Journal of The Korea Society of Computer and Information
Vol. 17 No. 5 , May 2012

2012-17-5-1-2

응용프로그램 실행에 따른 CPU/GPU의 온도 및 컴퓨터 시스템의 에너지 효율성 분석

최 홍 준*, 강 승 구*, 김 종 면**, 김 철 홍*

Analysis of the CPU/GPU Temperature and Energy Efficiency depending on Executed Applications

Hong Jun Choi*, Seung Gu Kang*, Jong-Myon Kim**, Cheol Hong Kim*

요 약

전력 소모 증가와 칩 내부 온도 증가라는 문제점들로 인해 동작 주파수 증대를 통해 CPU의 성능을 향상시키는 기법은 점차 한계에 다다르고 있다. 이와 같은 상황에서, CPU의 작업량을 줄여주는 GPU를 활용하는 것은 컴퓨터 시스템의 성능을 향상시키기 위해 사용되는 대표적인 방안 중 하나이다. GPU는 그래픽 작업을 위해 개발된 프로세서로 기존에는 그래픽 작업들만을 전담으로 처리하여 왔지만, CUDA와 같이 GPU 자원을 쉽게 활용할 수 있는 기술이 점차 개발됨에 따라서 GPU를 범용 연산에 활용함으로써 고성능 컴퓨터 시스템을 구현하는 기법이 주목을 받고 있다. 본 논문에서는 다양한 응용프로그램들을 수행하는 경우에 CPU와 GPU가 동시에 활용되는 고성능 컴퓨터 시스템을 목표로, 시스템에서 발생하는 온도와 에너지 효율성을 상세하게 분석하고자 한다. 이를 통해, CPU와 GPU가 동시에 활용되는 컴퓨터 시스템에서 향후 발생 가능한 온도와 에너지 소비 측면에서의 문제점들을 제시하고자 한다. 온도 분석 결과를 살펴보면, GPU를 이용하여 응용프로그램을 수행하는 경우에는 CPU와 GPU의 온도가 동시에 모두 상승하는 것을 할 수 있다. 이와 달리, CPU를 이용하여 응용프로그램을 수행하는 경우에는 GPU의 온도는 거의 변화가 없이 유지되고, CPU의 온도만이 지속적으로 상승한다. 에너지 효율성 측면에서 살펴보면, GPU를 이용하는 것이 CPU를 이용하는 것과 비교하여 동일한 응용프로그램을 수행하는데 있어서 더 적은 에너지를 소비한다. 하지만, GPU는 CPU에 비해 더 많은 전력을 소모하기 때문에 1Wh의 에너지당 발생하는 온도는 CPU에 비해 GPU에서 훨씬 높게 나타난다.

▸ Keyword : CPU, GPU, CUDA, 열섬현상, 에너지 효율성

Abstract

As the clock frequency increases, CPU performance improves continuously. However, power and

• 제1저자 : 최홍준 • 교신저자 : 김철홍
• 투고일 : 2012. 01. 31, 심사일 : 2012. 02. 28, 게재확정일 : 2012. 03. 20.
* 전남대학교 전자컴퓨터공학부(School of Electronics and Computer Engineering, Chonnam National University)
** 울산대학교 전기공학부(School of Electrical Engineering, University of Ulsan)
※ 본 연구는 2011년도 정부(교육과학기술부)의 재원으로 한국연구재단 기초연구사업(2011-0004289)의 지원과 지식경제부 및 정보통신산업진흥원의 대학 IT연구센터 지원사업(NIPA-2012-H0301-12-3005)의 연구결과로 수행되었음

thermal problems in the CPU become more serious as the clock frequency increases. For this reason, utilizing the GPU to reduce the workload of the CPU becomes one of the most popular methods in recent high-performance computer systems. The GPU is a specialized processor originally designed for graphics processing. Recently, the technologies such as CUDA which utilize the GPU resources more easily become popular, leading to the improved performance of the computer system by utilizing the CPU and GPU simultaneously in executing various kinds of applications. In this work, we analyze the temperature and the energy efficiency of the computer system where the CPU and the GPU are utilized simultaneously, to figure out the possible problems in upcoming high-performance computer systems. According to our experimentation results, the temperature of both CPU and GPU increase when the application is executed on the GPU. When the application is executed on the CPU, CPU temperature increases whereas GPU temperature remains unchanged. The computer system shows better energy efficiency by utilizing the GPU compared to the CPU, because the throughput of the GPU is much higher than that of the CPU. However, the temperature of the system tends to be increased more easily when the application is executed on the GPU, because the GPU consumes more power than the CPU.

▸ Keyword : CPU, GPU, CUDA, Hotspot, Energy efficiency

I. 서 론

싱글코어 마이크로프로세서에서는 성능을 향상시키기 위해 주로 동작 주파수를 높이는 방법을 사용한다. 하지만, 공정기술의 발달에 따라 단위 면적에 집적되는 트랜지스터의 수가 급속도로 증가되면서 높은 전력 소모와 칩에서 발생하는 온도 문제로 인해 마이크로프로세서의 동작 주파수를 높이는 것은 한계에 이르고 있다. 이와 같은 상황에서, 마이크로프로세서의 성능을 향상시키기 위해 연구자들은 새로운 패러다임인 멀티코어 프로세서 구조를 제안하게 되었다. 멀티코어 프로세서 구조는 하나의 칩에 여러 개의 코어들을 집적하여 성능과 전력 효율성을 높이는 기법으로 인텔 듀얼/쿼드코어 프로세서와 MIT Raw, Sun Niagra 등이 수 년 전부터 출시되고 있다[1-2]. 하지만, 현재 출시되는 듀얼/쿼드코어 CPU는 데이터 수준의 병렬성을 충분하게 활용하지 못한다는 단점이 존재한다.

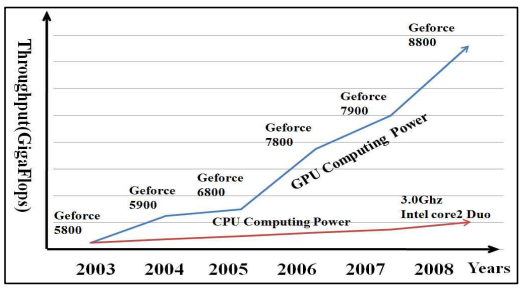

그림 1. CPU와 GPU의 성능 향상 비교
Fig. 1. CPU/GPU throughput comparison

이와 같은 상황에서 멀티코어 CPU 외에 높은 데이터 수준의 병렬성을 통해 연산처리 속도를 매우 높인 GPU는 컴퓨터 시스템의 성능을 개선시킬 수 있는 방안으로 주목을 받고 있다. GPU는 CPU에서 수행하던 그래픽 작업이 문자 위주의 출력에서 그림 위주의 출력으로 변화함에 따라 유발된 병목현상을 해결하기 위해 특별히 고안된 그래픽 처리 전용 프로세서이다. 최근의 컴퓨터 시스템에서는 복잡한 그래픽 처리 작업들을 GPU가 전담하면서 생긴 여유분의 CPU 연산 능력을 다른 작업에 활용함으로써 결과적으로는 전반적인 시스템 성능이 향상되고 있는 추세이다[3-4].

CPU와 GPU의 연산 처리 능력을 비교하는 그림 1에서 보이는 바와 같이, CPU의 성능 향상 폭에 비해 GPU의 성능 향상 폭이 월등하게 높다는 것을 확인 할 수 있다. CPU와 GPU의 연산처리 능력이 크게 차이가 나는 원인은 CPU에는 내부 코어의 개수가 몇 개에 불과한 반면, GPU는 수십 ~ 수백 개의 코어들을 집적하고 있다는 것에 기인한 바가 크다. 많은 수의 코어들을 포함하는 GPU의 하드웨어 병렬성을 적절히 이용한 빠른 연산처리 능력은 그래픽 분야 뿐 아니라, 과학이나 수학 분야, 신호나 이미지 처리 분야, 데이터베이스 가속이나 금융 분석과 같이 다양한 분야에서 사용될 수 있을 것으로 예상된다[5-6]. 이와 같이 GPU의 높은 연산처리 능력을 일반 응용프로그램에서도 사용하려는 기술이 대두되고 있는데 이를 GPGPU (General Purpose computation on the Graphic Processing Units)라고 한다[7]. 초기의 GPGPU는 쉐이딩 언어(shading language)와 OpenGL의

조합으로만 사용할 수 있었기 때문에 개발자에게 그래픽 파이프라인에 대한 높은 이해도를 요구하였다. 이로 인해 상당 수의 개발자들이 GPGPU를 이용하기에는 어려움이 존재해 왔다. 하지만 최근 들어, NVIDIA의 CUDA(Compute Unified Device Architecture), 인텔의 Ct, AMD의 Stream 그리고 OpenCL 등과 같이 GPGPU를 쉽게 이용할 수 있는 기술들이 등장함으로써 이러한 문제들은 점차적으로 해결되고 있다[8-11]. GPGPU 활용 기술들은 GPU를 이용하는 많은 응용프로그램들을 개발하는데 주로 사용되고 있으며, 특히 CUDA는 ANSI C와 C++을 기반으로 설계되었기 때문에 C에 익숙한 개발자들에 의해 널리 사용되고 있다. CUDA와 같이 GPGPU를 쉽게 이용할 수 있는 기술이 점차 개발됨에 따라서 GPU를 이용한 고성능 컴퓨팅 시스템 구현 기법이 다양한 측면에서 개발될 것으로 예상된다. 그러나 CPU와 GPU를 동시에 사용하는 시스템에서 GPU를 그래픽 처리 작업 이외에도 다양하게 사용한다면 성능의 향상을 얻을 수 있는 반면, 필연적으로 발생되는 높은 전력 소모와 높은 온도 등이 시스템의 주된 문제로 대두될 것이 분명하다. 높은 전력 소모 문제와 높은 온도로 인한 열섬 현상(hotspot)은 프로세서가 발전함에 따라서 지속적으로 제기되는 대표적인 문제들 중 하나이다[12].

프로세서 설계 기술이 발달함에 따라 마이크로프로세서 내부의 전력 밀도는 크게 증가하게 되고[13-14], 이는 마이크로프로세서 내부 온도를 상승시켜 열섬 현상을 유발하는 주된 원인이 된다. 열섬 현상은 마이크로프로세서의 작동 오류를 유발하고, 회로의 변형을 발생시켜 칩의 신뢰성(reliability) 및 수명에 부정적인 영향을 미치는 것으로 알려져 있다[15]. 열섬 현상으로 인한 부정적인 영향은 온도 상승에 따른 전자부품에서의 오류 발생률을 살펴보면 좀 더 명확하게 알 수 있다[16]. 온도 상승에 따른 오류 발생률은 지수적으로 증가하기 때문에 마이크로프로세서 설계에 있어서 온도 문제는 매우 심각함을 알 수 있으므로 칩의 신뢰성을 확보하기 위해서는 온도 문제는 반드시 해결되어야 할 것으로 판단된다.

앞서 기술한 바와 같이, 고성능 마이크로프로세서를 설계하는 경우 전력 소모를 감소시키고 열섬 현상을 완화시키는 저전력, 저온도 연구는 주된 이슈가 되고 있다. CPU 뿐만 아니라 GPGPU의 경쟁력 확보를 위해서도 저전력, 저온도 연구는 반드시 진행되어야 할 것으로 판단된다. 이에 따라, 본 논문에서는 저전력, 저온도 GPGPU 시스템을 개발하는 데 있어 기반 자료로 활용될 수 있도록 응용프로그램들을 CPU와 GPU를 활용하여 각각 수행하는 다양한 경우에 대해 시스템의 전력 효율성 및 온도를 분석하고자 한다. 이하 본 논문의 구성은 다음과 같다. 2장에서는 관련 연구로 GPGPU와 CUDA에 관해 설명하고, 3장에서는 본 논문의 실험 환경 및 결과 분석에 대하여 기술하고자 한다. 마지막으로 4장에서는 본 논문의 결론을 기술한다.

II. 관련 연구

1. GPGPU

GPU는 그래픽 처리에 특화되어 있는 프로세서로 2차원 또는 3차원 그래픽을 매우 빠르고 효율적으로 처리한다. 표 1에서 보이는 바와 같이, GPU는 매우 빠르게 발전하여 최근에는 큰 메모리 대역폭과 함께 CPU를 능가하는 연산 능력을 가지고 있을 뿐만 아니라 프로그램이 가능한 유연한 구조를 제공하기 때문에, GPU의 연산능력을 그래픽 처리 이외의 다른 분야에도 활용하기 위한 연구가 활발히 진행되고 있다[17]. 높은 연산 능력을 가진 GPU를 다른 분야에 적용하고자 하는 대표적인 기술이 GPGPU이다[7]. GPGPU는 그래픽 처리만을 담당하던 GPU가 성능이 향상됨에 따라 생긴 여유자원을 이용하여 CPU가 주로 취급하던 범용 작업을 대신하여 수행하도록 하는 기술이다. 기존의 GPU 파이프라인은 각각 고정된 역할을 수행하도록 구성되었지만 GPU 설계 기술이 발전함에 따라서 각각의 파이프라인 단계가 프로그래밍에 유연한 구조로 변화하는 중이다. 예를 들어, 버텍스(vertex) 처리 단계나 프레그먼트(fragment) 처리 단계를 프로그래밍 가능한 버텍스 프로세서나 프레그먼트 프로세서로 구현하여 개발자가 원하는 더 정교한 그래픽 처리를 위해서 직접 프로그래밍을 할 수 있도록 하고 있다.

표 1. 최신 GPU 사양
Table 1. Recent GPU specifications

	Radeon HD	Geforce GTX
출시일	2012.01	2010.11
공정	28nm	40nm
트랜지스터	4312x10e6	3000x10e6
코어 주파수	800MHz	772MHz
메모리 주파수	1250MHz	4008MHz
GFLOPS	2867.2	1581.1

또한, 프로그래밍 가능한 유닛들은 그래픽 연산뿐만 아니라 다른 연산을 하도록 프로그래밍할 수 있도록 설계되어 있다. 각각의 픽셀을 프레임 버퍼에 저장하기 위한 연산을 수행하는 프레그먼트 프로세서를 활용하면 병렬 연산을 더욱 효과적으로 처리할 수 있기 때문에 Krüger에 의해 선형 대수를

그래픽 프로세서를 이용해 해결할 수 있는 방법이 제시된 바 있다[18]. 이 외에도 몇몇 주요한 데이터베이스 연산과 생물정보학(bioinformatics) 알고리즘이 GPU의 처리능력과 유연성을 효과적으로 활용해 구현되고 있다[19-20].

2. CUDA

GPGPU를 활용하기 위해 OpenGL이나 DirectX와 같은 API가 존재하였으나, 이들을 이용하여 GPGPU 응용프로그램을 개발하기에는 많은 어려움이 존재한다. GPGPU 응용프로그램 개발을 보다 손쉽게 하기 위해 고급 언어로 GPU를 활용하는 CUDA, GLSL, Brook, Cg, HLSL 기술들이 개발되고 있다.

표 2. GPGPU 개발 플랫폼 비교
Table 2. GPGPU development platform

회사	개발 플랫폼	특징
NVIDIA & Microsoft	Cg	C언어를 기반으로 GPU를 프로그래밍하기 쉬운 자료형이 추가된 상위 레벨의 셰이딩 언어
	HLSL	버텍스, 지오메트리, 픽셀 셰이더의 프로그램 형식을 가지고 있는 다이렉트 3D API에 사용되는 셰이딩 언어
NVIDIA	CUDA	GPU에서 수행하는 병렬처리 알고리즘을 C 프로그래밍 언어를 비롯한 산업 표준 언어를 사용하여 작성할 수 있도록 도와주는 대표적인 GPGPU 기술
Apple	OpenCL	CPU, GPU, 기타 프로세서로 이루어진 다중 플랫폼을 구동하기 위한 데이터와 작업 병렬성을 갖는 프로그램을 작성할 수 있는 C99에 기반한 개방형 범용 병렬 컴퓨팅 프레임워크

특히, NVIDIA의 CUDA는 C/C++언어를 기반으로 개발되어 다른 기법들과 비교하여 보다 쉽게 활용할 수 있어 널리 사용되고 있다[6]. CUDA 프로그래밍 플랫폼은 2007년 NVIDIA에서 GPGPU를 이용한 병렬 컴퓨팅 소프트웨어를 쉽게 개발할 수 있도록 개발된 플랫폼이다. 개발자들이 많이 사용하는 C언어를 기반으로 설계된 CUDA는 GPU의 병렬처리 알고리즘을 사용하여 범용 응용프로그램을 작성할 수 있도록 도와주는 GPGPU 기술이다. CUDA를 통해 개발자들은 GPU 안의 병렬 계산 요소 고유의 명령어 집합과 메모리에 접근할 수 있으며, CPU와 달리 병렬처리를 할 수 있는 다수의 코어를 가지는 구조를 가진 GPU를 활용하여 여러 개의 스레드를 동시에 실행시킬 수 있다. 특히, 수행하고자 하는 응용프로그램이 병렬처리 연산에 적합한 경우에, CUDA를 이용한 GPU를 활용은 보다 큰 성능 향상을 가져 온다.

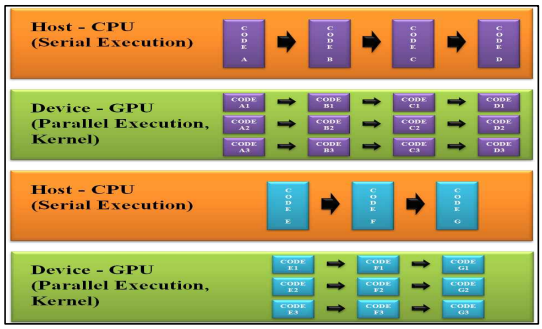

그림 2. CUDA를 이용하는 응용프로그램 실행 패턴
Fig. 2. Application execution pattern to exploit CUDA

CUDA를 이용하여 응용프로그램을 개발하고자 하는 경우에는 CUDA로 작성된 병렬 커널(parallel kernel)을 호출하는 형태로 프로그래밍을 해야 한다. 달리 말하면 GPU를 디바이스(device)라고 한다면 디바이스를 수행시킬 수 있는 호스트(host)가 반드시 필요하다는 의미이다. CUDA를 이용한 응용프로그램의 실행 패턴은 그림 2에서 보이는 바와 같이, 호스트(CPU)의 순차 코드(serial code)와 디바이스(GPU)의 병렬 코드(parallel code)가 순차적으로 반복된다. 그러므로 CUDA를 이용한 병렬처리는 호스트가 순차 코드를 수행하고 디바이스의 병렬 커널을 통해 처리할 작업을 디바이스의 여러 스레드를 호출하여 한 번에 병렬 처리하도록 프로그래밍 하는 것이 일반적이다.

그림 3은 행렬의 각 원소에 일정 수를 더하는 작업을 CPU 프로그램과 GPU 프로그램으로 각각 구현하여 비교하고 있다. GPU 코드에서 "__global__"로 표시되는 함수가 커널 함수인데 이 함수는 디바이스인 GPU의 다수의 스레드에 의해서 수행된다. 동시에 연산을 하는데 필요한 데이터는 스레드의 ID를 통해 각각의 스레드가 처리할 데이터를 지정하게 된다. 즉, GPU 프로그램 코드는 CPU 프로그램의 반복문을 풀어서 각 스레드에 의해 동시에 수행되도록 하는 것이다.

CPU 프로그램

```
void increment_cpu(float *a, float b, int N)
{
        for(int idx = 0; idx<N;idx++)
                a[idx]=a[idx]+b;
}

void main()
{
        ......
        increment_cpu(a,b,16);
}
```

GPU 프로그램

```
__global__ void increment_gpu(float *a, float b, int N)
{
        int idx = blockIdx.x * blockDim.x + threadIdx.x;
        a[idx]=a[idx]+b;
}

Void main()
{
        ......
        increment_gpu<<<4,4>>>(a,b,16);
}
```

그림 3. CPU와 GPU를 활용하는 코드
Fig. 3. Code to exploit CPU/GPU

그림 3에서 볼 수 있듯이, “blockIdx”, “blockDim”, 그리고 “threadIdx”라는 변수를 통해서 각 스레드가 처리할 데이터를 구분하는 것을 확인할 수 있는데 이는 전형적인 SIMD 프로그램에 해당된다. 실제로 메모리의 구조를 활용하여 그리드(grid), 블록(block) 그리고 스레드(thread)로 프로그램을 구성하는 것이 CUDA 프로그램에서 가장 중요한 부분이 된다. CUDA는 GPU의 메모리를 온칩(on-chip)과 오프칩(off-chip)으로 나누어 관리한다. 메모리의 관리는 프로그램의 실행 방식을 결정하는 그리드, 블록, 스레드의 구성에 크게 영향을 미치기 때문에 상황에 맞는 적절한 메모리 선택은 성능 향상에 큰 도움이 된다. 그리고 GPU는 CPU에 비해서 동작의 제어보다는 연산 위주로 설계되어 있기 때문에 동작 제어를 위한 명령이 많은 응용프로그램의 수행에 대해서는 비효율적일 것으로 보인다.

III. 실험 환경 및 결과 분석

1. 실험 환경

본 논문에서 목표로 하는 프로그램을 수행하는 프로세서의 종류에 따른 컴퓨터 시스템의 온도 및 에너지 효율성에 대한 특성을 분석하기 위한 실험 환경은 표 1에 나타낸 바와 같다.

표 3. 실험 환경
Table 3. Experimental environment

실험 환경 요소	장치 명
CPU	Intel Core 2 Duo
GPU	NVIDIA Geforce 8500GT
OS	Ubuntu
전력 측정 장비	Inspector2

본 논문의 실험 대상 CPU인 Intel Core2 Duo는 인텔에서 출시한 x86 구조의 마이크로프로세서로 높은 주파수를 통한 성능 향상 보다는 다중 명령어 수행을 제공함으로써 성능 및 전력 효율성을 향상시키고 발열 문제를 해결하는 구조를 기반으로 설계된 제품이다. 실험 대상 GPU인 NVIDIA Geforce 8500GT는 통합형 셰이더를 적용한 제품으로 16개의 셰이더를 통합하여 하나의 스트리밍 프로세서로 구성하고 연산 종류에 따라 픽셀과 버텍스 작업을 동적으로 셰이더에 분배하여 작업을 수행한다. 내부 텍스처 처리 성능을 향상시키기 위해서는 텍스처 주소와 필터링 유닛을 각각 8개씩 가지고 있다. 뿐만 아니라, 픽셀 처리 능력을 향상시키기 위하여 레스터 동작 부분에서도 4개의 픽셀을 동시에 처리할 수 있도록 구성되어 있으며 128 bit의 메모리 인터페이스를 가지고 있기 때문에 빠른 데이터 처리가 가능하다.

표 4. 벤치마크
Table 4. Benchmark

벤치마크 종류	벤치마크 명	설명
CUDA SDK	Matrix Transpose	고성능 전치 행렬 연산 프로그램
	Separable Convolution	분할 가능한 컨볼루션 필터가 간략하게 구현된 프로그램
	Histogram	64/256빈(bin)의 히스토그램을 효과적으로 구현한 프로그램
Parboil	SAD	MPEG 비디오 인코더(H.264)를 사용하여 다른 커널을 합을 구하는 프로그램
	CUTCP	3차원 그리드 위의 각 그리드 포인트에서 콜롬빅 포텐셜(coulombic potential)의 근거리 컴포넌트들을 계산하는 프로그램

벤치마크 프로그램으로는 NVIDIA 홈페이지에서 CUDA를 이용해 구현되어 있는 많은 종류의 CUDA SDK 프로그램 중 3개의 프로그램(Matrix Transpose, Separable Convolution, Histogram)과 Parboil 벤치마크들 중 2개(CUTCP, SAD)를 선택하여 수행한다[21-22]. 벤치마크 프로그램에 대한 상세한 설명은 표 4에 기술한다.

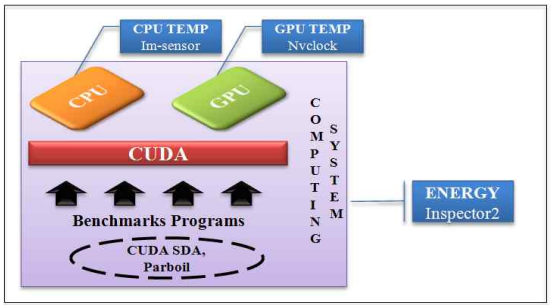

그림 4. 실험 환경 개요
Fig. 4. Experimental environment

그림 4에서 보이는 바와 같이, 실험에서는 벤치마크 프로그램들을 CUDA를 활용하여 CPU와 GPU에서 각각 수행하고, 이에 따른 시스템의 온도 및 에너지 효율성을 측정한다. 실시간으로 CPU와 GPU의 온도를 측정하기 위하여 리눅스 환경 하에서 CPU의 온도 정보를 포함한 다양한 상태 정보를 제공하는 lm-sensor 유틸리티와 GPU의 현재 온도 정보를 출력해주는 Nvclock 프로그램을 매 초마다 주기적으로 수행한다[23]. 또한 프로그램의 수행에 따라 소비되는 에너지를 측정하기 위해서는 실시간 전력 및 에너지를 측정할 수 있는 실험 장비인 Inspector2를 사용한다.

CPU/GPU 온도 측정 프로그램

```
while(1){
    cpu_temp = sensors;
    gpu_temp = nvclock –T –f;

    printf(cpu_temp, gpu_temp)
}
```

2. 결과 분석

2.1 프로그램 수행에 따른 온도 변화

그림 6~7은 표4에서 설명한 5가지의 벤치마크 프로그램들을 GPU와 CPU에서 수행하는 경우 발생되는 온도 변화를 수행 시간에 따라 상세하게 보여준다. 그림 6-7에서 세로 축은 GPU와 CPU의 온도 값(℃)을 나타내며, 가로 축은 응용 프로그램을 수행하는 시간을 초 단위로 나타내고 있다. 이 때, 가로 축의 수치는 수행하는 프로그램과 프로세서에 따라서 수행시간이 달라지기 때문에 각 그림마다 다른 값을 표시하고 있다. 그림에서 보이듯이, 프로그램 수행 시간은 일반적으로 GPU를 사용하는 경우가 CPU를 사용하는 경우에 비해 상당히 줄어드는 것을 확인할 수 있다. 그래프 범례에서 G는 GPU의 온도를 나타내고, C1과 C2는 CPU 각 코어의 온도를 나타낸다. 그리고 그림에서는 수행된 벤치마크 프로그램을 표기 하고자 하는 경우 그대로 사용하나 벤치마크 프로그램 명이 긴 Matrix Transpose와 Separable Convolution는 transpose와 convolution으로 각각 표기한다.

우리는 본 절에서 앞서 언급한 5가지의 프로그램을 CPU와 GPU에서 각각 수행하는 경우의 온도 패턴을 살펴보고자 한다. 그림 6은 CPU에서 응용 프로그램을 수행하는 경우의 온도 패턴을 보여주며, 그림 7은 GPU에서 응용 프로그램을 수행하는 경우의 온도 패턴을 보여주고 있다. CPU의 온도를 분석해 보면, 프로그램이 수행되는 동안에 CPU의 온도는 지속적으로 상승(transpose: 22℃, histogram: 20℃, convolution: 28℃, sad: 20℃, cutcp: 18℃)한다. 본 논문에서는 듀얼 코어를 사용하는데, 첫 번째 코어(C1)의 온도가 두 번째 코어(C2)의 온도에 비해 많이 상승(transpose: 11℃, histogram: 14℃, convolution: 16℃, sad: 15℃, cutcp: 12℃)하는 것을 볼 수 있다. CPU 내부의 코어들 사이의 업무 분배가 동일하게 이루어지지 못하는 것이 C1과 C2의 온도 상승 폭이 상이하게 나타나는 원인으로 분석된다. 그리고 GPU의 온도는 꾸준히 상승하는 CPU와는 달리 일정하게 유지되는 것을 확인할 수 있다. CPU에서 작업을 처리하는 동안 GPU는 대기 상태에 있기 때문에 온도가 상승하지 않는 것으로 분석된다.

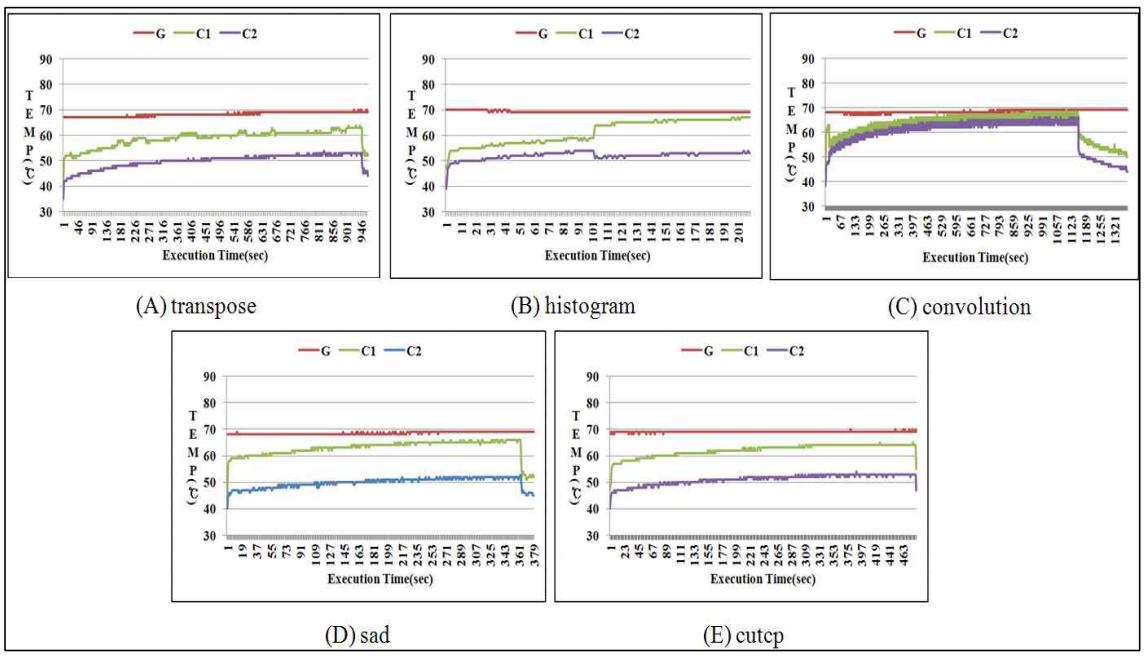

그림 6. 온도 변화 (CPU 작업)
Fig. 6. Temperature change per second (CPU operation)

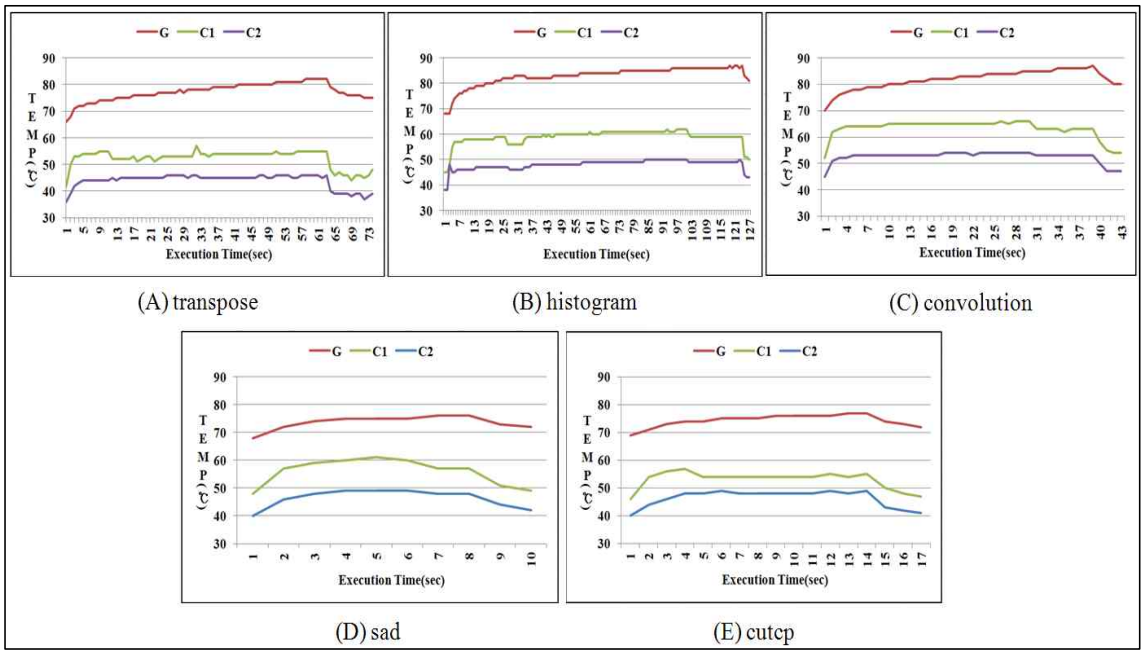

그림 7. 온도 변화(GPU 작업)
Fig. 7. Temperature change per second (GPU operation)

GPU에서 프로그램을 수행하는 경우 온도 패턴을 살펴보면, CPU와 마찬가지로 프로그램이 수행되는 동안에는 GPU의 온도가 지속적으로 상승(transpose: 16℃, histogram: 19℃, convolution: 17℃, sad: 8℃, cutcp: 8℃)하는 것을 그림 7에서 보여주고 있다. 하지만 CPU에서 프로그램을 수행하는 경우와는 달리, GPU에서 프로그램을 수행하는 경우에는 CPU의 온도 또한 소폭 상승(transpose: 15℃, histogram: 17℃, convolution: 14℃, sad: 13℃, cutcp: 11℃)하는 것을 확인 할 수 있다. 이는 현재 상용화된 GPU는 프로그램을 독립적으로 수행하지 못하고 외부입출력 장치들과의 데이터 전달 등에 있어서 CPU의 도움을 받기 때문인 것으로 분석된다. 즉, GPU에서 작업을 처리하는 동안 CPU 또한 작업을 수행하기 때문에 CPU의 온도 또한 상승한다. 결과적으로 프로그램을 수행하는 있어서 CPU를 활용하는 경우는 CPU의 온도만 상승하지만, GPU를 활용하는 경우에는 CPU 또한 작업을 수행하므로 CPU와 GPU의 온도가 모두 상승하게 되는 것을 알 수 있다.

수행된 각 벤치마크 프로그램들은 작업량과 병렬성에 따라 수행시간이 상이하게 나타난다. CPU에서 벤치마크 프로그램을 수행한 경우 온도 변화를 보여주는 그림 6을 살펴보면, 프로그램 수행을 완료하기까지 transpose는 946초, histogram은 201초, convolution은 1,321초, sad는 379초 그리고 cutcp는 463초가 소모된다. 다시 말하면, 수행 시간을 기준으로 각 벤치마크 프로그램의 시스템 자원 소비율을 분석한다면 convolution, transpose, cutcp, sad, histogram 순으로 나타난다. 반면에, GPU에서 벤치마크 프로그램을 수행한 경우 온도 변화를 보여주는 그림 7을 살펴보면, 프로그램 수행을 완료하기까지 transpose는 73초, histogram은 127초, convolution은 43초, sad는 10초 그리고 cutcp는 17초가 소모된다. 위의 경우와 동일한 방법으로 각 벤치마크 프로그램의 시스템 자원 소비율을 분석한다면, histogram, transpose, convolution, cutcp, sad 순으로 나타난다. CPU와 GPU에서 프로그램을 수행하는 경우 자원 소비율이 매우 상이하게 나오는 이유는 각 벤치마크 프로그램의 병렬성 때문으로 분석된다. 달리 말하면, convolution의 경우 병렬성이 매우 높기 때문에 수행시간이 1,321초에서 43초로 감소율이 매우 높은 반면, histogram은 병렬성이 낮기 때문에 CPU 대신 GPU를 활용하더라도 201초에서 127초로 수행시간 감소율이 매우 낮다. 실험 결과에서 보이는 자원 소비율의 변화에 따르면 병렬성이 높아 GPU에 보다 적합한 벤치마크 프로그램은 convolution, transpose, cutcp, sad 그리고 histogram 임을 알 수 있다.

실험 결과에서 볼 수 있듯이, GPU에서 프로그램이 실행될 때 발생하는 온도 상승 폭에 비해서 CPU의 온도 상승 폭이 더 큰 것을 확인할 수 있는데 이는 크게 두 가지 이유로 분석된다. 첫 번째는 현재의 온도가 높으면 온도가 상승하는데 있어서 물리적으로 더욱 많은 에너지를 필요로 하는데 본 실험에서는 GPU의 초기 온도가 CPU의 초기 온도에 비해 상당히 높기 때문이다. 두 번째는 CPU를 사용하는 경우에는 긴 수행시간으로 인해 더 많은 에너지를 소비하기 때문으로 분석된다. 대부분의 최신 프로세서들은 각각 냉각 시스템을 가지고 있어 온도를 제어한다. 하지만, 많은 연산으로 인해 특정 임계 온도에 도달하게 되면 온도를 제어하기 위한 온도 제어 회로가 동작한다. 온도 제어 회로는 온도를 감소하기 위하여 성능을 저하시킨 다는 단점이 존재하기 때문에 적절한 온도 제어가 필요하다고 판단된다. 본 논문에서 수행한 온도 분석 내용을 활용한다면 보다 적절하게 온도를 제어할 수 있으리라 기대된다.

2.2 프로그램 수행에 따른 에너지 효율성 분석

응용 프로그램을 수행하는 데 소비되는 에너지는 응용 프로그램을 수행하기 시작하는 순간부터 종료되는 시점까지 소모하는 전력의 총합으로 계산된다. 그러므로 프로그램 수행에 따른 에너지 소비량은 아래의 식 (1)로 구할 수 있다.

$$\mathbf{E} = \sum_{T_E}^{T_S} \mathbf{P} \qquad (1)$$

식 (1)에서 P는 컴퓨팅 시스템의 소모 전력을 나타내고 TS와 TE는 응용 프로그램 수행 시작 시간과 종료 시간을 나타낸다. 본 절에서는 식 (1)을 활용하여 각 응용 프로그램 수행에 따른 컴퓨팅 시스템의 에너지 소비량을 측정한다.

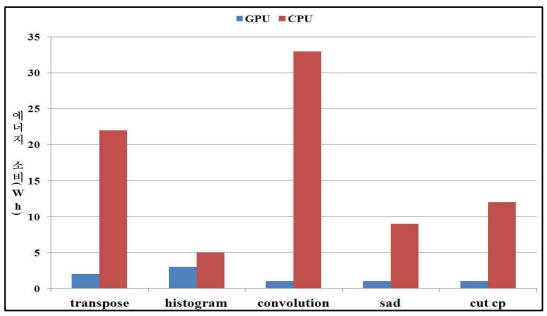

그림 8. CPU와 GPU의 에너지 소비량 비교
Fig. 8. CPU/GPU energy consumption comparison

그림 8의 가로축은 수행한 응용프로그램의 종류를 나타내고, 세로축은 CPU와 GPU에서 소비한 에너지를 Wh의 단위로 표현하고 있다. 동일 응용프로그램을 수행한 경우 GPU에 비해 CPU가 에너지를 최대 22.44배(평균:8.19배, transpose:8배, histogram:1.58배, convolution:22.44배, sad:3.6배, cut cp:5.33배) 많이 소비하는 것을 그림 8에서 보여주고 있다.

그림 8에서 볼 수 있듯이, 에너지 소비량 감소율은 convolution이 가장 크고 histogram이 가장 적게 나타난다. 이와 같은 결과는 에너지 소비량 감소폭이 2.1장에서 분석한 벤치마크 특성에 따른 수행 시간 감소율과 매우 밀접한 관계를 맺고 있음을 명확하게 보여준다. 다시 말하면, CPU와 GPU의 에너지 소비량의 차이가 큰 원인으로는 수식 (1)에서 나타내듯이, 수행 시간(TE-TS)이 길어지면 길어질수록 에너지 소비량이 증가하기 때문으로 분석된다. 그러므로 대부분의 응용프로그램에 있어서 많은 수의 코어들을 이용하여 병렬적으로 작업을 처리하기 때문에 수행시간이 CPU와 비교하여 월등히 짧은 GPU가 에너지를 적게 소모한다. 다시 말하면, 특별한 경우를 제외하고는 GPU를 이용하게 되면 더 적은 에너지를 소비하면서 작업을 처리할 수 있기 때문에 GPU는 CPU에 비해 에너지 효율성 측면에서 보다 좋다고 할 수 있다.

2.3 에너지당 온도 상승률

본 논문에서는 온도와 에너지 효율성 이외에, 에너지 소비에 따른 프로세서의 온도 상승률 또한 분석하고자 한다. 앞선 실험 결과에서 온도 상승 폭과 전력 소모량 모두 CPU가 GPU에 비해서 높다는 것을 확인한 바 있다. 그러므로 에너지당 온도 상승률은 쉽게 예상할 수 없으며 온도 제어를 위한 연구에서 중요한 데이터로 활용 될 것으로 예상된다.

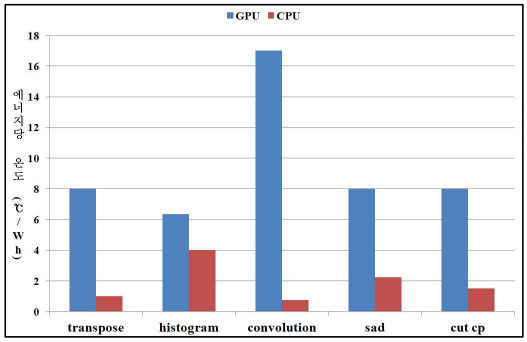

그림 9. 1Wh의 에너지 소비에 따른 온도 변화량
Fig. 9. Temperature variation depending on 1Wh energy consumption

그림 9에서, 온도 상승률은 1Wh의 에너지로 상승하는 온도 값으로 표현한다. 1Wh의 에너지로 상승하는 온도를 살펴보면, CPU의 온도 상승률이 GPU의 상승률에 비해서 낮은 값을 보여준다. GPU를 이용하여 응용프로그램을 수행하면 적은 에너지를 소모하고 온도 상승 폭이 낮았기 때문에 에너지당 온도 상승률이 CPU와 그리 큰 차이를 보이지 않을 것으로 판단하였다. 하지만, 원인을 분석한 결과 GPU는 훨씬 많은 연산을 처리하기 때문에 단위 시간당 전력 소모율이 CPU에 비해 월등하게 높아 에너지당 온도 상승률 또한 상대적으로 매우 높음을 알 수 있다. 달리 말하면, 같은 에너지를 소모하는 경우 GPU는 CPU에 비해서 훨씬 높은 온도가 발생할 것이다. 그러므로 에너지당 온도 상승률은 GPU의 자원을 보다 효과적으로 활용하는 convolution이 가장 높고 histogram이 가장 낮게 나타난다. 따라서 GPU를 활용하여 응용프로그램을 수행하는 경우 연산 능력과 에너지 소비 측면에서 뛰어나다고 해서 무조건적으로 GPU만을 장시간 사용한다면 GPU 온도는 CPU에 비해서 급격하게 상승할 것임을 예상할 수 있기 때문에 시스템의 안정성을 위해서는 이를 반드시 고려해야만 한다.

수행되어야 하는 작업을 CPU 또는 GPU에 효율적으로 할당하기 위한 기존의 연구들을 살펴보면, 가능한 GPU에 업무를 할당하는 기법, 보다 빨리 작업을 완료할 수 있는 처리장치에 할당하는 기법, 기존의 작업을 먼저 완료한 처리장치에 할당하는 기법 등 주로 성능만을 고려한 역할분담 기법을 제안하고 있다[24]. 본 논문에서 제시된 실험 결과를 활용하여 CPU와 GPU의 처리 비율 최적화 방안을 제시한다면, 수행되어야 하는 작업을 CPU 또는 GPU에 할당하는 경우, CPU와 GPU의 현재 상태(성능, 에너지, 에너지당 온도 상승률 등)와 할당된 작업이 CPU와 GPU에서 수행되는 경우 예상되는 상태를 동시에 다각적으로 고려하여 시스템의 효율성을 최적화할 수 있도록 업무를 분배하는 것이 바람직할 것으로 판단된다.

IV. 결론

CPU와 GPU를 모두 사용하는 컴퓨터 시스템에서는 성능뿐만 아니라, CPU와 GPU의 사용에 따른 온도 변화와 에너지 효율성 또한 고려되어야 할 요소이다. 본 논문에서는 응용프로그램을 수행하는 프로세서의 종류에 따른 프로세서의 온도와 에너지 효율성을 분석하였다. 온도 분석 결과, GPU를 이용하여 프로그램을 수행하는 경우에는 CPU를 배제하고 GPU 단독으로 동작할 수 없기 때문에 두 프로세서의 온도가 모두 상승하게 된다. 그리고 CPU를 이용하여 작업을 수행하는 경우에는 GPU의 온도는 거의 변화가 없이 유지되는 반면 CPU의 온도는 프로그램의 특성에 따라서 상승하게 되는 것을 확인할 수 있었다. 에너지 효율성 측면에서는 에너지가 수행 시간에 비례하기 때문에 빠른 처리 속도를 보이는 GPU를 이용하는 것이 더 적은 에너지를 소비해 동일한 작업을 완료하는 것을 확인할 수 있었다. 하지만, 1Wh의 에너지당 발생하는 온도는 더 적은 에너지를 소비하는 GPU가 CPU에 비해서 높게 발생한다. 이와 같은 본 논문의 분석 결과를 활용하여 CPU와 GPU를 적절히 선택하여 연산을 처리한다면 컴퓨팅 시스템의 효율성 측면에서 최적의 결과를 얻을 수 있을 것으로 기대된다.

참고문헌

[1] M. B. Taylor, J. Psota, A. Saraf, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, A. Agarwal, W. Lee, J. Miller, D. Wentzlaff, I. Bratt, B. Greenwald, H. Hoffmann, P. Johnson, and J. Kim, “Evaluation of the raw microprocessor: An exposed-wire-delay architecture for ilp and streams,” In Proceedings of International Symposium on Computer Architecture, pp. 2-13, 2004.

[2] P. Kongetira, K. Aingaran, and K. Olukotun, “Niagara: A 32-way multithreaded sparc processor,” IEEE Micro, Vol. 25, Issue. 2, pp. 21-25, Mar.-Apr., 2005.

[3] T. Akenine-Möller, E. Haines, and N. Hoffman, “Real-Time Rendering(2nd edition),” AK PETERS, 2002.

[4] K. Gray, “The Microsoft DirectX 9 Programmable Graphics Pipeline,” Microsoft Press, 2003.

[5] B. He, K. Yang, R. Fang, M. Lu, N. Govindaraju, Q. Luo, and P. Sander, “Relational joins on graphics processors,” In Proceedings of International Conference on Special Interest Group on Management Of Data, pp. 511-524, 2008.

[6] I. Buck, “Gpu computing with nvidia cuda,” In Proceedings of International Conference on Special Interest Group on Computer Graphics

and Interactive Techniques(SIGGRAPH), pp. 6, 2007.

[7] GPGPU, Available at http://gpgpu.org

[8] NVIDIA CUDATM Programming Guide Version 2.3.1, Nvidia Corporation, 2009.

[9] A. Ghuloum, E. Sprangle, J. Fang, G. Wu, and X. Zhou, "Ct: A flexible parallel programming model for tera-scale architectures," White paper, Intel Corporation, 2007.

[10] Technical Overview, ATI Stream Computing, AMD Inc., 2009.

[11] OpenCL, Available at http://www.khronos.org/opencl/

[12] J. H. Choi, J. H. Kong, E. Y. Chung, and S. W. Chung, "A Dual Integer Register File Structure for Temperature-Aware Microprocessors," Journal of KISS A Computer System and Theory, Vol. 35, No. 11-12, pp.540~551, Dec., 2008.

[13] J. H. Kong, and S. W. Chung, "Recent Thermal Management Techniques for Microprocessors," Communications of KIISE, Vol. 27, No. 11, pp. 72~79, Nov., 2009.

[14] F. Pollack, "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies," International Symposium on Microarchitecture keynote speech, 1999.

[15] P. Dadvar, and K. Skadron, "Potential thermal security risks," In Proceedings of the IEEE/ASME Semiconductor Thermal Measurement, Modeling, and Management Symposium(SEMI-THERM), pp. 229 - 234, 2005.

[16] J. H. Jeong, "Heat-radiant and Cooling Device of Central Processing Unit and Peripheral devices," Journal of Korea Intellectual Patent Society, Vol. 8, No. 4, pp. 33-43, Dec., 2006.

[17] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A. E. Lefohn, and T. J. Purcell, "A Survey of General-Purpose Computation on Graphics Hardware," Euro-graphics 2005, State of the Art Reports, pp. 21-51, 2005.

[18] J. Krüger and R. Westermann, "Linear algebra operators for gpu implementation of numerical algorithms," ACM Transactions on Graphics, Vol. 22, No. 3, pp. 908-916, Jul., 2003.

[19] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin, and D. Manocha, "Fast computation of database operations using graphics processors," In Proceedings of International Conference on Special Interest Group on Computer Graphics and Interactive Techniques(SIGGRAPH), pp. 215-226, 2004.

[20] W. Liu, B. Schmidt, G. Voss, and W. Muller-Wittig, "Streaming algorithms for biological sequence alignment on gpus," IEEE Transactions on Parallel and Distributed Systems, Vol. 18, No. 9, pp. 1270-1281, 2007.

[21] NVIDIA SDK, Available at http://developer.download.NVIDIA.com/compute/cuda/sdk/website/samples.html

[22] Parboil Benchmark suite, Available at http://impact.crhc.illinois.edu/parboil.php

[23] NVClock, Available at http://www.linuxhardware.org/nvclock/

[24] V. Jimenez, L. Vilanova, I. Gelado, M. Gil, G. Fursin and N. Navarro, "Predictive runtime code scheduling for heterogeneous architectures," In Proceedings of the 4th International Conference on High Performance Embedded Architectures and Compilers, pp.19-33 , 2009

저 자 소 개

최 홍 준

2009: 전남대학교 전자컴퓨터공학부 공학사
2011: 전남대학교 전자컴퓨터공학과 석사
2011-현재: 전남대학교 전자컴퓨터공학과 박사과정
관심분야: 저전력 설계, 고성능 컴퓨팅, 병렬처리, 컴퓨터 구조
Email :chj6083@gmail.com

강 승 구

2010: 전남대학교 전자컴퓨터공학부 공학사
2012: 전남대학교 전자컴퓨터공학과 석사
관심분야: 임베디드 소프트웨어, 컴퓨터 구조
Email :freexz84@gmail.com

김 종 면

1995: 명지대학교 전기공학사
2000: University of Florida ECE 석사
2005: Georgia Institute of Technology ECE 박사
2005 - 2007: 삼성종합기술원 전임연구원
2007 - 현재: 울산대학교 전기공학부 교수
관심분야 : 임베디드 SoC, 컴퓨터 구조, 프로세서 설계, 병렬처리
Email : jmkim07@ulsan.ac.kr

김 철 홍

1998: 서울대학교 컴퓨터공학사
2000: 서울대학교 대학원 컴퓨터공학부 석사
2006: 서울대학교 대학원 전기컴퓨터공학부 박사
2005 - 2007: 삼성전자 반도체총괄 SYS.LSI사업부 책임연구원
2007 - 현재: 전남대학교 전자컴퓨터공학부 교수
관심분야 : 임베디드시스템, 컴퓨터 구조, SoC 설계, 저전력 설계
Email : chkim22@chonnam.ac.kr