KIPS Trans. Comp. and Comm. Sys.
Vol.9, No.9 pp.189~196
ISSN: 2287-5891 (Print), ISSN 2734-049X (Online)

https://doi.org/10.3745/KTCCS.2020.9.9.189

GPGPU Task Management Technique to Mitigate Performance Degradation of Virtual Machines due to GPU Operation in Cloud Environments

Jihun Kang† · Joon-Min Gil††

ABSTRACT

Recently, GPU cloud computing technology applying GPU(Graphics Processing Unit) devices to virtual machines is widely used in the cloud environment. In a cloud environment, GPU devices assigned to virtual machines can perform operations faster than CPUs through massively parallel processing, which can provide many benefits when operating high-performance computing services in a variety of fields in a cloud environment. In a cloud environment, a GPU device can help improve the performance of a virtual machine, but the virtual machine scheduler, which is based on the CPU usage time of a virtual machine, does not take into account GPU device usage time, affecting the performance of other virtual machines. In this paper, we test and analyze the performance degradation of other virtual machines due to the virtual machine that performs GPGPU(General-Purpose computing on Graphics Processing Units) task in the direct path based GPU virtualization environment, which is often used when assigning GPUs to virtual machines in cloud environments. Then to solve this problem, we propose a GPGPU task management method for a virtual machine.

Keywords : GPU Virtualization, Performance Isolation, Scheduling, Cloud, GPGPU Task Management

클라우드 환경에서 GPU 연산으로 인한 가상머신의 성능 저하를 완화하는 GPGPU 작업 관리 기법

강 지 훈† · 길 준 민††

요 약

최근 클라우드 환경에서는 고성능 연산이 가능한 GPU(Graphics Processing Unit) 장치를 가상머신에게 적용한 GPU 클라우드 컴퓨팅 기술이 많이 사용되고 있다. 클라우드 환경에서 가상머신에게 할당된 GPU 장치는 대규모 병렬 처리를 통해 CPU보다 더 빠르게 연산을 수행할 수 있으며, 이로 인해 다양한 분야의 고성능 컴퓨팅 서비스들을 클라우드 환경에서 운용할 때 많은 이점을 얻을 수 있다. 클라우드 환경에서 GPU 장치는 가상머신의 성능 향상에 많은 도움을 주지만 가상머신의 CPU 사용 시간을 기반으로 작동하는 가상머신 스케줄러에서는 GPU 장치의 사용 시간이 고려되지 않아 다른 가상머신들의 성능에 영향을 미친다. 본 논문에서는 클라우드 환경에서 가상머신에게 GPU를 할당할 때 많이 사용되는 직접 통로기반 GPU 가상화 환경에서 GPGPU(General-Purpose computing on Graphics Processing Units) 작업을 수행하는 가상머신으로 인한 다른 가상머신들의 성능 저하 현상을 검증하고 분석하며, 이를 해결하기 위한 가상머신의 GPGPU 작업 관리 기법을 제안한다.

키워드 : GPU 가상화, 성능 격리, 스케줄링, 클라우드, GPGPU 작업 관리

1. 서 론

최근 IT 인프라가 클라우드 환경으로 변화하면서 대규모 작업을 빠르게 처리할 수 있는 고성능 컴퓨팅(High Performance Computing)에 대한 요구가 증가하였다. 가상머신에게 GPU를 할당하여 가상머신이 대규모 작업을 빠르게 처리할 수 있도록 지원하는 방법은 클라우드 환경에서 고성능 처리를 지원하기 위해 사용되는 방법의 하나이다. 고성능 컴퓨팅을 위해 사용되는 GPGPU[1] 기술은 전통적으로 CPU에서 처리하던 연산 작업을 수천 개의 연산 코어를 가지고 있는 GPU를 사용해 처리함으로써 연산 속도를 가속화한다. 연산 처리 시 GPU를 사용하면 수천 개의 연산 코어를 통한 대규모 병렬 처리가 가능하므로 CPU와 비교했을 때 동일한 규모의 연산을 상대적으로 더 짧은 시간에 처리할 수 있으며, 이러한 장점으로 인해 클라우드 환경에서도 가상머신에게 고성능 연산

※ 이 논문은 정부(과학기술정보통신부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. NRF-2019R1F1A1062039).
† 준 회 원 : 고려대학교 정보창의교육연구소 연구교수
†† 종신회원 : 대구가톨릭대학교 컴퓨터소프트웨어학부 교수
Manuscript Received : July 6, 2020
Accepted : July 23, 2020
* Corresponding Author : Joon-Min Gil(jmgil@cu.ac.kr)

을 지원하는 방법으로 많이 사용되고 있으며, 상용 클라우드 제공 업체들도 가상머신에게 GPU 장치를 할당한 GPU 클라우드 서비스를 제공하고 있다[2, 3].

클라우드 환경에서는 일반적으로 여러 개의 가상머신이 물리 서버의 자원을 공유하며 가상머신 사이의 자원 경쟁으로 인한 성능 불균형을 방지하기 위해 가상머신의 자원 사용 순서를 결정하기 위한 별도의 가상머신 스케줄러를 사용한다. 일반적인 가상머신 스케줄러는 가상머신의 CPU 사용 시간을 기반으로 CPU를 사용하는 순서를 결정한다. 가상머신 스케줄러는 각각의 가상머신에게 자신이 사용할 수 있는 CPU 사용 시간을 할당하고 할당받은 CPU 사용 시간의 소비량을 기반으로 작업 순서를 결정한다. 기본적으로 스케줄러는 CPU를 많이 사용한 가상머신의 우선순위를 낮추고 CPU를 적게 사용한 가상머신의 우선순위를 높혀 모든 가상머신들이 균등하게 CPU를 사용할 수 있도록 가상머신의 CPU 사용 순서를 조절한다.

일반적인 가상머신 스케줄러에서는 CPU 사용 시간만을 고려한다. GPGPU 작업의 특성상 전체 실행 시간 중 상대적으로 CPU를 적게 사용하기 때문에 GPGPU 작업을 수행하는 가상머신은 CPU를 적게 사용한 가상머신으로 분류된다. 이로 인해 GPGPU 작업을 수행하는 가상머신은 CPU가 필요한 시점에 다른 가상머신보다 먼저 CPU를 사용할 수 있게 되며, 결과적으로 GPGPU 작업을 수행하는 가상머신으로 인해 CPU 사용 순서가 뒤로 밀린 다른 가상머신들의 성능이 저하되는 문제가 발생한다. GPGPU 작업을 수행하는 가상머신이 다른 가상머신들의 성능에 영향을 끼치는 것은 가상머신 사이의 성능 격리를 지원해야 하는 클라우드 환경에서 큰 문제로 작용한다. 다수의 사용자가 가상머신을 통해 자원이 공유되는 클라우드 환경에서 가상머신들이 어떤 작업을 수행해도 서로 영향이 없도록 지원하는 성능 격리는 클라우드 환경에서 반드시 보장해야 하는 중요한 요소 중 하나이다.

본 논문에서는 기존 가상머신 스케줄러에서 가상머신의 GPU 사용을 고려하지 않아 GPGPU 작업을 수행하는 가상머신으로 인해 발생하는 다른 가상머신들의 성능 저하를 완화하기 위해 GPGPU 작업의 스레드 분할 처리를 통해 가상머신의 성능 저하를 완화하는 GPGPU 작업 관리 기법을 제안한다. 그리고 실험을 통해 GPGPU 작업을 수행하는 가상머신으로 인한 다른 가상머신들의 성능 저하를 확인하고 본 논문에서 제안하는 기법의 효율성을 검증한다. 본 논문의 주요 기여는 다음과 같다.

- 본 논문에서는 클라우드 환경에서 GPGPU 작업을 수행하는 가상머신으로 인해 함께 실행되는 다른 가상머신의 성능이 저하되는 문제를 증명하고 분석함.
- 본 논문의 접근 방식은 GPGPU 작업을 수행하는 스레드들을 여러 개의 그룹으로 분할 처리하여 GPGPU 작업의 종료 및 시작 횟수를 증가시켜 다른 가상머신들의 CPU 사용 기회를 증가시킴.
- 본 논문에서 제안한 기법은 OS, 하이퍼바이저, GPU 드라이버와 같은 시스템 구성 요소를 수정할 필요가 없기 때문에 시스템 복잡도를 최소화할 수 있음.
- 본 논문에서 제안한 기법은 GPGPU API인 OpenCL의 API를 수정하여 GPGPU 작업의 소스코드의 수정이 최소화되도록 지원함.
- 본 논문에서 제안한 기법은 기존 환경과 비교하여 최대 20%의 성능이 향상되며, 실험을 통해 검증함.

본 논문은 2장에서 GPU 가상화 환경과 관련된 관련 연구를 설명하고 3장에서는 실험을 통해 GPGPU 작업을 수행하는 가상머신으로 인한 성능 저하 문제를 분석한다. 4장에서는 GPGPU 작업을 수행하는 가상머신으로 인한 다른 가상머신들의 성능 저하를 완화하는 GPGPU 작업 관리 기법에 대해 설명하고 5장에서는 실험을 통해 제안된 기법을 기존 가상화 환경과의 성능 비교 실험을 통해 효율성을 검증한다. 마지막 6장에서 본 논문의 결론과 향후 연구에 대해 설명한다.

2. 클라우드 환경에서의 GPU 장치

클라우드 환경에서는 효율적인 컴퓨팅 자원 공유를 위해 가상머신 스케줄러를 사용해 가상머신의 작업 순서를 결정한다. 가상머신 스케줄러는 다수의 가상머신들이 서로의 성능에 영향을 최소화하기 위해 모든 가상머신들이 자원을 균등한 시간동안 사용할 수 있도록 지원하며, 이를 위해 가상머신의 CPU 사용 시간을 기반으로 모든 가상머신들이 균등한 시간동안 CPU를 사용할 수 있도록 가상머신의 자원 사용 순서를 관리한다. 일반적인 컴퓨팅 환경에서는 모든 작업을 CPU에서 처리한다. 부가적인 입출력 작업도 존재하지만, 명령어 처리, 수학 연산과 같은 작업 대부분은 CPU를 사용하기 때문에 CPU 사용 시간을 기반으로 작동하는 기존 가상머신 스케줄러가 GPU를 사용하지 않는 가상머신들 사이에선 효율적으로 공평성을 제공해준다. 하지만 가상머신 스케줄러에서 GPU 사용 시간을 고려하지 않기 때문에 GPGPU 작업을 수행하는 가상머신은 GPU 사용 시간에 제약 없이 GPGPU 작업을 수행할 수 있다.

Fig. 1은 직접 통로(Direct pass-through)기반 GPU 가상화 환경의 구조를 보여준다. 가상머신에게 GPU 장치를 할당하기 위해 사용되는 대표적인 기술은 직접 통로기반 GPU 가상화 기술이다. 직접 통로기반 GPU 가상화 기술은 하이퍼바이저의 관여 없이 장치를 사용하기 때문에 가상화 계층으로 인한 성능 저하가 최소화되고 가상머신이 GPU 장치의 전체 기능을 사용할 수 있도록 지원한다.

클라우드 환경에서 GPU를 사용하기 위한 다양한 기법들이 연구되고 있다. 기존 클라우드 환경에서 GPU 장치 적용 연구는 단일 호스트머신에서 작동중인 모든 가상머신이 GPGPU 작업을 수행한다고 가정하며, 가상머신들의 GPU 공유에 초점을 맞춘다.

vCUDA[10]는 다수의 가상머신이 단일 GPU 상태에서 CUDA 애플리케이션을 구동할 수 있도록 GPU 장치의 공유 기법을 제안한다. 해당 연구는 RPC 통신을 통해 GPGPU 프로그래밍 API인 CUDA를 사용한 작업을 수행할 때 실행 되

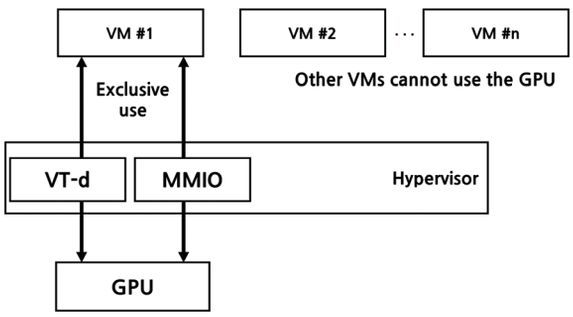

Fig. 1. Direct Pass-through Based GPU Virtualization

는 API와 매개변수를 GPU에 전달하고 연산처리 후 결과를 해당 가상머신이 반환아 다수의 가상머신들이 GPU를 공유하는 서버-클라이언트 구조이다.

GPUvm[11]은 하이퍼바이저 수준에서 GPU 장치의 가상화를 제안한 연구이다. 해당 연구는 GPU에 네이티브 GPU 디바이스 모델을 제공하고 가상머신이 GPU에 접근할 때 하이퍼바이저로 전달하여 GPU 접근을 중재하여 다수의 가상머신들이 GPU 장치의 공유 기능을 제공한다. 또한 Mediated Pass-Through[12]는 CPU에 내장되 있는 GPU 장치를 대상으로 다수의 가상머신에게 할당할 수 있는 기법을 제안한 연구이다. 해당 기술은 가상머신이 GPU에 접근하고 작업을 위한 버퍼들을 관리하고 가상머신들이 GPU 드라이버를 통해 GPU에 직접 접근할 수 있도록 지원한다.

I/O Paravirtualization[13]에 대한 연구는 Xen의 반가상화 가상머신에게 PCI 장치를 가상화해서 반가상화 가상머신이 호스트머신의 디바이스 파일을 통해 PCI 장치에 접근하기 위한 메모리 읽기, 쓰기 작업을 하이퍼바이저에서 중재하여 반가상화 가상머신에게 PCI 장치를 할당하기 위한 기법을 제안한다. GCloud[14]는 네트워크를 통해 다수의 사용자에게 게이밍 서비스를 제공하기 위한 클라우드 게이밍 기술을 제안한다. 해당 연구는 일반적으로 하이퍼바이저를 사용하는 가상화 기술을 사용하지 않고 유저레벨 가상화를 통해 다수의 사용자가 서버에 접속하여 GPU를 비롯한 다른 물리 자원을 공유하여 서비스를 사용할 수 있도록 지원한다.

클라우드 환경에서 GPU 장치를 적용하기 위한 연구는 앞서 설명한 것과 같이 다수의 가상머신이 GPU 장치를 공유하는 것과 GPGPU 작업의 성능에 초점을 맞추고 있으며. 다양한 유형의 작업을 수행하는 가상머신들이 동시에 실행되는 환경을 고려하지 않고 있다. 하지만 일반적인 클라우드 환경에서는 다양한 유형의 작업을 수행하는 가상머신들이 동시에 실행되기 때문에 GPGPU 작업을 수행하는 가상머신의 성능뿐만 아니라 다른 가상머신에게 미치는 영향도 고려해야 할 필요가 있다.

3. 가상머신의 GPGPU 작업이 다른 가상머신들에게 미치는 성능 영향

앞서 설명한 것과 같이 가상머신의 CPU 사용 시간을 기반으로 작동하는 기존 가상머신 스케줄러에서는 GPU 사용 시간을 고려하지 않는다. 본 장에서는 실험을 통해 GPGPU 작업을 수행하는 가상머신으로 인해 발생하는 가상머신들의 성능 저하를 확인하고 이를 분석한다.

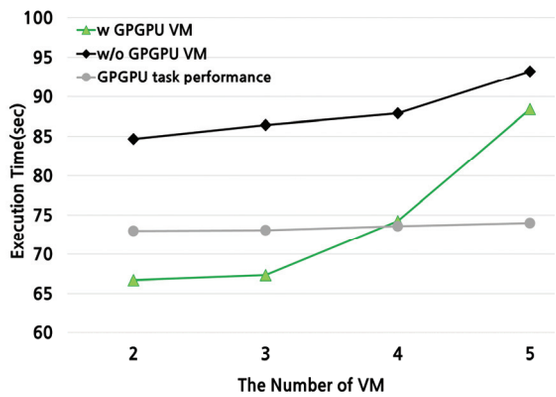

Fig. 2. Performance Degradation of VM Due to GPGPU Task

Fig. 2는 GPGPU 작업으로 인해 발생하는 가상머신의 성능 저하를 측정한 실험 결과이며, GPGPU 작업을 수행하는 가상머신외의 다른 가상머신들은 CPU 연산을 수행한다. GPGPU 작업을 수행하는 가상머신이 있는 경(Fig. 2에서 w GPGPU VM으로 표시)에는 GPGPU 작업을 수행하는 가상머신 1개를 기본적으로 실행시키고 CPU 연산을 수행하는 가상머신은 1-4개까지 개수를 증가시키며 실험을 수행한다. 그리고 GPGPU 작업을 수행하는 가상머신이 없는 경우는 CPU 연산을 수행하는 가상머신을 2-5개까지 증가하며 성능 측정을 수행한다.

실험 결과에서 볼 수 있듯이 GPGPU 작업을 수행하는 가상머신을 CPU 연산을 수행하는 가상머신과 동시에 실행하는 경우 CPU 연산을 수행하는 가상머신의 성능이 크게 저하되는 것을 알 수 있다. GPGPU 작업은 전체 작업 중 대부분을 GPU에서 처리하기 때문에 전체 실행 시간에 비해 CPU의 사용 시간이 적기 때문에 할당받은 CPU 사용 시간의 소모량이 적다. 또한, 직접 통로기반 GPU 가상화의 경우 단일 가상머신이 GPU를 독점하기 때문에 CPU와 같이 자원 경쟁이 발생하지 않는다. 이러한 특징들로 인해 GPGPU 작업을 수행하는 가상머신은 CPU를 사용할 때 다른 가상머신들보다 먼저 사용할 수 있으며 GPGPU 작업을 수행할 때 GPU 장치를 독점적으로 사용하기 때문에 다른 가상머신과 자원 경쟁을 할 필요가 없어서 동시에 운용되는 가상머신의 개수가 증가해도 GPGPU 작업의 성능은 크게 변화하지 않는다.

가상머신의 개수가 증가함에 따라 자원을 공유하는 대상이 증가하여 가상머신들의 성능이 저하되는 것은 당연한 현상이지만, GPGPU 작업을 수행하는 가상머신이 존재하는 상황에서 CPU 연산을 수행하는 가상머신의 성능은 매우 큰 폭으로 저하되는 것을 확인할 수 있다. 본 논문에서는 앞서 수행한 실험에서 보여준 가상머신의 성능 저하 문제를 완화하기 위한 GPGPU 작업의 관리 기법을 다음 장에서 자세히 설명한다.

4. 가상머신의 성능 영향을 완화하기 위한 GPGPU 작업 관리 기법

본 논문에서 제안하는 GPGPU 작업 관리 기법은 기본적으로 GPGPU 작업을 분할 처리하여 GPGPU 작업의 초기화와 종료 횟수를 증가시킨다. 이로 인해 GPGPU 연산을 수행하는 가상머신의 CPU 사용 시간을 증가시키고 다른 가상머신들의 CPU 사용 기회를 증가시킨다. GPGPU 작업은 기본적으로 수천 개에서 수만 개의 스레드로 구성된 대규모 병렬처리 구조를 갖는다. GPGPU 작업 실행 시 모든 작업은 스레드 단위로 실행되며, 스레드가 실행되는 중에는 CPU와 같은 문맥 교환이 발생하지 않기 때문에 작업이 끝날 때까지 임의로 중지할 수 없다. 즉, 한번 시작한 작업은 재스케줄링이나 다른 작업으로의 문맥 교환이 불가능하다.

또한, 하이퍼바이저 수준에서 GPGPU 작업을 관리할 수 없기 때문에 본 논문에서 제안하는 GPGPU 작업 관리 기법은 GPGPU 라이브러리인 OpenCL의 스레드 생성 함수를 수정하여 GPGPU 작업을 위해 생성하는 전체 스레드를 여러 개의 스레드 그룹으로 분할하고 스레드 그룹을 순차적으로 처리하는 방법을 사용한다. 이는 분할된 하나의 스레드 그룹이 작업이 완료되고 다음 스레드 그룹의 실행 전까지 GPGPU 작업을 중지시키는 효과를 얻을 수 있다. 그리고 다음 스레드 그룹의 실행을 위한 초기화 작업은 CPU에서 수행되기 때문에 CPU를 더 사용하게 만들고 초기화 횟수가 많아질수록 가상머신 스케줄링 단계에서 다른 가상머신들이 자원에 접근할 수 있는 기회를 증가시켜 성능 저하 문제를 완화할 수 있다.

Fig. 3은 본 논문에서 제안하는 GPGPU 작업 관리 시스템의 구조이다. GPU 작업을 관리하기 위해 GPGPU 프로그래밍 코드에서 스레드 생성 함수를 수정한다. 하이퍼바이저의 가상머신 모니터링 정보를 통해 현재 실행 중인 가상머신의 개수를 확인하여 전체 스레드에서 한 번에 처리할 스레드의 규모를 결정한다. 실행 중인 가상머신이 많을수록 한 번에 처리되는 스레드는 더 작은 규모로 분할된다. Fig. 3에서 보여주는 전체 구조도와 같이 호스트머신의 하이퍼바이저에서 가상머신의 개수에 대한 정보를 모니터링하고 그 정보를 GPGPU 작업을 수행하는 가상머신이 GPGPU 작업을 수행할 때 스레드를 생성하는 부분에서 가상머신의 개수에 따라 스레드 분할 규모를 결정하고 분할된 스레드 그룹은 차례대로 수행된다. 이로 인해 작업 초기화와 스레드 동기화 횟수가 증가하기 때문에 GPGPU 작업도 어느 정도 성능 저하가 발생한다. 본 논문에서는 이러한 특성을 고려하여 GPGPU작업을 다양한 규모로 분할하여 실험을 수행하였으며, 제세한 내용은 다음 장에서 설명한다. 또한, 실험 결과를 기반으로 GPGPU 작업의 성능에 최소한의 영향을 미치는 수준에서 GPGPU 작업의 분할 규모를 결정한다.

GPGPU 작업을 수행하는 가상머신으로 인한 다른 가상머신들의 성능 저하를 최소화하기 위해 Equation (2)를 만족하는 범위 내에서 Equation (1)을 통해 GPGPU 작업의 스레드를 분할한다.

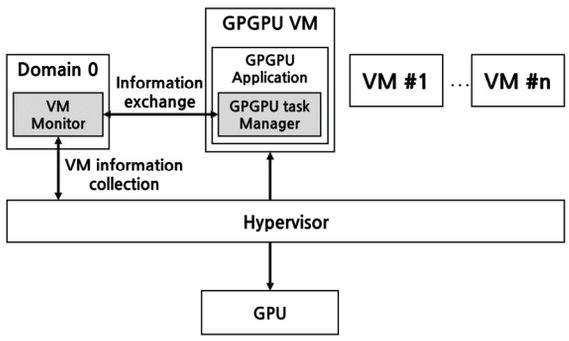

Fig. 3. Overall Structure

$$Number\ of\ ThreadGroup = \frac{Total\ Thread\ of\ task / Number\ of\ GPUcore}{Number\ of\ VM \times \alpha} \quad (1)$$

Equation (1)은 GPGPU 작업 성능에 영향을 최소화하는 범위 내에서 GPGPU 작업을 수행하는 가상머신으로 인한 다른 가상머신들의 성능 저하를 최소화할 수 있는 스레드 분할 크기를 결정한다. Equation (1)에서 *Total Tread of task*는 GPGPU 작업을 구성하는 스레드의 개수를 의미하며, *Number of VM*은 현재 실행 중인 가상머신의 개수를 의미한다. 그리고 *Thread Group*은 GPGPU 작업의 분할 개수이며, α는 선행 실험을 통해 Equation (2)를 만족하도록 도출한 매개변수이다.

$$\begin{aligned} &Minimize \sum_{i=1}^{n} Runtime_{VM_i} \\ &s.t. \quad Number\ of\ ThreadGroup \in \mathbb{Z} \\ &\quad 0 < \alpha \le \frac{Total\ \ Thread\ \ of\ \ task}{Number\ \ of\ \ GPU\ \ core} \end{aligned} \quad (2)$$

본 논문에서 제안하는 GPGPU 작업 관리자는 GPGPU 작업의 소스 코드 내부에 GPU 작업을 호출하는 커널 함수가 호출되면 수행된다. 본 논문에서는 OpenCL의 커널 함수를 호출하는 API인 ***clEnqueueNDRangeKernel()***함수 대신 커널 함수를 호출하기 위한 ***clEnqueueNDRange Kernel_N()*** 함수를 구현했다. 제안한 함수는 기존 OpenCL의 커널 함수 호출 API 대신 사용되며 커널 함수가 실행되기 전에 커널 함수의 매개변수와 실행 중인 가상머신에 대한 정보를 기반으로 본 논문에서 제안한 Algorithm 1이 동작하도록 GPGPU API를 수정하였다.

Algorithm 1에서 보여주는 것처럼 가상머신 개수를 모니터링하는 모듈은 하이퍼바이저 내부에서 상시 작동한다. 그리고 GPGPU 작업이 시작되면 가상머신 개수 정보를 받아온 후 동시 수행 중인 가상머신의 개수에 따라 GPGPU 작업 분할 규모를 정하게 된다. 본 논문에서는 실험을 통해 가상머신의 개수에 따라 GPGPU 작업 분할 규모를 결정하였다. 그리고 분할된 스레드 그룹이 모두 실행될 때까지 GPGPU 작업을 차례대로 수행한다. 또한, GPGPU task에서 생성되는 스레드의 개수가 GPU 코어의 개수보다 작은 경우, GPGPU 작업은 한 번에 처

리될 수 있기 때문에 GPGPU 작업을 분할처리 하지 않으며, 현재 호스트머신에서 GPGPU 작업을 수행하는 가상머신 혼자 실행되는 경우에도 GPGPU 작업을 분할 처리 하지 않는다.

Algorithm 1

```
1  if GPGPU Task's Kernel function Called
2    if Number of VM != 1
3      if totThread 〈 number of GPU core
4        totThread = number of Threads
5        activeVM = total number of VMs
6        remainthread = totThread
7        threadgroup = totThread / ( activeVM * α )
8        divthread = totThread / threadgroup
9        if totThread != divthread * threadgroup
10         remaindThread = totThread % threadgroup
11       end if
12       while 0 != remainJob
13         Processing Job( divThread )
14         remainJob =- divThread
15         if remainJob == 0
16           if remainderThread != 0
17             Processing thread( remaindThread )
18           end if
19           return processing result;
20         end if
21       end while
22     end if
23   end if
24 end if
```

본 논문에서 제안한 GPGPU 작업 관리자는 동시 수행되는 가상머신의 개수에 따라 미리 설정된 크기만큼 GPGPU 작업을 동적으로 분할 하도록 지원한다. 또한, 스레드를 동일한 스레드 개수로 스레드 그룹을 분할하고 남은 나머지 스레드가 존재한다면 가장 마지막에 처리한다.

GPU 작업의 특징상 한번 실행을 시작한 스레드는 중간에 정지 및 문맥 교환이 발생하지 않기 때문에 GPGPU 작업을 수행하기 전에 스레드 분할을 수행한다. 새로운 GPGPU 작업이 실행될 때 모니터링 정보를 통해 가상머신의 개수를 확인하여 위 알고리즘을 통해 스레드 분할을 수행한다. GPGPU 작업의 전체 스레드를 분할하면서 스레드의 생성, 실행 및 종료 횟수가 많아짐에 따라 GPGPU 작업 성능이 저하되기 때문에 가상머신의 개수에 따른 GPGPU 작업의 분할 규모는 클라우드 서비스 제공자 측면에서 CPU 연산을 수행하는 가상머신과 GPGPU 작업을 수행하는 가상머신 사이의 성능에 대한 효율성을 고려하여 GPGPU 작업의 분할 규모를 설정해야 한다. 동시에 실행되는 가상머신의 개수에 따른 분할 규모를 설정하면 GPGPU 작업 관리자를 통해 새로운 GPGPU 작업을 수행할 때 마다 호스트머신에서 실행 중인 가상머신의 개수에 따라 자동적으로 GPGPU 작업의 분할 규모를 설정한다.

따라서 본 논문에서 제안하는 GPGPU 작업 관리 기법은 새로운 GPGPU 작업이 시작될 때 호스트머신에서 실행 중인 가상머신의 개수를 확인하여 현재 상태에 따라 스레드 분할 규모를 결정하고 GPGPU 작업을 수행하는 가상머신만 존재하는 경우 스레드 분할을 수행하지 않고 전체 스레드를 한 번에 처리하도록 지원하며, 호스트머신에서 실행 중인 가상머신의 개수에 따라 분할 규모를 결정할 수 있도록 구성하였다.

본 논문에서는 제안하는 기술은 Xen 기반 가상화 환경에서 OpenCL API를 사용한 GPGPU 애플리케이션을 실행시키는 가상머신이 실행되는 환경을 대상으로 한다. 하지만 유저 레벨에서 작업 분할을 수행하기 때문에 가상화 플랫폼에 종속적이지 않아 KVM과 같은 다른 가상화 플랫폼에 적용할 수 있다. 또한, GPGPU 프로그래밍 모델의 기본적인 구조는 API마다 크게 다르지 않기 때문에 다른 GPGPU API에 손쉽게 적용할 수 있다.

5. 실험 및 성능 평가

이번 장은 본 논문에서 GPGPU 작업을 수행하는 가상머신으로 인한 성능 저하 문제를 해결하기 위해 제안한 GPGPU 작업 관리 기법 효율성을 평가하기 위한 실험을 수행한다. GPGPU 작업 관리 기법에 대한 효율성을 검증하기 위해 3가지 실험을 수행한다.

첫 번째는 GPGPU 작업의 분할 규모에 따라 변하는 CPU 연산을 수행하는 가상머신의 성능이다. GPGPU 연산으로 인해 발생하는 다른 가상머신들의 성능 저하 문제를 완화하는 것은 본 논문에서 제안한 기법의 가장 큰 목적이다. 두 번째는 GPGPU 작업의 분할 규모에 따른 GPGPU 작업의 성능 변화이다. GPGPU 작업의 분할을 통해 스레드 그룹이 많아질수록 GPGPU 작업의 스레드 생성, 동기화 및 작업 종료 횟수가 증가하기 때문에 GPGPU 작업을 수행하는 가상머신의 성능이 저하된다. 하지만 CPU 연산의 성능 향상을 위해 GPGPU 작업 성능 저하가 심해지면 또 다른 성능 영향 문제가 발생하기 때문에 GPGPU 작업의 성능은 어느 정도 유지하면서 CPU 연산을 수행하는 가상머신의 성능 저하를 완화할 필요가 있다. 세 번째는 최적의 GPGPU 작업 분할 규모를 적용했을 때의 성능 측정이다. 앞서 설명한 것과 같이 GPGPU 작업을 수행하는 가상머신과 CPU 연산을 수행하는 가상머신의 성능 둘 다 고려대상이기 때문에 최적의 GPGPU 작업 분할 규모를 찾아야 한다.

본 논문에서는 두 가지 실험을 수행한다. 첫 번째 실험은 GPGPU 작업의 스레드들을 2, 4, 10 그리고 20개의 그룹으로 분할 처리할 때 가상머신들과 GPGPU 작업의 성능을 측정함으로써 GPGPU 작업 분할 처리 성능과 CPU 연산을 수행하는 가상머신 성능 사이의 관계를 분석한다. 두 번째 실험에서는 GPGPU 작업 분할 처리 성능과 CPU 연산을 수행하는 가상머신 성능 측정 결과를 기반으로 동시에 실행되는 가상머신의 개수에 따른 최적의 스레드 분할 규모를 도출하고 도출된 분할 규모로 GPGPU 작업을 분할 처리했을 때 가상머신들의 성능을 측정한다. 실험을 위해 오픈소스 기반 가상화

플랫폼인 Xen 4.4.1으로 구축된 가상화 환경에서 가상머신을 생성하여 실험하였으며 자세한 실험 환경은 Table 1과 같다.

Table 1. Experiment Environment

	Host Machine	VM with GPU	VM
CPU (vCPU)	Intel Xeon E3-1231V3	4vCPU	4vCPU
Memory	32GB	6GB	6GB
GPU	-	Radeon HD 6570 (2GB GPU Memory)	-
OS	Ubuntu 14.04	Windows 7	Windows 7
Hypervisor	Xen 4.4.1	-	-
Calculation Type	-	GPU Matrix Multiplication	CPU Matrix Multiplication

실험에서 GPGPU 작업은 직접 통로 기반 GPU 가상화 기술을 통해 GPU를 할당받은 1개의 가상머신에서 수행하고 동시 수행되는 가상머신의 개수에 따라 성능을 측정하기 위해 CPU 연산 가상머신은 1-4개로 개수를 증가시키며 실험을 수행한다.

실험은 본 논문에서 제안한 GPGPU 작업 관리 기법의 효율성을 검증하기 위해 기존 가상화 환경과 GPGPU 작업 관리 기법이 적용된 환경의 성능을 측정하여 비교한다. GPGPU 작업을 수행하는 가상머신 때문에 발생하는 다른 가상머신들의 성능 저하 문제로 인해 CPU 연산을 수행하는 가상머신의 성능이 저하되는 환경에서 본 논문에서 제안한 GPGPU 작업 관리 기법의 효율성을 평가하기 위해 GPGPU 작업을 수행하는 가상머신은 5120×5120 행렬 곱셈을 4회 반복하고 CPU 연산을 수행하는 가상머신은 1600×1600 행렬 곱셈을 수행한다.

실험을 위해 사용된 CPU 행렬 곱셈 작업은 성능 차이를 명확하게 확인할 수 있도록 1분 이상 수행되도록 작업 규모를 결정하였으며, GPU 행렬 곱셈의 경우 GPU 메모리에 행렬 곱셈을 위한 데이터를 복사하고 GPU 연산 수행 전에 사용할 GPU 메모리 공간을 미리 할당해야하기 때문에 실험에서 사용한 GPU의 메모리 2GB가 초과하지 않으면서 1분 이상 작업을 수행할 수 있도록 작업 규모와 반복 횟수를 설정하였다.

그리고 GPU 및 CPU 행렬곱셈 작업은 각각 단일 가상머신에서 수행했을 때 평균적으로 65초, 73초 동안 수행되며 동시에 실행했을 때 최소 65초 동안은 서로에게 성능 영향을 미치며 동시에 실행된다.

Fig. 4는 GPGPU 작업의 스레드를 동시 실행 중인 가상머신의 개수와 상관없이 동일한 규모로 분할 처리했을 때의 성능을 보여준다. 또한, GPGPU 작업을 분할 처리하여 작업 시작 및 종료 횟수가 증가함에 따라 발생하는 오버헤드를 측정하기 위해 기존 환경과 GPGPU 작업 관리 기법을 적용한 후의 GPGPU 작업 성능도 Fig. 5에서 보여준다.

Fig. 4의 실험은 스레드 그룹을 2, 4, 10 그리고 20개로 분할했을 때의 성능을 보여준다. 실험에서 분할된 스레드 그룹의 개수는 ThreadGroup으로 표시하며, 값이 클수록 스레드 그룹이 많아지고 이는 스레드 실행을 위해 작업 시작 및 종료 횟수가 증가하는 것을 뜻한다. Fig. 4의 실험 결과에서 볼 수 있듯이 GPGPU 작업을 분할 실행하였을 때 GPGPU 작업을 수행하는 가상머신으로 인해 발생하는 성능 저하가 완화되는 것을 보여준다. 하지만 GPGPU 작업을 너무 작은 단위로 분할한 경우 CPU 연산 성능은 향상되지만, GPGPU 작업 성능이 저하되는 문제가 발생하기 때문에 GPGPU 작업의 분할 규모에 대한 설정이 필요하다. Fig. 6은 앞서 수행한 실험의 결과를 바탕으로 최적의 분할 규모를 통한 가상머신의 성능을 보여준다.

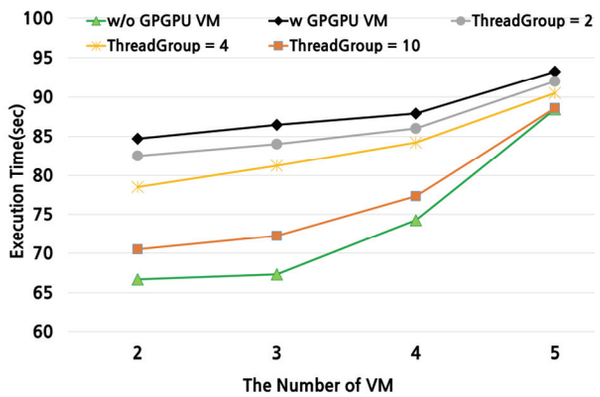

Fig. 4. Performance of Virtual Machine when GPGPU Work is Divided into the Same Scale

Fig. 5는 GPGPU 작업의 분할 규모에 따른 성능을 보여준다. 동시에 실행되는 가상머신이 2개인 경우 단일 수행하는 경우와 전체 스레드를 10개의 그룹으로 분할해서 순차 처리하는 경우 성능 차이가 크지 않았지만 가상머신 5개가 동시에 작업을 처리할 때 GPGPU 작업을 분할 처리하면, 단일 수행 시 성능과 10개의 스레드 그룹으로 분할 처리하였을 때의 성능 차이가 크게 발생하였다. 그리고 Fig. 4와 Fig. 5에서 보여주는 것과 같이 GPGPU 작업의 스레드 그룹을 20개로 분할한 경우 함께 실행되는 CPU 연산을 수행하는 가상머신의 성능은 큰 차이가 없지만, GPGPU 작업의 성능은 크게 저하된 것을 확인할 수 있다.

Fig. 4와 Fig. 5를 통해 동시에 수행되는 가상머신의 개수가 적을 때에는 GPGPU 작업의 스레드 그룹의 개수가 GPGPU 작업에 큰 영향을 미치지 않았지만 동시에 실행되는 가상머신의 개수와 GPGPU 작업의 스레드 그룹의 개수가 증가할수록 GPGPU 작업 성능의 저하를 발생시켰다. 스레드의 분할 개수를 증가시킬수록 CPU 연산을 수행하는 가상머신의 성능은 향상되지만 가상머신이 증가하고 분할된 스레드 그룹의 개수가 증가할수록 GPGPU 작업의 성능은 크게 저하된다. 앞서 수행한 실험을 통해 동시에 실행되는 가상머신의 개수에 따라 차별적으로 스레드 분할 그룹을 결정한다.

Fig. 6은 본 논문에서 제안한 기법을 통해 동시에 실행되는 가상머신의 개수에 따라 앞서 설명한 수식 (1)의 α를 최적의 값으로 설정했을 때의 성능 측정 결과이다. Fig. 6의 그래프에서 실선은 CPU 연산을 수행하는 가상머신의 성능이며, 점선은 기존 환경과 본 논문에서 제안한 기법을 적용했을 때, 가상머신의 GPGPU 작업 성능을 보여준다.

Fig. 5. Performance of GPGPU Operation According to the Size of the Thread Division

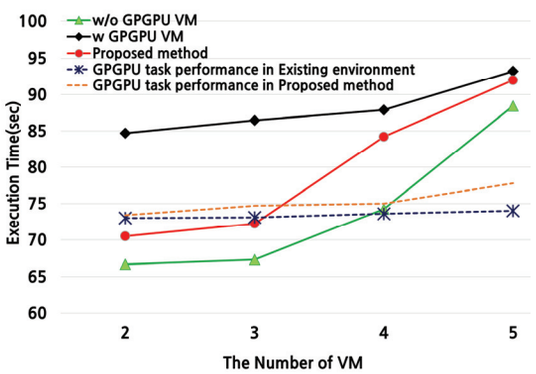

Fig. 6. Performance of Virtual machine Through the Optimal Value of Thread Division Scale

Fig. 6에서 실선으로 구성된 그래프들은 각각 GPGPU 작업을 수행하는 가상머신이 없을 때 다른 가상머신의 성능(w/o GPGPU VM), CPU 연산을 수행하는 가상머신들이 GPGPU 연산을 수행하는 가상머신과 함께 실행될 때(w GPGPU VM) 그리고 본 논문에서 제안한 기법을 적용했을 때의 성능(Proposed method)을 보여주며, 점선으로 구성된 그래프들은 각각 기존 환경과 본 논문에서 제안한 기법을 사용했을 때의 가상머신에서 실행되는 GPGPU 연산 성능을 보여준다.

Fig. 6에서 보여주는 것과 같이 GPGPU 작업을 분할 처리할 때 최적의 분할 규모를 설정하면 GPGPU 작업의 성능을 최대한 유지하면서 CPU 연산을 수행하는 가상머신의 성능이 향상되는 것을 확인할 수 있다. GPGPU 연산을 수행하는 가상머신과 함께 실행되는 CPU 연산 가상머신은 기존 환경과 비교했을 때 최대 20%의 성능이 향상되는 것을 확인할 수 있으며, GPGPU 연산을 수행하는 가상머신이 없는 경우와 비교했을 때 GPGPU 연산을 수행하는 가상머신으로 인한 성능 저하 문제를 효율적으로 해결한다는 것을 확인할 수 있다.

본 논문에서 제안한 GPGPU 작업 관리 기법은 GPGPU 작업을 수행하는 가상머신이 다른 가상머신들에게 미치는 성능 영향을 효율적으로 완화할 수 있다는 것을 실험 결과를 통해 보여준다.

본 논문의 접근 방식은 GPGPU 작업을 적당한 규모로 분할 처리한다. 앞서 3장에서 수행했던 실험 결과와 같이 GPGPU 작업의 분할된 스레드 그룹의 개수가 많을수록 GPGPU 작업의 성능이 저하되는 것을 확인했다. 이로 인해 본 논문에서는 GPGPU 작업의 성능 저하가 최소화되면서 다른 가상머신들의 성능이 향상될 수 있도록 GPGPU 작업의 분할 규모를 결정한다.

Fig. 6에서 점선으로 구성된 그래프가 보여주는 것처럼 본 논문에서 제안한 기법은 GPGPU 작업을 분할 처리하기 때문에 기존 환경처럼 한 번에 처리되는 방식과 비교하면 어느 정도 성능 저하를 발생시킨다. 이는 본 논문에서 제안한 GPGPU 작업 관리 기법을 통해 GPGPU 작업의 스레드 생성과 실행 횟수를 조절하기 때문에 발생하는 성능 저하이다. 하지만 제안한 GPGPU 작업 관리 기법으로 인해 GPGPU 작업의 성능이 5% 미만으로 발생시키면서 다른 가상머신들의 성능은 최대 약 20%가 향상되기 때문에 본 논문에서 제안한 기법이 GPGPU 작업을 수행하는 가상머신으로 인한 성능 영향을 효율적으로 완화하고 다른 가상머신들의 성능을 향상시키는 것을 보여준다.

앞서 수행한 실험의 결과를 통해 GPGPU 작업의 분할 처리 기법을 사용함으로써 GPGPU 작업을 수행하는 가상머신으로 인한 성능 저하 문제를 완화하고 CPU 연산을 수행하는 가상머신의 성능을 향상할 수 있었다. 하지만 분할 규모가 작아지면 CPU 연산을 수행하는 가상머신의 성능이 더 향상되지만, GPGPU 작업 성능이 저하되기 때문에 GPU 클라우드 서비스 제공자 측면에서 서비스의 특징과 사용자에 따라 호스트머신에서 실행 중인 가상머신 개수에 따라 자동 분할시켜 줄 GPGPU 작업의 분할 규모에 대한 설정이 필요하다. 또한, 본 논문에서 제안한 기법은 GPGPU 작업을 분할처리 할 때 GPGPU 작업의 매우 작은 성능 저하만 발생시키며, 시스템 전체에 미치는 성능 영향은 거의 없다.

6. 결론 및 향후 연구

본 논문에서는 GPGPU 작업을 수행하는 가상머신으로 인한 성능 저하 문제를 확인하고 성능 저하 문제를 완화하기 위한 GPGPU 작업 관리 기법을 제안하였다. GPGPU 작업 관리자는 기본적으로 GPGPU 작업 수행 시 스레드들을 분할하고 분할된 스레드 그룹이 작업을 완료하고 다음 스레드 그룹을 시작하기 전에 다른 가상머신들에게 자원 접근 기회를 증가시켜 다른 가상머신들의 성능 저하 문제를 완화한다. 또한, 스레드들을 분할 처리함으로써 작업 시작을 위한 초기화 작업과 종료 시 결과 반환으로 인한 오버헤드가 존재하지만, 이는 무시할 수 있을 정도이며, 본 논문에서 제안한 기법으로 인해 호스트 머신에서 실행 중인 가상머신의 성능에 끼치는 영향은 매우 적다.

본 논문에서 제안한 GPGPU 작업 관리 기법은 실험 결과에서 보여주는 것과 같이 GPGPU 작업과 CPU 연산을 수행하는 가상머신이 동시에 실행되는 동안 가상머신 사이의 성능 영향을 완화할 수 있었으며, GPGPU 작업을 수행하는 가상머신의 성능을 최대한 유지하면서 CPU 연산의 성능을 향상시킬 수 있었다. 하지만 GPGPU 작업의 분할 규모를 너무 작은 수준으로 조정한다면 CPU 연산의 성능은 더 향상되지만, GPGPU 작업의 성능이 저하 폭이 증가한다. 예를 들어 실험에서 사용한 GPGPU 작업을 500개의 스레드 그룹으로 분할 처리 하였을 때 CPU 연산의 성능은 큰 변화가 없었지만, GPGPU 작업의 수행 시간이 약 38%가량 증가되는 문제

가 있었다. GPGPU 작업의 성능 저하는 고성능 연산의 장점을 얻을 수 없는 문제를 발생시킬 수 있기 때문에 클라우드 제공자가 서비스의 SLA(Service Level Agreement)나 기타 사용자들에게 제공해야 하는 성능 측면의 요소들과 GPGPU 작업량을 고려하여 GPGPU 작업의 스레드 분할 규모를 정해야 할 필요가 있다. 하지만 호스트 머신에서 수행 중인 가상머신의 개수에 따라 GPGPU 작업의 분할 규모를 수동으로 결정하는 부분은 클라우드 서버의 자원 사용량이나 연산 성능, 가상머신 스케줄링 정보 등 성능 매개변수 등을 사용한 성능 모델링을 통해 최적화를 할 수 있을 것이다. 또한, 본 논문에서는 CPU 연산을 수행하는 가상머신의 성능 저하를 완화 하는데 초점을 두고 제안한 방법을 사용했을 때 GPGPU 연산을 수행하는 가상머신의 성능 저하를 어느 정도 허용했지만, 이는 공평성과도 관련되어있기 때문에 적절한 균형이 필요하다.

본 논문의 추후 연구로는 GPGPU 작업 규모와 호스트머신에서 실행 중인 가상머신의 개수에 따라 적응적으로 스레드의 분할 규모를 결정하도록 자동화하는 것이다. 본 논문에서 제안한 방법은 스레드 분할규모를 실험을 통해 도출한 매개변수를 통해 결정한다. 하지만 추후에는 GPGPU 작업을 실행하는 가상머신과 함께 실행되는 다른 가상머신의 개수, 자원 사용량과 같은 다양한 요소를 고려한 GPGPU 작업의 성능 모델링을 구축하여 호스트머신의 상태에 따라 보다 정밀하게 작동할 수 있도록 연구할 것이다. 이를 통해 GPGPU 작업의 동적 관리 기법을 개발하여 GPGPU 작업의 성능과 다른 가상머신들의 성능 사이에서 최적의 작업 분할 규모를 도출할 수 있도록 본 연구에서 제안한 GPGPU 작업 기법을 확장할 것이다.

References

[1] W. Hwu and D. Kirk, "Programming Massively Parallel Processors: A Hands-On Approach," Morgan Kaufmann, 2010, pp.20-24.

[2] Amazon, Amazon EC2 Instance Types [Internet]. https://aws.amazon.com/ec2/instanc e-types/?nc1=f_ls.

[3] Alibaba Cloud, Elastic GPU Service [Internet], https://hpc.aliyun.com/product/gpu_bare_metal.

[4] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt and A. Warfield, "Xen and the art of virtualization," In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles, SOSP '03*. ACM: New York, NY, USA, 2003, pp.164-177.

[5] Xen Project, Credit Scheduler [Internet], https://wiki.xen.org/wiki/Credit_Scheduler.

[6] AMD, OpenCL: Open Computing Language [Internet], https://www.khron os.org/opencl/.

[7] nVidia, CUDA: Compute Unified Device Architecture [Internet], http://www.nvidia.com/object/cuda_home_new. html.

[8] Xen Project, VGA Passthrough [Internet], https://wiki.xen.org/wiki/Xen_VGA_ Passthrough.

[9] D. Abramson, J. Jackson, S. Muthrasanallur, G. Neiger,G. Regnier, R. Sankaran, I. Schoinas, R. Uhlig, B. Vembu, and J. Wiegert, "Intel virtualization technology for directed I/O," *Intel Technology Journal*, 2006.

[10] L. Shi, H. Chen, J. Sun, and K. Li, "vCUDA: GPU-accelerated high-performance computing in virtual machines," *IEEE Transactions on Computers*, Vol.61, No.6, pp.804-816, 2012.

[11] Y. Suzuki, S. Kato, H. Yamada, and K. Kono, "Gpuvm: Gpu virtualization at the hypervisor," *IEEE Transactions on Computers*, Vol.65, No.9, pp.2752-2766, 2016.

[12] K. Tian, Y. Dong, and D. Cowperthwaite, "A Full GPU Virtualization Solution with Mediated Pass-Through," *USENIX Annual Technical Conference*, pp.121-132, 2014.

[13] A. A. Sani, K. Boos, S. Qin and L. Zhong, "I/o paravirtualization at the device file boundary," In *ACM SIGPLAN Notices*, Vol.49, No.4, pp.319-332, 2014.

[14] Y. Zhang, P. Qu, J. Cihang, and W. Zheng, "A cloud gaming system based on user-level virtualization and its resource scheduling," *IEEE Transactions on Parallel and Distributed Systems*, Vol.27, No.5, pp.1239-1252, 2016.

강 지 훈

https://orcid.org/0000-0003-4773-6157
e-mail : k2j23h@korea.ac.kr
2011년 배재대학교 게임공학과(학사)
2013년 배재대학교 게임멀티미디어공학과 (석사)
2020년 고려대학교 컴퓨터학과(박사)
2020년 ~ 현 재 고려대학교 정보창의교육연구소 연구교수
관심분야 : Cloud Computing, GPU Virtualization, HPC Cloud

길 준 민

https://orcid.org/0000-0001-6774-8476
e-mail : jmgil@cu.ac.kr
1994년 고려대학교 전산학과(학사)
1996년 고려대학교 전산학과(석사)
2000년 고려대학교 전산과학과(박사)
2001년 ~ 2002년 University of Illinois at Chicago, Post-Doc.
2002년 ~ 2006년 KISTI 슈퍼컴퓨팅센터 선임연구원
2006년 ~ 2010년 대구가톨릭대학교 컴퓨터교육과 교수
2010년 ~ 현 재 대구가톨릭대학교 컴퓨터소프트웨어학부 교수
관심분야 : 빅데이터, 인공지능, 클라우드컴퓨팅, 분산시스템