병렬축소 및 다양한 최적화 기법을 적용하였다. for문들을 분해하여 나열하는 루프 언롤링기법, do ~ while문이나 if문 등과 같이 분기를 하는 명령구조를 단순구조로 변경하는 분기 발산기법, 여러 불필요한 간접 선언들을 직접 선언으로 변경하고, 비동기 전송 함수를 이용하여 지연시간 숨기기 등을 수행하였다. 이렇게 얻어진 성능 향상은 약 10배 이상이다. 이것이 최고의 성능향상을 의미하는 것은 아니다. 여기에는 개발자의 능력과 경험에 크게 좌우되며 시스템의 특성도 반영되기 때문이다.

Ⅳ. 결론

많은 레거시 시스템을 운용하고 있는 조직이라면 기존 레거시 시스템의 성능 개선에 관심이 많을 것이다. 이러한 현실에서 기존의 레거시 시스템을 적은 투자로 10배 이상의 성능향상을 가져올 수 있다면 매력적이지 않을 수 없다. 또한 CUDA로 개발할 경우 GPU 프로세서가 발전함에 따라 GPU 교체 및 추가만으로도 어느 정도의 성능향상을 기대할 수 있는 장점이 있다.

본 논문에서는 기존 레거시 시스템에 GPU기술을 적용하기 위한 현실적인 방안으로 ICIDF(Identify Compute- Intensive Data set and Function) 방법론을 제시하였고 이를 입증하기 위해 AES 프로그램의 적용사례를 통해 충분한 성능 향상이 이루어짐을 보였다. 현재 국내 시스템 개발에 GPU기술이 적용된 사례가 많지 않지만, 점점 늘어가고 있는 추세이다. 아직 발열과 같은 문제가 있으나 현재도 많은 벤더들이

상용제품들을 출시할 만큼 냉각기술이 많이 발전되고 있으니 큰 문제는 없을 것으로 판단된다. 또한 GPU기술에 의한 성능향상이 개발자의 실력에 많이 좌우되기 때문에 보다 폭넓은 시스템 적용과 GPU 개발자의 양성에도 힘써야 할 것으로 판단된다.

참고문헌

[1] 이승학ㆍ김경훈ㆍ안치영ㆍ최승원, “GPU를 이용한 SDR 시스템용 LTE MIMO 기지국 기능 구현,” 디지털산업정보학회 논문지, 제8권, 제4호, 2012, pp. 91-98.

[2] 이윤혁ㆍ김동욱ㆍ서영호, “GPGPU기반의 디지털 홀로그램 콘텐츠의 고속 생성 기법,” 디지털산업정보학회 논문지, 제9권, 제1호, 2013, pp. 151-162.

[3] 고장혁ㆍ이동호, “GPU를 이용한 정보시스템 성능향상에 관한 연구,” 한국군사과학기술학회 종합학술대회, 2013.

[4] GE Intelligent Platforms, “GPGPU COTS Platforms - High-Performance Computing Solutions,” 2011, pp. 2-6, http://defense.ge-ip.com/gpgpu

[5] Mark Harris, “Optimizing Parallel Reduction in CUDA, NVIDIA," 2007, pp. 7-37.

[6] M. Harris, S. Sengupta, Y. Zhang, and A. Davidson, "CUDPP: CUDA data parallel primitives library," 2009. http://gpgpu.org/developer/cudpp/.

[7] Jeff A. Stuart, J. D. Owens, “Multi-GPU MapReduce on GPU Clusters,” IEEE International Parallel & Distributed Processing Symposium, 2011, pp. 1068-1079.

[8] Reza Farivar, Abhishek Verma, Ellick chan, Roy H Campbell, “MITHRA: Multiple data Independent Tasks on a Heterogeneous Resource Architecture,” IEEE Cluster Computing and Workshops, 2009.

[9] Parth R. Trivedi, “c2cudatranslator: Automatic conversion of source code for C to CUDA C,” 2012. http://code.google.com/p/c2cudatranslator

[10] Karl malbrain, 786/1280 Byte Table AES C byte-implementation 03 OCT 2006, http://www.geocities.ws/malbrain

■ 저자소개 ■

고 장 혁
Kauh Janghyuk

1998년 3월~현재 국방과학연구소 선임 연구원
2014년 2월 광운대학교 컴퓨터과학과 박사과정수료
1998년 2월 광운대학교 컴퓨터과학과 이학석사
1996년 2월 광운대학교 컴퓨터과학과 이학사

관심분야 : 정보 보호, 정보시스템, 병렬처리
E-mail : jhkauh@add.re.kr

이 동 호
Lee Dongho

1984년 9월~현재 광운대학교 컴퓨터소프트웨어학과 교수
1988년 2월 서울대학교 컴퓨터공학과 공학박사
1983년 2월 서울대학교 컴퓨터공학과 공학석사
1979년 2월 서울대학교 전자공학과 공학사

관심분야 : 컴퓨터 네트워크, 차세대 인터넷, BAN
E-mail : dhlee@kw.ac.kr

논문접수일: 2015년 2월 13일
수 정 일: 2015년 3월 5일
게재확정일: 2015년 3월 9일