http://www.jsebs.org
ISSN: 2288-3908

The Journal of Society for e-Business Studies
Vol.23, No.1, February 2018, pp.37-45
https://doi.org/10.7838/jsebs.2018.23.1.037

GCN 아키텍쳐 상에서의 OpenCL을 이용한 GPGPU 성능향상 기법 연구

A Study on GPGPU Performance Improvement Technique on GCN Architecture Using OpenCL API

우동희(DongHee Woo)*, 김윤호(YoonHo Kim)**

초 록

현재 프로그램이 운용되는 시스템은 기존의 싱글코어 및 멀티코어 환경을 넘어서 매니코어, 부가 프로세스 및 이기종 환경까지 그 영역이 확장되고 있는 중이다. 하지만, 기존 연구의 경우 NVIDIA 벤더에서 나온 아키텍쳐 및 CUDA로의 병렬화가 주로 이루어졌고 AMD에서 나온 범용 GPU 아키텍쳐인 GCN 아키텍쳐에 대한 성능향상에 관한 연구는 제한적으로 이루어졌다. 이런 점을 고려해 본 논문에서는 GCN 아키텍쳐의 GPGPU 환경인 OpenCL 내에서의 성능향상 기법에 대해 연구하고 실질적인 성능향상을 보였다. 구체적으로, 행렬 곱셈과 컨볼루션을 적용한 GPGPU 프로그램을 본 논문에서 제시한 성능향상 기법을 통해 최대 30% 이상의 실행시간을 감소시켰으며, 커널 이용률 또한 40% 이상 높였다.

ABSTRACT

The current system upon which a variety of programs are in operation has continuously expanded its domain from conventional single-core and multi-core system to many-core and heterogeneous system. However, existing researches have focused mostly on parallelizing programs based CUDA framework and rarely on AMD based GCN-GPU optimization. In light of the aforementioned problems, our study focuses on the optimization techniques of the GCN architecture in a GPGPU environment and achieves a performance improvement. Specifically, by using performance techniques we propose, we have reduced more then 30% of the computation time of matrix multiplication and convolution algorithm in GPGPU. Also, we increase the kernel throughput by more then 40%.

키워드 : 병렬 컴퓨팅, 최적화, 아키텍쳐, 가속기
OpenCL, Optimization, GP-GPU, GCN Architecture, GPU

본 연구는 2016학년도 상명대학교 교내연구비를 지원받아 수행하였음.
* First Author, Graduate School of Computer Science, Sangmyung University(deltahotel93@gmail.com)
** Corresponding Author, Department of Computer Science, Sangmyung University(E-mail: yhkim@smu.ac.kr)
Received: 2017-12-20, Review completed: 2018-02-13, Accepted: 2018-02-20

1. 서 론

현재 컴퓨터 구조의 발전 방향은 기존의 싱글 코어, 멀티 코어, 매니 코어를 넘어 점점 더 이기종 환경으로 발전하고 있다[11]. 이러한 이기종 환경에서는 기존 컴퓨터의 연산을 담당해 왔던 CPU 외에 특수한 목적을 위해 설계된 DSP(Digital Signal Processor), 또는 FPGA (Field Programmable Gate Array), 특히 GPU (Graphic Processing Units)를 이용하는 프로그램의 기반이 확장되었다[10]. 기존 CPU 코어만의 연산으로는 시간이 오래 걸리거나 처리할 수 없었던 다양한 물체들의 상호작용을 다루는 입자 시뮬레이션이나 인공지능과 같은 영역을 GPU와 같은 가속기가 처리할 수 있게 됨에 따라 프로그램 개발 방향은 점점 더 GPU 환경으로 옮겨가고 있는 중이다[14]. 기존의 CPU를 기반으로 설계된 프로그램은 근본적으로 구조가 다른 GPU 디바이스 상에서도 동일한 성능으로 실행된다는 보장이 없기 때문에, GPGPU (General-Purpose Graphic Processing Unit) 환경에서의 성능향상에 관한 연구가 현재 진행중이다[12]. 하지만 선행 연구들은 특정 알고리즘의 GPU 환경 안에서의 구동 및 NVIDIA 계열의 맥스웰, 파스칼 아키텍쳐 등의 GPGPU 성능에 관한 연구를 위주로 진행되었고 AMD 계열의 GCN 아키텍쳐에 관한 성능향상 연구는 비교적 제한적인 수준에서 이루어졌다[9, 11, 13]. 이러한 문제는 GCN 계열 GPU의 GPGPU 성능 폭을 제한하였으며 GCN 아키텍쳐의 범용성에도 불구하고 GPGPU 활용에 있어 GCN 아키텍쳐 선택을 어렵게 하였다. 특히 임베디드 기기 및 가속기로써 GCN 아키텍쳐가 적용되는 다양한 도메인을 고려할 때, GCN 아키텍쳐의 최적화는 필수적이다.

따라서 본 논문은 GCN 아키텍쳐를 분석 하고 이에 따른 성능향상 기법을 제안하고자 한다. 이를 위해 GCN 아키텍쳐 상의 구조 및 성능향상 요소를 분석하고 GCN 아키텍쳐 기반의 GPU 자원을 최대한 이용할 수 있는 효율적인 성능향상 기법을 제안한다. 이후 제안하는 기법을 행렬 곱셈, 컨볼루션 두 알고리즘에 적용하고 성능향상 폭을 정량적으로 제시, 본 논문에서 제안하는 성능향상 기법에 대한 일반적 적용이 가능함을 보인다.

본 논문의 구성은 다음과 같다. 제2장에서는 본 논문의 실험 환경이 되는 GCN 아키텍쳐의 내부 구조를 분석한다. 제3장에서는 제2장에서 연구한 GCN 아키텍쳐를 기반으로 성능향상 요소를 분석, 성능향상 기법을 제안하고 이를 기반으로 제4장에서 프로그램의 성능향상을 정량적으로 제시한다. 제5장은 본 논문에 대한 결론과 향후 연구방향을 제시하였다.

2. GCN 아키텍쳐

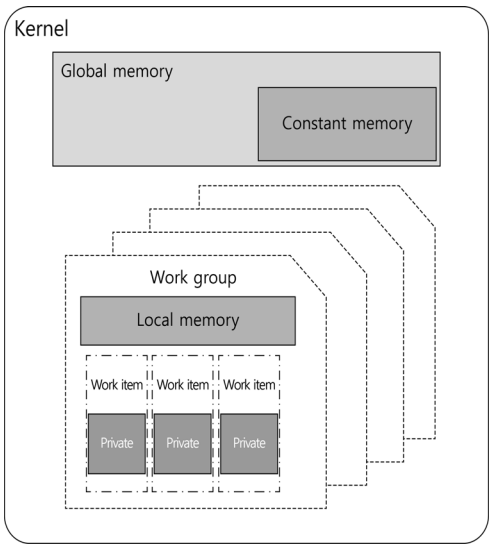
본 장에서는 GCN(Graphic Core Next) 아키텍쳐의 전반적인 특징 및 메모리 구조, 커널의 동작을 살펴보고 성능향상에 중요한 부분을 살펴본다.

2.1 GCN 아키텍쳐 구조

AMD에서 개발한 GCN 아키텍쳐는 2011년 출시한 GPU의 마이크로 아키텍쳐와 명령어 집합을 통칭한다[5]. 주요 특징은 다음과 같다.

- **Compute Units(CU)**
  CU는 GCN 아키텍쳐내 처리를 담당하는 프로세스로서, 스케쥴러, 4개의 SIMD Vector Units으로 구성되어 있다.
- **Wavefront**
  웨이브프론트는 GCN GPU 내 하나의 엑티브 하드웨어 쓰레드로 볼 수 있다. 커널 내 작업 그룹도 웨이브프론트의 개수에 따라 나눠진다.
- **SIMD Vector Unit**
  각 코어(CU) 안의 SIMD 벡터유닛은 16-lane의 정수 및 부동 소수점 ALU, 64KB의 VGPR (Vector General Purpose Register)로 구성되어 있다.

GPU는 그래픽 처리를 위한 복잡한 태스크들의 집합을 처리하기 위해 관리를 필요로 하기 때문에 지연시간을 효과적으로 숨길 수 있는 멀티쓰레딩을 특징으로 갖는다[3].



〈Figure 1〉 OpenCL Memory Model

2.2 메모리 구성

OpenCL의 메모리 구조는 추상적으로 정의되어 있다. 캐시 일관성 적용이 구조적으로 힘든 GPU의 경우 소프트웨어 관리 캐시를 제공하며 개발자에게 메모리 공간의 매핑을 맡긴다. OpenCL에서 정의하고 있는 추상 메모리 모델은 <Figure 1>과 같다.

각 메모리 영역을 설명하면 다음과 같다.

- **글로벌 메모리**
  글로벌 메모리는 GPU 안에서 일반 시스템 RAM과 같은 역할을 한다.
- **상수 메모리**
  상수 메모리는 읽기 전용 데이터로 사용된다.
- **로컬 메모리**
  로컬 메모리는 연산 디바이스 내에서(GCN 디바이스의 경우 CU)유일한 공간을 가지는 스크래치패드 메모리이다.
- **프라이빗 메모리**
  작업 그룹 내 하나의 작업 아이템이 독점적으로 소유하고 있는 메모리이다

GCN 디바이스중 하나인 HD7970의 메모리 구조는 다음과 같다. 각 작업 그룹(벡터 프로세서)은 32KB의 로컬 데이터를 가지며 256KB의 프라이빗 메모리(레지스터 파일)를 가진다. 프라이빗 메모리와 로컬 메모리는 논리적으로는 분리돼 있으나 물리적으로는 같은 공간을 차지하는 형태로 설계돼있다. 메모리의 경우 데이터가 속한 위치에 따라 최대 10배 이상 속도 차이가 나기 때문에 효율적인 메모리 사용은 프로그램의 실행시간 감소에 큰 부분을 차지한다[1].

2.3 커널 동작

OpenCL 호스트 함수는 호스트와 커널 사이의 연결에 대한 동작, 커널의 세부사항을 설정하며 커널 설정에 따라 타겟 디바이스 내부의 동작 또한 변화한다. 변화의 대상에는 작업 아이템 및 작업 그룹의 위치 등이 있으며 이러한 동작은 다른 작업 그룹과의 통신, 동기화 등에 영향을 끼친다[6].

OpenCL 내에서 커널 인스턴스(또는 작업 아이템)는 전체 실행공간을 구성하고 있다. 프로그램이 커널 영역으로 접근하면 OpenCL 문맥(Context) 안에서 정의된 작업 풀(pool)인 큐 객체(queue object)가 함수를 호출 해 큐 안의 작업들을 처리한다. 이 때, 실행 공간은 1, 2, 최대 3차원으로 정의된다[4]. 실행 공간 내에서 처리되는 작업들은 각각의 하드웨어 유닛에서 서로 독립적으로 동작한다. 독립적으로 실행되는 작업 아이템은 SIMD 레인 개수에 비례해 하나의 작업 그룹을 이루며 작업 그룹의 실행은 CU 안에서 이뤄진다. 본 연구에서 실행 환경으로 선택한 HD7970의 경우 64개의 작업 아이템이 하나의 SIMD 유닛에서 독립된 하드웨어 쓰레드로 동작한다. 웨이브프론트는 64개의 작업 아이템을 기준으로 하나를 구성하는데 이 때 웨이브프론트의 크기는 하드웨어에 의존적이며 커널 내에서 하나의 웨이브프론트를 기준으로 동작한다.

3. 성능향상 기법

본 장에서는 GCN 아키텍쳐 구조를 분석하고 그에 따른 성능향상 기법을 제안하고자 한다. 이를 위해 커널의 성능에 직접적으로 연관된 두 가지 요소, LDS 이용과 커널 동작을 분석하고 그에 따른 성능향상 기법을 제시한다.

3.1 LDS(Local Data Share) 적재

<Figure 1>의 LDS(로컬 메모리)는 글로벌 메모리에 비해 최대 10배 더 빠른 대역폭을 가지며 실제 어플리케이션에서 약 3배 이상의 속도로 CU와 통신한다[5]. 또한 LDS 내의 데이터는 재사용이 가능하다는 장점이 있다. LDS는 주로 CU에 의한 복잡한 계산을 필요로 하는 데이터를 적재하며 같은 작업그룹 내의 데이터 공유에 쓰인다. GCN 디바이스 중 하나인 HD7970의 하드웨어 상세를 살펴보면 CU당 64KB 만큼의 LDS를 가지고 있으며 타 GCN 디바이스 또한 비슷한 크기의 LDS를 가진다[1, 7].

커널이 동작하는 CU 내 LDS는 명시적으론 분리돼있지만 실제 SIMD 레인에서 다수의 CU와 같은 메모리공간을 차지하기 때문에 LDS 사용에 대한 제한은 상대적으로 적다. 이러한 점을 고려할 때 직접적으로 조작할 수 있는 LDS의 크기는 식 (1)과 같다.

$$LMS \leq N_{CU} \cdot Size_{LDS} \quad (1)$$

위 식에서 LMS(Local Memory Size)는 가용 로컬 메모리 사이즈를 나타내고 N_{CU}와 $Size_{LDS}$는 각각 CU의 개수와 디바이스 내 LDS의 크기를 나타낸다. 처리가 요구되는 전체 메모리에 대해서 식 (1)만큼의 데이터를 LDS에 할당 할 경우, GCN 아키텍쳐 내에서의 테스크는 CU 내에서 해당 크기만큼의 처리 데이터에 대한 저지연 접근을 보장받는다. 이를 통해 GPGPU 어플

리케이션은 글로벌 메모리와 직접적으로 통신하는 대신 글로벌 메모리의 데이터를 LMS 크기만큼의 LDS에 복사, 더 낮은 지연시간으로 커널과의 상호작용이 가능하다.

3.2 작업그룹 조정

커널 이용률은 CU와 PE의 이용률과 높은 상관관계를 가진다. OpenCL은 커널 동작에 대해 상세한 가이드라인을 제시하는 대신 작업 차원의 수(1, 2, 또는 3)와 지역 작업 그룹 분할로 이를 대신하였다. GCN 아키텍처에서 이러한 역할을 하는 요소는 웨이브프론트로, 커널 이용률은 웨이브프론트의 동작과 깊은 상관관계를 가지고 있다.

웨이브프론트의 이용률은 전역 작업그룹과 지역 작업그룹의 크기 분할로 결정된다. 따라서 커널의 성능향상을 위해서는 작업 그룹 및 작업 아이템의 개수를 명시적으로 지정해줘야 한다.

GCN 디바이스 내 웨이브프론트는 모두 2의 승수개의 작업 아이템으로 구성돼있다. 하나의 웨이브프론트는 SIMD 레인을 기준으로 실행되기 때문에 2의 승수만큼의 작업 아이템이 작업 그룹을 이룰 때 높은 SIMD 이용률을 보인다. 작업 그룹을 웨이브프론트 크기에 맞추면, 최적의 작업그룹의 크기는 32의 배수(32, 64, 128, 192, 256)로 나타낼 수 있으며 이는 작업 아이템과의 상호작용이 얼마나 필요한가에 따라 의존적이다. 작업 그룹의 크기가 웨이브프론트의 크기를 초과하더라도 2의 승수 개만큼의 작업 아이템이 다수의 웨이브프론트에서 동작하기 때문에 이론적으로는 전체 SIMD 이용률을 가지며 실제로도 높은 SIMD 이용률을 보인다. 이는 작업 그룹의 논리적 차원이 바뀌더라도 똑같다. 웨이브프론트 내 쓰레드가 처리할 수 있는 최재한의 작업 개수를 적제함에 따라, 실질적으로 동작하는 쓰레드의 개수를 최대화 하고 전체 처리율을 향상시킴이 가능하다.

4. 성능 분석

본 장에서는 제안한 성능향상 기법을 기존의 병렬 알고리즘 프로그램에 적용, 정량적으로 얼마만큼의 성능향상이 있는지를 확인한다. 성능 분석의 경우 CODEXL을 이용해 확인하였으며 성능 비교는 커널의 동작시간과 커널 이용률 두 개를 기준으로 진행하였다[5].

4.1 실험 환경 및 선정 알고리즘

성능 분석 환경은 <Table 1>과 같다. 본 실험에서 성능에 관한 지표는 GPU의 실행 시간만을 기준으로 하였다.

〈Table 1〉 Experiment Environment

Category	Description
CPU	Intel i5-4440
RAM	8G / Dual-channel
GPU	HD7970
OS	Windows 8
Platform	Visual Studio 2012

성능 측정을 위한 알고리즘은 행렬 곱셈 및 컨볼루션 두 개의 알고리즘을 사용하였다. 행렬 곱셈의 경우 슈퍼 컴퓨팅 연산능력 측정을 위해 많이 쓰이는 방법으로 크기에 따라 쉽게 확장 가능하며 쉽게 병렬화가 가능한 장점이 있다

[13]. 이미지 컨볼루션은 처리 이미지의 픽셀의 주변 정보를 이용해 변경을 가하는 알고리즘으로 이미지 처리, 패턴인식 등의 기계학습 알고리즘으로서도 많이 쓰이고 있다[2, 8]. 각각의 픽셀은 GPU 내에서 SIMD 형태로 처리된다.

본 성능분석 실험에서 사용한 행렬은 'float' 데이터 형의 정방행렬이며 컨볼루션의 경우 512×512 흑백(8bit) 영상에 7×7 크기의 DoG (Difference of Gaussian) 필터를 사용하였다.

4.2 행렬 곱셈

<Table 2>는 병렬화 된 행렬 프로그램에 대한 기존 코드와 성능향상 기법을 적용한 코드의 커널 이용률을 비교한 표이다. 커널의 경우 가장 큰 변화를 보인 것은 LDS 사용율과 전역 작업의 크기이다. LDS의 경우 평균적으로 2.5k로 도출되었다. 전역 작업그룹의 경우, 기존 프로그램은 하나의 작업 아이템이 하나의 연산을 맡고 그룹의 분리가 안됐기 때문에 웨이브프론트당 이용률이 낮았다. 변경된 코드의 경우 작업 그룹을 웨이브프론트 크기에 맞췄기 때문에 이용률이 높아졌음을 확인할 수 있다.

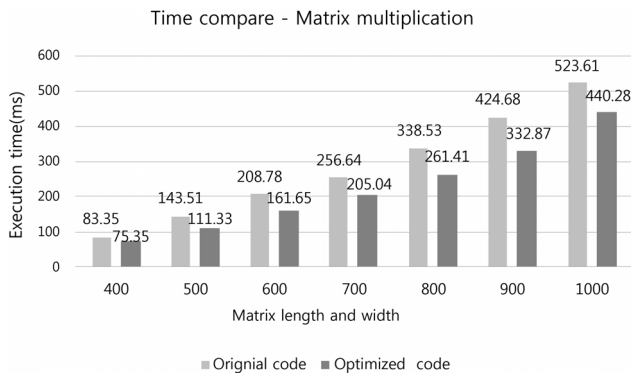
〈Table 2〉 Kernel throughput-Matrix Multiplication

	Original code	Optimized code
VGPR per work item	7	8
SGPR per work item	26	36
LDS usage	0	2.5k
Average work group per wave	4.8	8

원본 코드와 성능향상 기법을 적용한 코드에 대한 성능 비교 결과는 <Figure 2>와 같다. <Figure 2>를 통해 행렬 단위가 늘어날수록 최대 20% 이상의 실행시간 감소가 가능함을 볼 수 있다. 행렬 단위가 작을 때는 비교적 성능향상 폭이 적으나, 단위가 높아질수록 실행시간 감소폭이 증가하였다.

4.3 이미지 컨볼루션(Convolution)

<Table 3>은 커널 이용률을 나타낸 도표이다. 기존 코드는 매 연산마다 글로벌 메모리에서 데이터를 커널로 전송하는 형식으로 설계돼있다. 성능향상 기법을 적용한 코드의 경우 주기적으



〈Figure 2〉 Comparison of Execution Time-Matrix Multiplication

로 접근이 필요한 픽셀 데이터를 LDS로 복사하여 VGPR 이용률을 87% 이상 증가시켰다. 기존 프로그램의 경우 연산 작업이 웨이브프론트에 대한 고려 없이 큐에 삽입돼 웨이브프론트당 상대적으로 적은 작업그룹이 실행되었다. 수정된 코드의 경우 작업 그룹당 아이템을 데이터를 웨이브프론트의 실험환경상 제한인 64개에 근접시켜 더 높은 커널 이용률을 보였다.

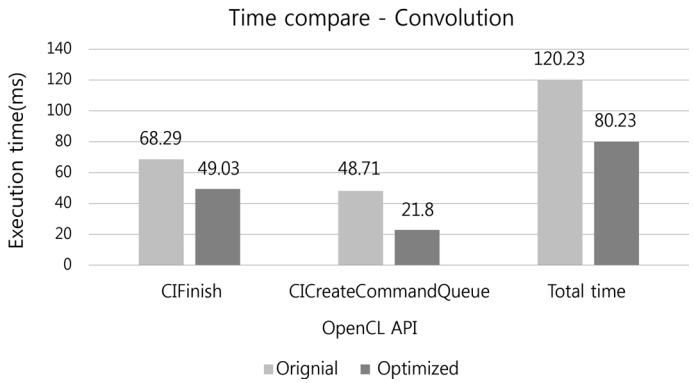
〈Table 3〉 Kernel Throughput-Convolution

	Original code	Optimized code
VGPR per work item	8	15
SGPR per work item	40	26
LDS usage	0(k)	12(k)
Average work group per wave	4.8	6.4

<Figure 3>은 각 OpenCL API 함수의 실행시간을 도식화 한 것이다. <Figure 3>에서 볼 수 있듯이 GCN 아키텍처 성능향상 기법에 기반 한 변경된 코드의 실행시간이 약 25% 감소된 것을 볼 수 있다. 특히, 작업 큐를 생성하는 API 함수의 실행시간이 반 이상 감소하였다. 변경된 코드의 경우 큐에 삽입해야 하는 작업의 개수도 이와 비례하여 감소함에 따라 더 짧아진 실행 시간을 산출하였다.

5. 결 론

본 논문은 이기종 환경 내에서 동작하는 프로그램에 대한 GPU, 그 중에서도 GCN 아키텍처 성능향상 기법에 대해 다뤘다. 이를 위해 GCN 아키텍처에 대한 특징을 파악하고 제3장에서 앞장에서 연구한 GCN 아키텍처의 특징을 토대로 한 GCN 아키텍처 기반 GPGPU 성능향상 기법을 제안하였다. 제4장에서는 제안한 최적화 기법을 토대로 두 가지 프로그램에 대한 최적화를 진행하고 향상된 성능을 정량적으로 보임과 함께 본 연구에서 제안한 GCN 아키텍처 기반 GPU의 성능향상 기법에 대한 타당성을 입증하였다. 이를 통해 범용 병렬 컴퓨팅 프레임워크인 OpenCL과 GCN 아키텍처의 관계에 대해 살펴보았고 GCN 아키텍처 기반 GPU의 GPGPU 최적화 기법을 제시하였다. 향후 연구로는 본 연구에



〈Figure 3〉 Execution Time of Main OpenCL API Function-Convolution

서 제안한 최적화 기법을 통해 향후 타 아키텍쳐 기반의 최적화 연구와 함께 호스트상에서의 OpenCL 인자 데이터 및 커널 분석과 이에 따른 자동 최적화된 인자 설정을 제안한다.

References

[1] AMD OpenCL Programming User Guide.

[2] Aritsugi, M., Fukatsu, H., and Kanamori, Y., "Parallel Image Convolution Processing with Replicas in a Network of Workstations," Institute of Electronics Information and Communication, Vol. 88, No. 6, pp. 1199-1209, 2005.

[3] Choi, H. J. and Kim, C. H., "Performance Evaluation of the GPU Architecture Executing Parallel Applications," The Korea Contents Society, Vol. 12, No. 5, 10-21, 2012.

[4] Fraire, J. A., Ferreyra, A., and Marques, C., "OpenCL Overview, Implementation, and Performance Comparison," IEEE, Vol. 11, No. 1, pp. 274-280, 2013.

[5] http://www.amd.com/ko-kr.

[6] http://www.khronos.org/opencl/.

[7] Huang, D., Wen, M., Xun, C., Chen, D., Cai, X., Qiao, Y., Wu, N., and Zhang, C., "Automated Transformation of GPU-Specific OpenCL Kernels Targeting Performance Portability on Muiti-Core/Many-Core CPUs," Lecture Notes in Computer Science, No. 8632, pp. 210-221, 2014.

[8] Jung, H. I., Park, I. S., and Ahn, H. C., "Identifying the Key Success Factors of Massively Multiplayer Online Role Playing Game Design using Artificial Neural Networks," The Journal of Society for e-Business Studies, Vol. 17, No. 1, pp. 23-38, 2012.

[9] Lee, D., Dinov, I., Dong, B., Gutman, B., Yanovsky, I., and Toga, A. W., "CUDA optimization strategies for compute- and memory-bound neuroimaging algorithms," Computer Methods and Programs in Biomedicine, Vol. 106, No. 3, pp. 175-187, 2012.

[10] Lee, S. G., "Enhancing Performance of Embedded System using FPGA Processor," Namseoul University Press, Vol. 7, No. 1, pp. 56-67, 2010.

[11] Lee, Y. H. and Kim, Y. J., "Parallel Intersection Detection Algorithm using CUDA," HCI, Vol. 2008, No. 2, pp. 451-455, 2008.

[12] Moon, H. J., Jeon, J. N., and Kim, S., "A Performance Analysis for Benchmarks on Heterogeneous Environment," KISS, Vol. 23, No. 2B, pp. 1635-1638, 1996.

[13] Oyarzun, G., Borrell, R., Gorobets, A., and Oliva, A., "MPI-CUDA sparse matrix-vector multiplication for the conjugate gradient method with an approximate inverse preconditioner," Computers & Fluids, Vol. 92, pp. 244-252, 2014.

[14] Venetillo, J. S. and Celes, W., "GPU-based particle simulation with inter-collisions," The Visual Computer, Vol. 23, No. 9-11, pp. 851-860, 2007.

저 자 소 개

우동희 (E-mail: deltahotel93@gmail.com)
2015년 상명대학교 컴퓨터과학과 졸업 (학사)
현재 상명대학교 컴퓨터과학과 졸업 (석사)
관심분야 병렬처리, 기계학습, 인공지능, 데이터 분석

김윤호 (E-mail: yhkim@smu.ac.kr)
1985년 서울대학교 계산통계학과 (학사)
1987년 서울대학교 계산통계학과 계산학전공 (석사)
1996년 서울대학교 전산과학전공 (박사)
현재 상명대학교 미래융합공과대학 컴퓨터과학과 교수
관심분야 분산시스템, 사물인터넷, 시맨틱 웹, 저작권보호기술, 디지털 콘텐츠