저가의 그래픽스 장치를 이용한 GPU 클러스터 시스템 구현

이종민^{*}, 이중화^{**}, 김성우^{***}

요 약

최근 들어 GPU 성능이 급격하게 향상됨에 따라서 그래픽 처리가 아닌 일반 계산 분야에도 활용하기 위하여 GPGPU 분야에 대한 연구에 많이 이루어지고 있다. 본 논문에서는 저가의 그래픽스 장치에 내재된 GPU를 활용하여 비용 대비 성능이 높은 시스템을 구현하기 위하여 기존의 슈퍼컴퓨터 구조를 벤치마킹한 시스템 구조를 제안하고, 8개의 GPU를 사용하는 GPU 클러스터 시스템을 구현한다. 또한 이를 위한 소프트웨어 환경을 구축하여 n-body 문제를 사용하여 시스템 성능을 분석한다. 성능 평가 결과 사용하는 GPU 수가 많아질수록 통신경비로 인하여 문제 크기가 충분히 커야 효율적임을 알 수 있었다. 그리고 GPU내 자원 부족 문제 때문에 발생하는 문제 크기 제약을 블록 단위로 계산하여 완화할 수 있는 방법을 적용하여 최대 팔백만 개의 천체에 대한 계산을 할 수 있었다.

Implementation of a GPU Cluster System using Inexpensive Graphics Devices

Jong Min Lee^{*}, Jung Hwa Lee^{**}, Seong Woo Kim^{***}

ABSTRACT

Recently the research on GPGPU has been carried out actively as the performance of GPUs has been increased rapidly. In this paper, we propose the system architecture by benchmarking the existing supercomputer architecture for a cost-effective system using GPUs in low-cost graphics devices and implement a GPU cluster system with eight GPUs. We also make the software development environment that is suitable for the GPU cluster system and use it for the performance evaluation by implementing the n-body problem. According to its result, we found that it is efficient to use multiple GPUs when the problem size is large due to its communication cost. In addition, we could calculate up to eight million celestial bodies by applying the method of calculating block by block to mitigate the problem size constraint due to the limited resource in GPUs.

Key words: GPU, Parallel Processing(병렬 처리), Cluster(클러스터), OpenCL, PVM

1. 서 론

최근 들어 GPU(Graphics Processing Unit) 성능

이 급격하게 향상됨에 따라서 그래픽, 게임과 같이 전통적으로 GPU를 사용하던 분야뿐만 아니라 일반 계산 분야에도 이를 활용하기 위한 연구가 많이 이루

※ 교신저자(Corresponding Author): 이종민, 주소: 부산광역시 부산진구 엄광로 995(614-714), 전화: 051)890-1736, FAX: 051)890-2629, E-mail: jongmin@deu.ac.kr
접수일: 2011년 7월 20일, 수정일: 2011년 9월 8일
완료일: 2011년 10월 4일

^{*} 정회원, 동의대학교 컴퓨터소프트웨어공학과

^{**} 정회원, 동의대학교 컴퓨터소프트웨어공학과
(E-mail: junghwa@deu.ac.kr)

^{***} 정회원, 동의대학교 컴퓨터소프트웨어공학과
(E-mail: libero@deu.ac.kr)

※ 이 논문은 2010학년도 동의대학교 교내연구비에 의하여 연구되었음(2010AA182).

어지고 있다. 1990년대 후반 3D 그래픽 처리를 위한 가속 기능이 필요함에 따라서 GPU의 성능은 지속적으로 향상되어 현재는 거의 모든 개인용 컴퓨터에 장착되어 사용되고 있으며, 범용 응용 분야로 그 적용 분야를 넓혀가고 있다[1]. 2011년 6월에 발표된 세계 500대 슈퍼컴퓨터 중 2위 Tianhe-1A, 4위 Nebulae, 5위 TSUNAMI 2.0이 모두 Nvidia사의 GPU를 사용할 정도로 슈퍼컴퓨터 분야에서도 GPU를 활용하는 연구가 많이 진행되고 있다[2].

GPU를 활용하여 기존 CPU의 연산을 보다 고속화시키기 위한 연구는 전통적인 그래픽스 처리 분야 외에도 의료 분야[3], 공간 데이터 처리[4], 비디오 인코딩과 같이 계산이 많이 필요한 분야에서 적용하기 위한 연구가 많이 이루어지고 있다. GPU를 이러한 범용 계산에 사용하는 것을 GPGPU(General Purpose computing on GPU)라고 한다. GPU를 범용 계산에 활용하기 위한 연구는 NVIDIA와 AMD/ATI에서 주도적으로 진행되고 있다. NVIDIA는 2006년 11월 CUDA(Compute Unified Device Architecture)라는 범용 병렬 컴퓨팅 구조를 도입하여 기존의 GPU를 범용 계산에 활용할 수 있는 기반을 제시하였다. AMD/ATI는 2006년 CTM(Close-To-Metal) 기술을 소개하여 GPGPU 개념을 도입하기 시작하였다.

GPU를 슈퍼 컴퓨팅 분야에서 대규모로 사용하는 연구는 이미 활발하게 진행되고 있어 GPU의 효율성은 입증되었으나, 일반적으로 접근하기 어려운 문제점이 있다. 국내에서도 이러한 GPGPU 분야에 대한 연구가 여러 분야에서 이루어지고 있으나 아직은 GPU 한두 개 정도를 사용하여 적용 가능성을 확인하고 있는 수준으로 판단된다. 개인용 컴퓨터에는 일반적으로 GPU를 한두 개 정도 장착하여 사용하는 것이 가능하여 쉽게 사용가능하나, 계산 용량에 한계가 있는 단점이 있다. 따라서 좀 더 많은 계산 용량을 필요로 하는 연구에 사용가능한 GPU 플랫폼에 대한 연구가 필요하다.

본 논문에서는 좀 더 쉽게 GPU 컴퓨팅을 활용할 수 있도록 저가의 그래픽스 장치에 내재된 GPU를 사용하여 계산 노드를 구성하고, 이러한 계산 노드를 고속 스위치로 연결하여 사용하는 GPU 클러스터 시스템 구조를 제안하고 이를 구현한다. GPGPU 전용으로 나온 고가의 장비 대신 게임 목적으로 사용되는 저가의 그래픽스 장비를 활용함으로써 비용 대비 성

능이 우수한 GPU 클러스터 시스템을 구현하는 것이 본 논문의 목적이다. 구현된 GPU 클러스터 시스템의 성능 평가를 위하여 천체 시뮬레이션 연구에 사용되는 n-body 문제[5]를 사용하여 GPU 수 증가에 따른 시스템의 계산 능력을 측정한다. CPU와 GPU의 계산 용량, GPU 수와 문제 크기를 변화시켰을 때의 계산 용량 등을 다양하게 측정함으로써 구현된 시스템의 성능을 평가한다.

본 논문의 구성은 다음과 같다. 2절에서는 GPU를 범용 계산 목적으로 활용하는 GPGPU 분야에 대하여 기술하고, 기존의 GPU 클러스터 시스템에 대하여 알아본다. 3절에서는 본 논문에서 제안하는 시스템 구조와 소프트웨어 개발 환경을 기술하고, 4절에서 구현된 시스템에 대한 성능 평가를 한다. 마지막으로 5절에서는 결론을 기술한다.

2. 관련 연구

2.1 GPGPU

최근 들어 GPU 개발은 주로 NVIDIA와 AMD/ATI 양사가 주도하고 있다. 본 절에서는 시스템 구현에 사용된 NVIDIA사의 GPU 하드웨어 구조에 대하여 기술한다. NVIDIA사는 2006년 G80 구조에 기반한 GeForce 8800을 출시하였는데, 이는 최초로 C언어를 사용하여 단일 명령어-다중 쓰레드 모델을 지원하는 GPU로 내부적으로 128개의 스트리밍 프로세서를 가지고 있었다. 2008년 6월에는 G80 아키텍처를 개선한 2세대 구조인 GT200이, 2009년 9월에는 3세대 구조인 페르미(Fermi) 구조가 소개되었으며, 이에 기반한 제품으로 GeForce GTX480, GTX580 등이 2010년 이후 출시되어 사용 중에 있다.

페르미 아키텍처 기반의 GPU 내에는 최대 512개의 CUDA 코어가 있으며, 이들은 32개씩 하나의 단위를 이루는데 이를 스트림 멀티프로세서(SM; Stream Multiprocessor)라고 한다. SM 안에는 32,768개의 32비트 레지스터와 64KB의 공유 메모리/L1 캐쉬가 있으며, 필요에 따라서 공유 메모리와 L1 캐쉬를 각각 48KB와 16KB 또는 16KB와 48KB로 지정하여 사용 가능하다. 연산에 필요한 데이터를 보관하는 384비트 메모리 인터페이스는 6개의 64비트 메모리로 분할되어 있으며, 최대 6GB GDDR5 메모리까지 가능하다. SM과 GDDR5 메모리 간에는 최대 768KB

L2 캐쉬로 연결되어 있어 메모리 국부성이 있는 응용 프로그램 실행에 최적화되어 있다. GPU와 CPU 간에는 PCI-Express 버스로 연결된다.

2.2 기존 GPU 클러스터

그림 1은 Stony Brook Visual Computing Cluster의 시스템 구조이다[6]. GPU 성능이 2000년 대 들어 비약적으로 발전하면서 CPU에 대한 FLOPS 성능 면에서 더욱 좋아지게 되자 이를 활용하기 위한 연구가 시작되었다. Stony Brook Visual Computing Cluster는 한 개의 Pentium Xeon 2.4GHz CPU와 한 개의 2.5GB 주기억장치, 128MB를 가지는 NVIDIA GeForce FX 5800 Ultra 그래픽스 장치로 구성된 노드 32개를 1Gbps 이더넷 스위치로 연결한 구조이다. Pentium Xeon 2.4GHz CPU와 NVIDIA GeForce FX 5800 Ultra의 단일 정밀도 실수 연산 성능은 각각 10 GFLOPS와 16 GFLOPS이다. 따라서 Stony Brook Visual Computing Cluster의 CPU 클러스터 연산 성능은 최대 320 GFLOPS, GPU 클러스터 연산은 최대 512 GFLOPS이다. 도심의 공기 흐름 시뮬레이션을 통해서 GPU 클러스터의 성능이 CPU 클러스터에 비해서 4.6배 정도 개선됨을 보였다. GPU 성능이 CPU 성능을 넘어서기 시작한 초창기의 GPU 클러스터링이란 점에서 의미가 있다고 할 수 있다.

그림 2는 AC(Accelerator Cluster)의 노드 구조이다[7]. 하나의 계산 노드(compute node)에 두 개의 듀얼코어 2.4GHz AMD Opteron/8GB 주기억장치와 한 개의 NVIDIA Tesla S1070/4GB GDDR3 메모리, H101-PCIX FPGA 가속기(accelerator)를 가지며, 총 32개의 계산 노드를 2GB/sec Infiniband 스위치로

그림 1. Stony Brook Visual Computing Cluster 시스템 구조

그림 2. AC(Accelerator Cluster) 노드 구조

연결한 클러스터 시스템이다. NVIDIA Tesla S1070에는 240개의 코어가 있는 GT200 GPU가 네 개 있어 총 960개의 GPU 코어가 있다. NVIDIA Tesla S1070은 배정밀도 실수 연산이 345.6GFLOPS의 성능을 보이므로 예상되는 AC의 최대 성능은 11.06TFLOPS이다. 실제 시스템의 성능을 평가하기 위하여 High-Performance Linpack(HPL) 벤치마크를 사용한 실험 결과 16노드 AC 클러스터의 배정밀도 성능이 1.7TFLOPS 정도로 나타났으며, 이는 Tesla S1070이 연결된 호스트 노드가 최적의 환경이 아니어서 성능을 감소시키는 요인이 된 것으로 평가되었다.

3. 시스템 구현

3.1 GPU 클러스터 구조

다수의 GPU 그래픽스 장치를 가지는 시스템의 구조를 설계하기 위하여 기존 슈퍼컴퓨터 구조를 벤치마킹하도록 한다. 2010년 11월 발표된 Top 500 슈퍼컴퓨터에 대한 분석 결과[2] 중 시스템 구성에 영향을 많이 주는 시스템 구조와 연결 방식을 살펴본다. 시스템 구조는 클러스터(cluster), MPP(massively parallel processing), 성운형(constellations)으로 구분되며, 각각 83.0%, 16.6%, 0.4%의 사용 비율을 보인다. 클러스터는 비교적 저가의 상용 기성품(COTS; commercial off-the-shelf) 컴퓨터를 여러 대 연결하여 병렬 처리에 활용하는 방식으로 확장성 면에서 다른 방식에 비해 장점이 있어 초기의 Cray X1과 같은 벡터 컴퓨터 방식보다 많이 사용되고 있다.

Top 500 슈퍼컴퓨터에 사용된 연결 방식은 주로 기가비트 이더넷(45%)과 Infiniband 스위치(43%)가 사용되고 있으며, 기타 방식이 12%이다. 이는 2세대,

3세대 다중컴퓨터에서 컴퓨터의 처리 성능을 높이기 위하여 프로세서 간 통신을 맞춤(custom) 하드웨어로 처리하였던 것에 비하여 검증된 상용 네트워크 장비를 활용함으로써 높은 통신 성능과 함께 유지보수 면에서도 유리하기 때문이다.

기존의 슈퍼컴퓨터 구조 분석 결과를 바탕으로 그림 3과 같이 확장하기 쉬운 클러스터 시스템 구조를 사용하여 시스템을 구현한다. 전치(front-end) 컴퓨터와 네 대의 GPU 내장 컴퓨터를 기가비트 이더넷 스위치를 통하여 연결함으로써 클러스터 시스템을 구성하고, 향후 더 많은 GPU 내장 컴퓨터를 연결하여 시스템 성능을 높일 수 있도록 한다.

전치 컴퓨터는 실제 GPU 연산이 일어나는 par1-par4 컴퓨터로부터 연산 결과를 받아서 다른 컴퓨터로 필요한 데이터를 전달해 주고, 실행 결과를 보여주는 역할을 한다. GPU가 있는 par1-par4 컴퓨터는 AMD 페넘II x6 1055T 2.8GHz CPU를 사용하며, 각각 8GB의 주기억장치를 가진다. GPU는 NVIDIA GTX 480 카드를 컴퓨터 내에 2장 연결하여 사용한다. NVIDIA GTX 480은 480개의 CUDA 코어를 내장하고 있으며, 1,401MHz로 동작한다. GPU 내에서 사용가능한 메모리는 1.5GB이며, 메모리 클럭은 1,848MHz로 최대 177.4GB/sec의 메모리 대역폭을 가진다. GTX 480의 단일 정밀도 실수 연산 성능은 FMA(fused multiply-add) 기준으로 최대 1.34 TFLOPS이므로 구현된 GPU 클러스터 시스템의 최대 성능은 10.72 TFLOPS가 될 것으로 예상된다. 그림 4는 구현된 GPU 클러스터 시스템의 사진이다.

3.2 소프트웨어 개발 환경

Top 500 슈퍼컴퓨터에 사용된 운영 체제는 리눅스

그림 3. GPU 클러스터 시스템 구조 개념도

그림 4. 구현된 GPU 클러스터 시스템

스(91.8%), 유닉스(3.8%), 윈도우즈(1.0%) 등이 사용되고 있음을 알 수 있다. 리눅스 계열의 운영체제는 소스가 공개되어 있어 수정이 용이하며, 기존의 운영체제에 비하여 기억장치를 많이 필요로 하지 않으며 CPU에 운영체제에 따른 부하가 비교적 낮아서 연산 처리에 유리한 장점이 있기 때문이다. 따라서 본 논문에서 구현한 GPU 클러스터 시스템의 운영 체제로 리눅스 계열의 Ubuntu 10.10을 사용한다.

실제 GPU를 활용하기 위해서는 OpenCL 1.1 표준[8]을 구현하고 있는 NVIDIA CUDA Toolkit이 필요하다. 본 논문에서는 OpenCL C++ API를 사용하여 보다 간편하게 프로그램을 개발할 수 있도록 OpenCL 응용 프로그램 실행에 필요한 GPU 인식 및 사용 설정, 커널 컴파일 및 GPU 적재 등의 공통적인 실행 과정을 구현하는 OpenCLHelper 추상 클래스를 설계, 구현한다. GPU를 사용하는 응용 프로그램 개발을 하기 위해서는 OpenCLHelper 추상 클래스를 상속받아서 실제 응용 프로그램 구현에 필요한 멤버 함수만 구현하면 된다.

par1-par4 컴퓨터 간에 연산에 필요한 자료를 전송하기 위한 통신 플랫폼으로는 PVM[9]을 사용한다. 분산 컴퓨터 간 통신을 위한 플랫폼으로 CORBA, PVM, MPI 등 여러 가지 경우를 고려할 수 있다. 원격 객체 간 자료 공유와 메소드 실행이 가능하도록 프록시 패턴이 적용된 CORBA를 활용하기 위하여 C++ 사용이 가능한 omniORB를 설치하여 OpenCL 응용 프로그램을 실험적으로 개발, 테스트해 본 결과 원격 객체로 한 번에 전달 가능한 자료 전송 크기가 작아서 효율성 면에서 좋지 않았다. PVM은 이종 분산 시스템에서 메시지 전송을 지원해 주는 소프트웨어 패키지로 가상 기계 개념을 가지고 있어 동적 자원 관리와 프로세스 제어 함수가 풍부하다. xpvmm[10]을 사용하면 PVM 제어 및 실행 결과를 확인하기

용이하여 여러 계산 노드에서 실행되는 프로그램의 상태를 파악하기 쉽다.

4. 성능 평가

본 절에서는 천체 운동을 시뮬레이션 하는 n-body 문제[5]를 사용하여 구현한 GPU 클러스터 시스템의 성능을 평가한다.

4.1 문제 특성 분석

nbody 문제에서 n 개의 천체에 대한 한 번의 위치 벡터와 속도 벡터 변화를 구하기 위한 순차 실행 속도 t_s 는 각각의 천체에서 모든 n 개의 천체에 대하여 계산을 해야 하므로 $2n^2$ 임을 쉽게 알 수 있다.

한 개의 GPU를 사용하여 nbody 문제를 해결하기 위한 병렬 실행 속도 t_p 는 다음과 같은 요소를 가진다. 먼저, 호스트에서 GPU로 위치 벡터와 속도 벡터를 전송해야 하므로 이를 위한 통신 시간 t_{comm1} 은 통신 초기화 시간 $t_{startup}$ 과 n 개의 위치 벡터와 속도 벡터를 GPU로 전송하기 위하여 $2nt_{data}$ 시간이 필요하다. n 개의 위치 벡터와 속도 벡터를 GPU내 p 개의 CUDA 코어로 한 번의 위치 변화와 속도 변화를 계산하기 위한 시간 t_{comp2} 는 $2n^2/p$ 만큼의 시간이 필요하다. 계산된 결과를 호스트에서 다시 보기 위해서는 GPU내에 있는 위치 벡터와 속도 벡터 계산 결과를 다시 호스트로 가져와야 하므로 그에 따른 통신 시간 t_{comm3} 는 t_{comm1} 과 동일하다. 따라서 nbody 문제를 p 개의 코어를 가지는 GPU로 해결하기 위한 병렬 실행 속도 t_p 는 $t_{comm1} + t_{comp2} + t_{comm3}$ 임을 알 수 있다. 이를 정리하면 다음과 같다.

$$t_p = 2(t_{startup} + 2nt_{data}) + 2\frac{n^2}{p} \quad (1)$$

따라서 속도 향상 요소 $\frac{t_s}{t_p}$ [11]는 다음과 같다.

$$\frac{t_s}{t_p} = \frac{2n^2}{2(t_{startup} + 2nt_{data}) + 2\frac{n^2}{p}} \quad (2)$$

식 (2)에서 n 이 커지게 되면 $\lim_{n \rightarrow \infty} \frac{t_s}{t_p} = p$ 임을 알 수 있다. 이는 순차 실행할 때에 비하여 GPU를 사용하여 병렬 처리할 때 GPU내 코어 개수 p 만큼 성능 향

상 요인이 있음을 의미한다.

4.2 메모리 모델

구현한 GPU 클러스터 시스템을 사용하여 n-body 문제를 구현하기 위해서는 계산에 필요한 메모리를 호스트 주소 공간과 GPU 주소 공간에 적절히 분배하여 설계해야 한다. 그림 5는 계산 노드에서 단일 GPU를 사용하여 계산을 할 때의 메모리 모델 NBodyS이다. n 개의 천체에 대한 위치 벡터와 속도 벡터에 대한 정보를 호스트 주소 공간의 hPos와 hVel에 저장하며, 이를 GPU 주소 공간에 있는 bCurrPos, bCurrVel로 전송하여 각 천체의 현재 위치 벡터와 속도 벡터로 사용한다. 이로부터 일정 시간 후 다음 위치 정보와 속도 정보를 계산하여 bNewPos, bNewVel에 저장한다. 다음 계산을 위하여 bNewPos, bNewVel에 있는 정보를 다시 bCurrPos, bCurrVel로 옮겨 계산을 진행하거나, 계산이 종료되면 호스트 주소 공간에 있는 hPos, hVel로 값을 전송하여 사용자가 활용할 수 있도록 한다.

그림 6은 단일 호스트 내에 있는 GPU를 두 개 사용할 때의 메모리 모델을 보여준다. 그림 5의 단일 GPU 사용할 때에 비하여 각 GPU에서 $n/2$ 개의 천체에 대해서만 계산을 하면 되나, 각 GPU에 있는 계산 결과를 호스트 주소 공간에 다시 전송하여 n 개의 천체에 대한 계산 결과를 매번 모아야 하는 점이 다르다. 즉, hPos, hVel의 값을 이동하는 과정이 단계 {1 → 2 → 3}*의 순서로 반복되어 매번 새로운 위치 값을 계산할 때마다 호스트 주소 공간에 있는 위치 벡터, 속도 벡터의 값을 GPU 주소 공간으로 옮기고, 다음 위치, 속도를 계산한 후 그 결과를 다시 호스트 주소 공간으로 모으는 과정이 필요하게 된다. 이는

그림 5. 단일 호스트 내 단일 GPU 사용할 때의 메모리 모델 (NBodyS)

그림 6. 단일 호스트 내 GPU 두 개 사용할 때의 메모리 모델 (NBodyM)

두 개의 GPU에 위치 벡터와 속도 벡터가 분산되어 계산되기 때문이다. 본 논문에서 구현된 GPU 클러스터 시스템을 구성하고 있는 여덟 개의 GPU를 모두 계산에 활용하기 위해서 그림 6과 유사하게 각 GPU에서 계산해야 하는 위치 벡터와 속도 벡터를 적절히 분산시키고 다시 합치는 과정을 반복하도록 설계하여 실험에 활용하였다.

4.3 실험 결과

본 논문에서 구현한 GPU 클러스터 시스템의 성능을 평가하기 위하여 CPU만 사용하여 계산을 하는 경우와 함께 NVIDIA GTX 480 그래픽스 카드 즉, GPU의 개수를 1, 2, 4, 8개로 변화시켜 가면서 계산을 한 경우 등을 다양하게 비교 평가하는 실험을 수행한다. 이때 천체의 개수는 $1024 \cdot 2^i$ ($i = 0, 1, 2, \dots$)으로 하고, 위치 벡터와 속도 벡터를 100회 또는 1000회 갱신하는 시간을 측정하여 평가에 사용한다.

먼저 CPU와 GPU의 성능 비교를 위하여 GPU를 사용하지 않고 CPU에서만 nbody 계산을 하는 경우와 GPU 하나를 사용하여 계산을 하는 경우를 비교 평가한다. GPU와의 성능 평가를 위하여 Intel Core 2 Quad Q9300 2.5GHz CPU를 가지는 컴퓨터에 윈도우즈 운영체제와 우분투 운영체제를 설치하여 동일한 프로그램에 의한 실험 결과를 수집한다. 그림 7은 천체의 개수를 1K~16K로 변화시키면서 1000회 계산을 할 때 GPU와 함께 CPU에 의한 성능 변화를 보여준다. 계산량이 많아질수록 GPU를 사용하는 경우가 CPU를 사용하는 경우에 비하여 성능이 많이 향상됨을 알 수 있다.

그림 7. CPU와 GPU 성능 비교(100회 계산)

그림 8은 성능 변화를 좀 더 세밀하게 관찰하기 위하여 GPU를 사용했을 때의 성능을 1로 하여 다른 경우를 정규화한 것이다. Intel Core 2 Quad Q9300 2.5GHz CPU를 사용하는 경우는 윈도우즈 환경과 함께 리눅스 운영체제인 우분투를 설치하여 동일하게 동작하는 nbody 프로그램을 실행하여 성능 비교를 한다. 실험 결과에서 윈도우즈 운영체제를 사용하는 경우가 우분투 운영체제를 사용하는 경우에 비하여 성능 저하가 많음을 알 수 있다. 이는 사용된 컴파일러의 최적화 관련 특성과 운영체제 실행 환경에 의한 영향으로 생각되나, 이에 대한 분석은 본 논문의 범위를 벗어나므로 생략한다. 페넘 II 2.8GHz CPU에서 우분투 운영체제를 사용하는 경우가 Intel Q9300 2.5GHz CPU를 사용하는 경우보다 성능이 좀더 향상된 것으로 보이나, CPU 클럭 속도를 감안하면 큰 차이는 없는 것으로 보인다. 성능 비교 결과를 종합하면 전반적으로 CPU에서 실행할 때보다 GPU에서 프로그램을 실행하는 것이 최대 240~450배 정도 성능 향상이 있음을 알 수 있다.

그림 9는 호스트 내 GPU를 한 개 또는 두 개 사용한 경우 NBodyS와 NBodyM과 함께 네 대의 호스트

그림 8. 정규화된 CPU와 GPU 성능 비교

그림 9. 클러스터 내 전체 GPU 사용시 성능 비교 (100회 계산)

에 있는 GPU를 각각 한 개 또는 두 개 사용한 NBodyS-PVM4, NBodyM-PVM4의 성능 평가 결과이다. 서로 비교할 수 있도록 NBodyS를 1로 하여 정규화하였으며, 그림에서 알 수 있듯이 천체의 개수 n 이 작을 때는 단일 호스트 내에 있는 GPU를 사용하는 것이 더 좋은 성능을 보이나 천체의 개수 n 이 64K 개를 넘어서면서 NBodyS-PVM4와 NBodyM-PVM4의 성능이 더 좋아짐을 알 수 있다. 이는 병렬 처리하면서 같이 공유해야 하는 데이터를 주고 받는 통신 경비가 계산해야 할 천체의 개수가 작을 때는 전체 실행 시간에서 더 많은 부분을 차지하다가 천체의 개수가 커져 계산량이 많아지면서 반대로 되기 때문이다. 괄호 안의 숫자는 계산에 사용된 GPU의 개수를 의미한다.

단일 GPU를 사용하여 한 번에 계산 가능한 천체의 크기는 최대 256K개까지 가능하다. 이는 GPU 내부에 사용가능한 자원 부족 현상에서 기인한다. 이러한 제약점을 해결하기 위하여 한 번에 계산하는 단위를 전체 위치 벡터와 속도 벡터가 아니라 그보다 작은 블록 단위로 나누어 계산하도록 함으로써 자원

그림 10. $n=1M$ 일 때의 처리 성능 (100회 계산)

부족 문제를 해결한다. 그림 10은 천체의 크기 $n=1M$ 일 때 블록 크기를 다양하게 하여 실험한 결과이다. 실험 대상은 하나의 계산 노드에 있는 GPU 한 개에 대하여 블록 단위 계산 방법을 적용한 NBodyS-large와 각각 두 대와 네 대의 계산 노드에 있는 GPU를 사용하여 실험한 NBodyS-large-PVM2, NBodyS-large-PVM4이다. 실험 결과 적절한 블록 크기는 8K~16K 정도면 충분함을 알 수 있다.

표 1은 GPU 내에서 한 번에 계산하는 단위인 블록 크기를 16K로 고정시켰을 때 천체의 개수 n 을 512K~8M개로 변화시켜 가면서 측정한 실험 결과이다. 단일 GPU를 사용하여 256K개 보다 큰 천체의 움직임 계산할 수 있도록 한 NBodyS-large와 구현된 클러스터 내 네 대의 호스트에서 각각의 GPU를 사용하여 계산하도록 한 NBodyS-large-PVM4의 성능 비교 결과 여러 대의 호스트에서 계산을 나누어 수행한 NBodyS-large-PVM4가 동일한 천체 수에 대하여 더 빨리 계산을 수행함을 알 수 있다. 또한 NBodyS-large의 경우 최대 계산 가능한 천체의 수가 2M로 단일 GPU를 사용하여 NBodyS에서 한 번에 계산할 수 있는 천체의 최대 크기인 256K보다 여덟 배 더 향상되었음을 알 수 있다. NBodyS-large-PVM4의 경우에는 계산을 네 대의 호스트에 있는 GPU에 분산하여 수행하므로 NBodyS-large보다 네 배 더 많은 천체에 대하여 계산 가능하다. 따라서 n -body 문제의 경우 한 번에 계산하는 단위를 적절하게 조절함으로써 더 많은 천체에 대한 계산이 가능하게 됨을 알 수 있다.

표 1. 블록 크기=16K일 때 실행 결과(100회 계산, 단위: 초)

n	NBodyS-large	NBodyS-large-PVM4
512K	34.654	19.497
1M	136.497	72.140
2M	544.653	276.679
4M	N/A	1,101.517
8M	N/A	58,459.064

5. 결 론

GPU를 슈퍼 컴퓨팅 분야에서와 같이 대규모로 사용하는 연구는 이미 활발하게 진행되고 있으나, 이를 일반적으로 접근하여 사용하기에는 비용 상의 문제

가 있다. 이러한 문제점을 해결하기 위하여 본 논문에서는 저가의 그래픽스 장치에 내재된 GPU를 사용하여 비용대비 성능이 높은 시스템을 구현하기 위하여 기존 슈퍼컴퓨터 구조를 벤치마킹하여 시스템 구조를 설계하고, 이를 기반으로 단일 정밀도 실수 연산 성능이 1.34 GFLOPS인 저가의 그래픽스 장치 NVIDIA GTX 480을 여덟 개 사용하여 최대 10.72 TFLOPS 실수 연산이 가능한 GPU 클러스터 시스템을 구현하였다. 또한 구현한 시스템을 위한 소프트웨어 개발 환경을 구축하고, 이중 시스템을 위한 범용 병렬 프로그래밍 개방형 표준인 OpenCL을 사용하여 보다 쉽게 GPU를 활용한 계산이 가능하도록 하였다.

구현된 GPU 클러스터 시스템의 성능을 평가하기 위하여 천체 운동을 시뮬레이션 하는 n-body 프로그램을 구현하고, 천체의 수 n 을 변화시켜 문제 크기를 다양하게 하여 GPU 클러스터 시스템의 성능을 평가하였다. 전반적으로 문제 크기가 작을 때는 단일 호스트에 있는 GPU를 한 개 또는 두 개 사용하는 것이 성능이 좋았으며, 문제 크기가 일정 수준 이상이 되면 여러 대의 호스트를 사용하여 GPU를 활용하는 것이 성능이 좋았다. 본 논문에서는 총 여덟 개의 GPU를 사용하는 시스템을 구현하였는데, 향후 이를 확장했을 때 성능 저하가 없도록 시스템 연결 및 그에 따른 프로그래밍 모델에 대한 연구가 필요하다.

참 고 문 헌

- [1] 전진홍, 이용우, 이명호, “GPU를 활용한 고성능 컴퓨팅,” 정보처리학회지, 제15권, 제5호, pp. 119~125, 2008년.
- [2] Home | TOP500 Supercomputing Sites, <http://www.top500.org/>.
- [3] 이병훈, 이호, 계희원, 신영길, “다중 GPU 기반의 고속 삼차원 역전사 기법,” 멀티미디어학회 논문지, 제12권, 제2호, pp. 209~218, 2009.
- [4] J.S. Park, J.M. Lee, S.Y. Lee, and S.W. Shin, “A Fast 3D Spatial Analysis Techniques Using Graphic Process Units,” in Proceedings of American Society for Photogrammetry and Remote Sensing (ASPRS) 2011 Annual Conference, Milwaukee, 2011.
- [5] Q.-D. Wang, “The Global Solution of the N-Body Problem,” *Celestial Mechanics and Dynamical Astronomy*, Vol.50, No.1, pp. 73-88, 1991.
- [6] Z. Fan, F. Qiu, A. Kaufman, and S. Yoakum-Stover, “GPU Cluster for High Performance Computing,” in Proc. the ACM/IEEE Supercomputing Conference 2004, pp. 47-58, 2004.
- [7] V.V. Kindratenko, J.J. Enos, G. Shi, M.T. Showerman, G.W. Arnold, J.E. Stone, J.C. Philips, and W.-M. Hwu, “GPU Clusters for High-Performance Computing,” in *Proc. Workshop on Parallel Programming on Accelerator Clusters (PPAC'09)*, IEEE International Conference on Cluster Computing, pp. 1-8, 2009.
- [8] A. Munshi (ed.), “The OpenCL Specification: Version 1.1,” Khronos OpenCL Working Group, 2010.
- [9] PVM: Parallel Virtual Machine, <http://www.csm.ornl.gov/pvm/>.
- [10] XPVM: A Graphical Console and Monitor for PVM, <http://www.netlib.org/utk/icl/xpvm/xpvm.html>.
- [11] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications using Networked Workstations and Parallel Computers*, 2nd ed., NJ: Pearson Prentice Hall, 2005, pp. 62-64.

이 종 민

1992년 경북대학교 컴퓨터공학과
공학사
1994년 한국과학기술원 전산학과
공학석사
2000년 한국과학기술원 전자전산
학과 공학박사

1997년~2002년 삼성전자 무선사업부 책임연구원
2005년 Research Associate, Univ. of California at
Santa Cruz
2002년~현재 동의대학교 컴퓨터소프트웨어공학과 부
교수
관심분야: 모바일컴퓨팅, 병렬컴퓨팅, 라우팅

김 성 우

1991년 한국과학기술원 전기 및
전자공학과 공학사
1993년 한국과학기술원 전기 및
전자공학과 공학석사
1999년 한국과학기술원 전기 및
전자공학과 공학박사

1999년~2001년 한국전자통신연구원 선임연구원
2002년~현재 동의대학교 컴퓨터소프트웨어공학과 부
교수
2008년~2009년 California State University at Fresno,
visiting professor
관심분야: 센서 네트워크, 임베디드운영체제, 지능시스
템제어

이 중 화

1992년 2월 부산대학교 전자계산
학과 학사
1995년 2월 부산대학교 전자계산
학과 석사
2001년 8월 부산대학교 전자계산
학과 박사

2002년 3월~현재 동의대학교 컴퓨터소프트웨어공학과
부교수
관심분야: 데이터베이스, XML, 한글정보처리