논문 2016-11-08

CPU-GPU간 긴밀성을 위한 효율적인 공유메모리 접근 방법과 검증 시스템 구현

(Implementation of Integrated CPU-GPU for Efficient Uniform Memory Access Method and Verification System)

박 현 문, 권 진 산, 황 태 호*, 김 동 순
(Hyun-moon Park, Jinsan Kwon, Tae-ho Hwang, Dong-Sun Kim)

Abstract : In this paper, we propose a system for efficient use of shared memory between CPU and GPU. The system, called Fusion Architecture, assures consistency of the shared memory and minimizes cache misses that frequently occurs on Heterogeneous System Architecture or Unified Virtual Memory based systems. It also maximizes the performance for memory intensive jobs by efficient allocation of GPU cores. To test between architectures on various scenarios, we introduce the Fusion Architecture Analyzer, which compares OpenMP, OpenCL, CUDA, and the proposed architecture in terms of memory overhead and process time. As a result, Proposed fusion architectures show that the Fusion Architecture runs benchmarks 55% faster and reduces memory overheads by 220% in average.

Keywords : CPU-GPU, Main memory access architecture, Unified virtual memory access system, GPGPU

Ⅰ. 서 론

GPGPU의 다수 코어를 효율적으로 사용하고 유닛 간의 하드웨어적 구조적 차이와 성능을 극복하기 위해 CPU-GPU간의 공용 메모리 컨트롤러로 메모리 공간을 통일하는 헤테로지니어스(heterogeneous) 기반 이기종 시스템구조(Heterogeneous System Architecture, HSA)가 제안되었다 [1-4]. 두 유닛이 일관성 버스(Coherent Bus)로 하나의 가상 메모리 공간을 메인 메모리에서 공유함으로써 메모리일관성 제공과 이기종 메모리에서 메인 메모리의 복사 없이 GPU와 CPU가 직접 접근하게 되었다 [5]. 하지만 계층적 캐시 구조로 이루어진 CPU는 사이클 간 시간을 최소화하고 순차적인 명령어의 실행속도를 극대화하고 작은 다수 코어(ManyCore)로 이루어진 GPU는 병렬 ALU(Arithmetic Logic Units)들을 이용하여 많은 수의 스레드 처리와 픽셀(pixel) 계산에 중점을 두는 아키텍처를 위해 매우 작고 빠른 L1/L2 캐시를 가진다. 또한, SIMT(Single Instruction Multiple Thread)의 특성상 CPU보다 높은 메모리 대역폭이 필요하다. 이러한 구조적 차이는 라디아나 벤치마크에서 CPU의 L2 캐시미스는 58%인 반면, CPU의 L3 캐시미스는 14%로 메인메모리의 동일 접근 데이터에서 캐시실패와 캐시미스율로 확인가능하다 [6]. 이러한 높은 캐시미스에 따른 오버헤드를 최소하기 위해 이기종 시스템에서 전체 일관성(full coherence) 요구에 따라 CPU 요청에 맞는 핸들링 레지스터(Miss Status Handling Register, MSHR)가 필요하다. GPU에 맞게 수천수만 개의 MSHR 항목을 유지하는 것은 이기종 시스템에 오버헤드가 크다. 따라서 상호간의 스누핑(Snooping)으로 메

*Corresponding Author (taeo@keti.re.kr)
Received: 1 Feb. 2016, Revised: 9 Mar. 2016, Accepted: 22 Mar. 2016.
H.m. Park, J.K. Kwon, T.h. Hwang, D.S. Kim: Korea Electronics Technology Institute
※ 본 논문은 산업통산자원부 산업융합원천기술개발 사업으로 지원된 연구결과입니다. [10041664, 멀티 Shader GPU 통합형 멀티 코어 퓨전 프로세서 원천 기술 개발]

 2016 Apr. 11(2) 57-65
ISSN : 1975-5066
http://dx.doi.org/10.14372/IEMEK.2016.11.2.57

모리일관성을 유지하는 것이 효율적이다 [3]. 상호간의 스누핑은 적은 데이터 공유할 때 매우 유용하지만 비일관성 데이터까지 일관성요구(coherence-request)로 GPU 간섭으로 자원이 낭비되며, 잘못된 캐시 크기로 인한 오버헤드를 가진다. 또한 대역폭이 넘는 큰 데이터를 처리에서 두 유닛 간 필요 이상의 일관성요구로 인한 오버헤드를 가진다 [6, 7].

그 밖에도 NVIDIA와 ATI의 GPU코어와 GDDR5는 GPU의 평균 셀 체류시간(Mean Cells Retention Time, MCRT)은 최대 28ms이지만, DDR3는 최소 64ms이다. 또한, CPU와 DDR3의 IO는 8개이지만, GDDR은 GPU와 16-32개의 IO로 구성된다. 결과적으로 GPU와 CPU의 최대버스폭은 336.3 GB/s, 17.0~68.2GB/s으로 GPU대비 1/5~1/20의 버스폭을 갖는다. 계층간 캐시미스까지 고려해도 CPU의 느린 메인메모리 접근 IO로 인해 두 유닛이 공유한 내부버스의 효율성이 낮아진다 [8, 9]. 따라서 공유메모리구조에서도 CPU에 맞춰서 데이터 접근과 처리하는 GPU의 효율성도 저하된다. 둘째는 GPU는 자원배분 효율적으로 하지 않기 때문에 적은자원에도 많은 셀을 사용하게 되고 이를 통해 다수의 Cell이 메인메모리에 접근하게 된다. 앞서 두 유닛의 서로 다른 스레드 접근 시간과 가상메인 메모리 영역의 데이터 잔존 시간으로 인해 복사와 쓰기에 유닛간의 접근 지연이 발생한다.

메인메모리에 이기종 코어의 테스크(Task)를 직접 공유할 수 있는 별도의 HW 컨트롤러가 제공되면 GPU는 두 유닛의 IO 속도와 상관없이 CPU에 요구하는 연산을 메인메모리에서 직접 수행하고, 가상공유메모리에 따른 지연도 최소화할 수 있다. CPU는 GPU 연산 동안 별도의 프로세싱이 가능하게 된다. 또한 HW 컨트롤러에서 Job에 따라 GPU 셀에 자원을 배분할 수 있다면, GPU Core의 효율성이 증가하고 전력 향상과 메모리 중심의 컴퓨팅(Memory Intensive Computing)에서 높은 성능을 보여줄 수 있다. 본 연구는 CPU와 GPU 간의 직접적인 데이터 복사 없이 CPU가 요청한 데이터의 주소 정보를 제공받아 메모리 테이블 관리자 계층에서 GPU 캐시에 맞게 상호 변환·분배·제어하고, 데이터 로드에 따라 GPU 셀을 할당하는 퓨전 알고리즘을 제안하였다. 기능에 따라 작업관리자(Job Manager)와 리매퍼(Re-mapper)와 프리페처(Re-fetcher)로 분류하여 CPU-GPU 간 데이터 처리에 병목현상(Bottleneck)을 줄이고, GPU의 셀을 효율적으로 배분함으로써 모바일 시스템에서 이기종 시스템구조보다 에너지 절감을 할 수 있다. 부가적으로 전체일관성을 위한 스누핑이 작업관리자에 맞춰 소단위로 유지됨으로 두 유닛간의 오버헤드도 감소하고, GPU의 성능저하가 감소된다.

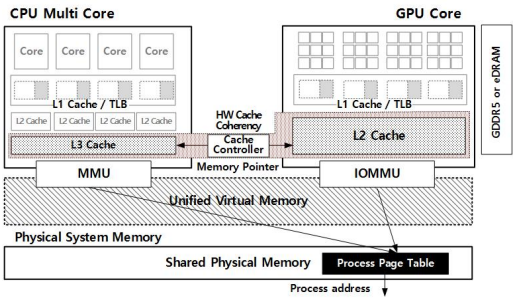


그림 1. 이기종 시스템에서 통합가상메모리 구조
Fig. 1 HSA unified virtual memory architecture

2장에서는 기존의 이기종 시스템구조와 제안된 알고리즘을 설명하였다. 3장은 본 제안 알고리즘에 GPU Cell 효율적인 처리를 로디니아 벤치를 통해 검증하고, 개발된 융합 구조 분석기(Fusion Architecture Analyzer, FAA)를 이용해 제안된 기법과 기존의 OpenCL, OpenMP, CUDA과의 비교성능분석을 하였다.

II. 제안된 이기종 시스템 구조

1. 이기종 시스템 구조

ATI, ARM와 관련기업들은 CPU-GPU간의 성능을 향상과 저전력을 위해 이기종 시스템인 HSA를 제안하였고, 세부적으로 가상화 공유메모리 기술인 hUMA(heterogeneous Uniform Memory Access)와 공유캐시 컨트롤러 기능의 hQ(Heterogeneous Queuing)가 제안되었다 [2, 3]. 그림 1에서 Unified Virtual Memory는 hUMA로 부합되며, 두 유닛간의 메인 메모리영역을 공유한다. hQ는 공유캐시컨트롤러(Uniform Cache Controller)와 데이터 관리자 역할을 한다. GPU에 프로세서 작업을 할당하며 양방향으로 하드웨어가 캐시를 스누프하고 일관성을 자동으로 확보한다. 어느 프로세서가 캐시의 데이터를 갱신했을 경우 다른 프로세서를 탐지할 수 있어 메모리 일관성 에러를 최소화하고 특정작업에 CPU-GPU중 적합한 코어가 처리하도록 한다.

하지만 GDDR은 DDR보다 월등히 빠른 속도로 동작하지만, 공유 가상화 메모리로 인해 GPU의 동작 속도도 DDR에 맞춰질 수밖에 없다. 또한, 앞서

서론에서 서로 다른 아키텍처 구조와 다양한 문제로 CPU의 캐시 버스폭 향상 [10]과 다양한 방안 [7, 8, 11]이 제시되었지만, 전체적으로는 GPU 성능 저하를 극복하기 어렵다.

2. 제안된 융합 구조

1.1 제안 알고리즘의 전체 구조와 기능

본 제안은 그림 2와 같이 L3 캐쉬 코드 정보와 메인메모리 데이터를 개발된 '작업 관리자(Job Manager)'와 '캐시 컨트롤러(Cache Controller)' '리맵퍼(Re-mapper)' 그리고 '프리페처(Pre-fetecher)'로 CPU-GPU간 주소를 공유하고 GPU 코어에서 물리 메인 메모리에 직접적인 접근을 위한 데이터 주소 변환 역할을 통해 GPU의 접근빈도 및 크기에 큰 영향 없이 처리가 가능하고 가상메모리 구조로 인한 오버헤드도 줄일 수 있다. 그 밖에도 데이터를 GPU의 처리량에 맞게 셀을 분배·제어해주는 역할을 통해 메인메모리의 오버헤드를 줄이는 동시에 GPU 코어에 Cell의 개수를 조절함으로써 성능을 최적화하고 전력을 최소화할 수 있다.

그림 3은 작업관리자의 처리 과정을 A), B), C), D)의 4단계로 나타내었다. A)는 CPU 호스트 인터페이스와 같이 버스와 브리지를 통해 GPU의 구동

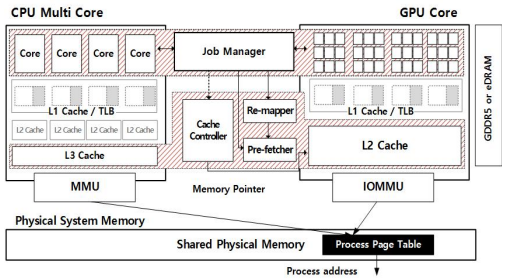


그림 3. 제안된 융합 알고리즘 구조
Fig. 3 Proposed fusion architecture

가능하고, CPU는 B)와 같이 GPU 컴파일된 코드와 데이터, 그리고 GPU 코어별로 분할된 데이터의 메모리 주소, 코어별 오프셋 정보, 파라미터 등 기반 정보를 Job Issue 메시지로 GPU에 전달한다. 즉 GPU에 처리하는 데이터 범위와 위치가 정의된다. 작업관리자는 현재 처리되는 `애플리케이션의 L2 Memory Access Latency의 평균을 내어 범위(scalability)를 산출하고 할당될 Shader Core의 개수를 정의한다. Shader Core개수는 연산속도에도 영향을 주지만 L1 Cache와 L2 Cache간의 캐시 미스(Cache miss)에 큰 영향을 주기 때문에 최적화하는 과정도 중요하다. C) GPU의 각 코어 작업 할당

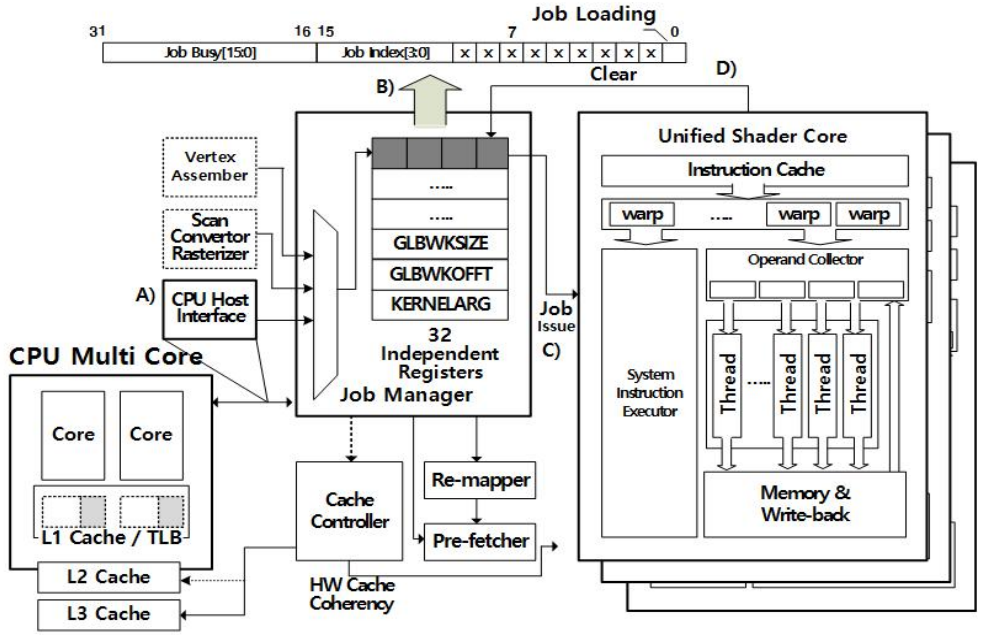


그림 2. 제안된 작업관리자 처리 과정
Fig. 2 Processing of job manager in proposed fusion architecture

을 알려준다. 할당된 작업에 따라서 GPU는 메인 메모리에 접근해서 처리하게 된다. GPU에서 처리하는 동안 A)는 다음에 처리할 데이터를 프리페처에게 요구하게 된다. 프리페처는 L2에 다음 처리할 데이터를 가져오고 이미 처리된 데이터는 메인 메모리에서 해당 데이터를 Flush한다. D) GPU는 위임받은 데이터 작업이 끝나면 완료(Clear) 신호를 작업관리자에게 보내며 작업관리자는 CPU에 GPU의 작업완료를 전달한다.

리매퍼와 프리페처는 앞서 작업관리자로 가상 메모리가 아닌 물리적인 시스템 메모리에 접근하기 때문에 오버헤드를 가진다. 하지만 오버헤드가 4 Cycles 이내기 때문에 시스템에 큰 영향을 주지 않는다. 또한, CPU가 사용하는 메인 메모리의 페이지 테이블 주소를 GPU에서 접근하기 때문에 캐시 미스도 줄어든다.

프리페처는 GPU가 처리할 일을 분할(Partition)하는 역할을 한다. L3의 캐시 영역을 GPU의 코어가 한 번의 작업에 필요한 공간의 2배를 예약하고 이를 두 개의 윈도우(window)로 구분한다. 첫 번째 윈도우(windows 0)에는 현재 GPU의 작업 데이트를 로딩하고, 두 번째 윈도우 영역(windows 1)에는 이어서 처리할 작업을 위한 데이터의 로딩을 위한 예약 작업을 한다. 이렇게 작업된 윈도우는 GPU의 메모리 Latency hiding을 위해 사용된다. 작업이 끝난 GPU는 작업관리자에게 L2 Cache 작업의 완료를 알리고, 리매퍼로 받은 공유된 메인 메모리 번지에 처리결과를 이동한 후 GPU의 L2/L1 캐시를 지운다.

1.2 GPU의 효율적 할당 방법

작업관리자는 GPU 코어에 작업을 할당하는 역할을 한다. GPU는 할당된 작업을 수행하면서 일정 주기마다 메모리 응답 시간 정보를 작업관리자에게 전달하며, 작업관리자는 개별 GPU 코어에 전달받은 메모리 응답 시간을 일정 개수만큼 저장하여 이들의 평균(Average Memory Latency, AML)을 계산한다. 그림 4는 최적화된 GPU 코어 개수(n optimize)를 구하기 위한 알고리즘으로, 작업관리자는 GPU 각 코어의 일정주기마다 메인 메모리 응답 시간(Main Memory Response Time)을 전달 받는다. 작업관리자는 수신된 응답시간을 전체 GPU의 평균 응답시간(AML)을 계산한다. 계산된 값을 가지고, 지정된 Latency threshold high 값과 비교하여 AML 값이 더 클 경우 목표하는 코어 개수(n optimize)를 줄인다. 만약 AML 값이 더 크다면 코어 개수는

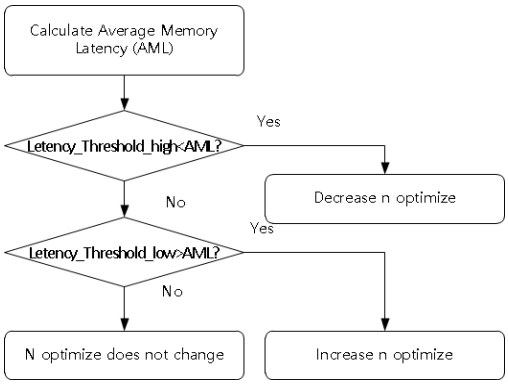


그림 4. GPU 코어 개수를 구하는 알고리즘
Fig. 4 Optimize algorithm to number of GPU cores

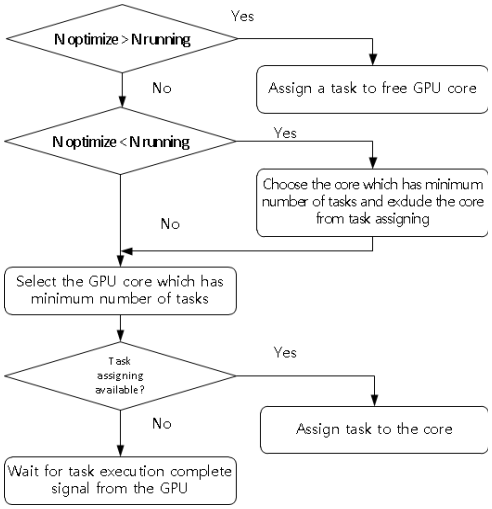


그림 5. GPU 코어에 할당 알고리즘
Fig. 5 GPU cores allocation algorithm

현재와 같이 유지한다. 코어 수 결정을 위한 임계치의 최고와 최저값(Latency Thre- shold Low, High)는 코어마다 다르므로 실험적으로 정의한다.

그림 5는 그림 4의 최적화된 코어 개수(n optimize)를 가지고 GPU 코어에 할당하는 알고리즘 구조도이다. 최적화된 코어 수가 현재 동작하는 코어 수(Nruning) 보다 크면, 작업이 할당되지 않은 GPU의 코어를 할당한다. 최적화된 코어 수가 현재 동작하는 코어 수보다 작으면, 최적화된 코어 수와 동작하는 코어 수를 비교한다. 현재 동작하는 코어 개수가 최적화된 코어 수보다 많다면, 현재 동작하고 있는 코어 중 작업이 할당되지 않는 코어에 코

의 할당작업을 추가하며, 전체 코어가 동작한다면, 할당 작업이 가장적은 코어에 할당한다. 작업 할당량은 GPU 각 코어에 할당된 스레드 양으로 정의한다.

만약, 전체 코어 작업할당이 불가능하다면, GPU 코어에서 작업 완료 신호를 기다린다.

Ⅲ. 제안 기법의 성능 비교

3장의 성능비교분석은 4장의 표1과 같은 환경에서 기존 GPU를 이용한 기법과과 제안된 GPU-CPU 공유알고리즘을 비교분석을 수행하였다.

1. 기존의 GPU 자원 효율성 분석

기존의 이기종 시스템구조의 hQ는 단순히 GPU에 프로세서 작업을 할당하는 구조로 되어 있기 때문에 해당 GPU가 얼마만큼의 셀을 할당하고 처리하는지는 관여하지 않는다. 따라서 GPU 자원 낭비의 문제점을 가진다. 그림 6는 GPU 자원 낭비에 대한 성능 분석 위해 오픈 소스 프로젝트에서 추출하거나 로디니아 벤치(Rodinia Benchmark)를 이용하여 성능 분석을 하였다 [10, 12].

그림 6, 7, 8에서 왼쪽 4개의 LavaMD, Hotspot 등은 Compute-intensive를 측정 가능한 애플리케이션이며, 오른쪽 4개의 NW, Srad는 Memory-intensive 분석 애플리케이션이다. 그림 6과 같이 왼쪽 4개의Compute-intensive 계열 알고리즘 중 LavaMD와 Backprop는 코어 수에 따라 비례한 성능을 보이며, Pathfinder, Backprop은 조금 낮지만 증가됨을 확인할 수 있다. 하지만 메모리의 의존도가 높은 Memory-intensive 알고리즘들 중 Bfs는 코어 변화 따라 미약하게 성능이 증가한 반면에 Kmean는 코어 8개보다 16개가 오히려 성능이 감소됨을 알 수 있었다.

그림 7의 코어 개수가 변화함에 따른 IPC 코어 1개를 기준으로 정규화한 그래프로 Compute-intensive는 코어 수 증가에 따라 개별 코어 성능이 크게 떨어지지 않지만, Memory-intensive는 코어 수 증가에 따라 개별 코어 성능이 크게 떨어지는 것을 알 수 있다. 평균으로 볼 때 코어 1개일 때보다 16개에서는 Memory-intensive에서는 16% 수준으로 IPC가 크게 감소하는 반면에, Compute-intensive는 81.3%으로 감소폭이 작았다. CPU-GPU메모리 공유구조에서 어떤 형태의 응용프로그램 인지를 판별하여 그 특성에 따라 GPU 코어 할당을 하는 것이 중요하다.

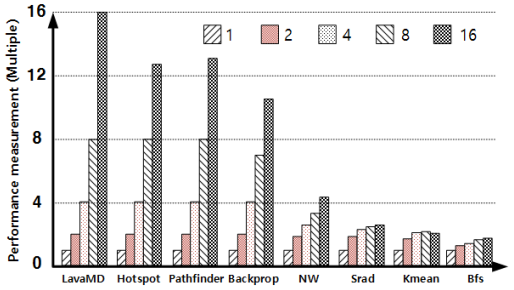


그림 6. GPU 코어에 따른 컴퓨팅과 메모리 집중 어플리케이션의 성능 그래프

Fig. 6 Performance graph are GPU core based on compute/memory intensive application type

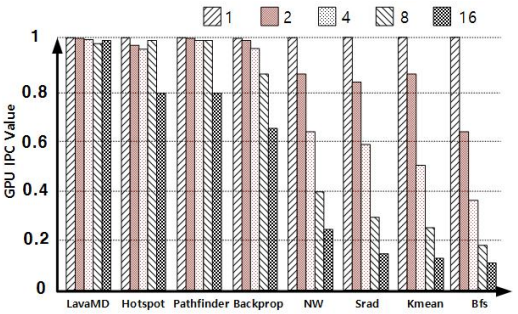


그림 7. GPU 코어 개수에 따른 IPC 변화

Fig. 7 IPC variation with the number of GPU core

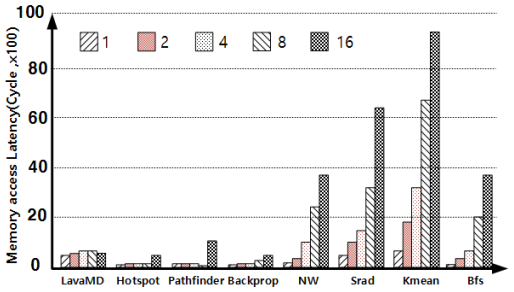


그림 8. GPU 코어수와 어플리케이션에 따른 memory access latency의 변화

Fig. 8 Performance memory access latency graph of GPU core and applications

그림 8은 코어 수의 변화에 따른 Memory access latency의 변화를 나타낸 그래프로 기본 단위는 백회이다. Compute-intensive는 Memory access latency가 16코어 환경에서 평균 690

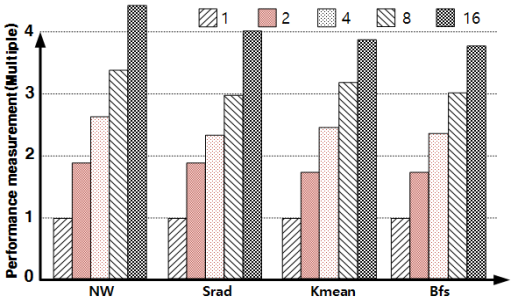


그림 9. GPU 코어에서 메모리 집중 어플리케이션
Fig. 9 Benchmark graph are GPU core based on memory intensive application type

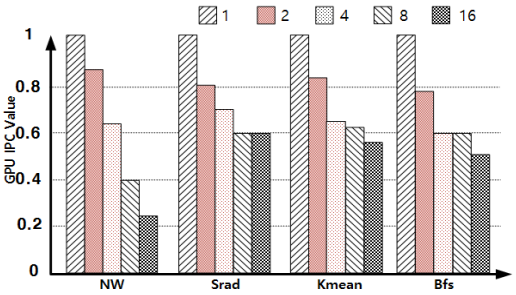


그림 10. GPU 코어 개수에 따른 IPC 변화
Fig. 10 IPC variation with the number of GPU core by memory intensive application

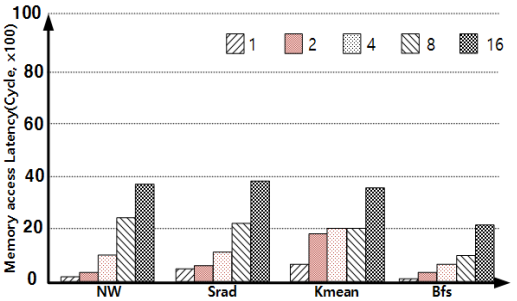


그림 11. GPU 코어수와 어플리케이션에 따른 Memory access Latency의 변화
Fig. 11 Performance memory access latency graph of GPU core and applications

사이클 정도의 Memory access latency로 코어 수 증가에 따른 변화가 크지 않는 반면, Memory-intensive는 코어 수가 증가함에 따라 Memory access latency가 약 4천~9천 사이클로 크게 증가했다. 코어 수가 많아질수록 여러 코어에서 동시에 많은 공유메모리 접근이 일어나므로 이러한 문제가 발생하게 된다.

2. 제안된 GPU 효율성 분석

앞서 제안된 알고리즘에 맞춰 GPU 자원에 효율성을 분석하였다. GPU 자원 낭비에 대한 성능 분석을 위해 로디니아 벤치(Rodinia Benchmark)와 오픈소스를 이용하였다 [12]. 앞 장에서 논의한 Computing / Memory-intensive 중, GPU 코어에 따라 성능이 향상되지 못한 Memory-intensive의 응용어플리케이션인 NW, Srad, Kmean, Bfs를 비교분석하였다. 그림 9와 그림 6의 비교에서 본 제안 알고리즘이 NW에서는 큰 향상이 없었지만, Srad와 Kmean, Bfs에서는 GPU 코어가 증가 될수록 높은 성능을 보여줄 수 있었다.

그림 10은 본 제안된 알고리즘의 IPC 변화를 나타냈다. 점으로 표기한 것이 제안알고리즘이다. IPC의 경우 2 Core 전체적으로 성능이 감소하거나 많이 증가하지 않은 것으로 나타났다. 또한 NW와 같은 알고리즘에서는 향상이 없었다. 반면, Strad, Kmean, Bfs 알고리즘의 8, 16개의 코어 비교에서는 성능이 크게 향상되었으며, 특히 Kmean에서 가장 크게 향상되는 것으로 나타났다. 그림 11에서도 그림 10의 IPC 결과와 비슷한 결과를 그래프로 나타낼 수 있었다. Latency는 NW는 그림 8과 비교했을 때 큰 변화가 없는 반면에 Stad, Kmean, Bfs에서 코어가 증가할수록 성능 향상 효과가 컸으며, 4코어 이하에서는 성능향상 효과가 미진했다.

3. FAA를 통한 성능분석

선행연구에서 자체 개발한 FAA(Fusion Architecture Analyzer)를 QEMU와 연동하여 개발하였다 [13]. 하지만 FAA를 QEMU의 연동에서 성능과 상관없는 메인메모리의 오버헤드가 발견되었다. 따라서 이를 개선하기 위해 그림 12과 같이 OpenMP v4.0, OpenCL v1.2, CUDA v6.5, Fusion v1.1을 라이브러리, 컴파일러로 분리하고, 별도의 융합구조 에뮬레이터를 QEMU에 동작하던 해당 라이브러리와 컴파일러를 하드웨어기반으로 독립시켰다. UI는 Windows 기반으로 제작하였으며, UI는 요소 테스트를 하는 Control Block과 개별 그래픽을 컨트롤하는 Analysis Block으로 구분되며, 이를 측정하는 개별 코어와 알고리즘별 분석시간과 분야별 요소가 확인할 수 있다.

표 1은 벤치마크한 환경을 나타내었다. CUDA 성능과 비교분석하기 위해서 NVIDIA사 X1 모바일

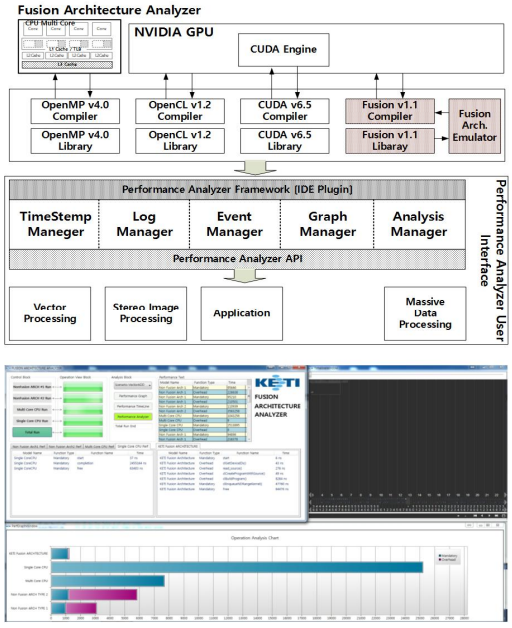


그림 12. FAA 개발 플랫폼 구조 및 UI
Fig. 12 Development of FFA platform arch. and system UI

표 1. FAA 벤치마크환경
Table 1. The specification on FAA

	System Information
CPU-RAM	i7 4790K 24GB DDR3
GPU	nvidia Quadro K620
GPU Core	384 CUDA Cores
GPU RAM	2 GB DDR3
CPU-GPU Bus Band	PCI Express 2.0 x16
GPU Bandwidth	29.0 GB/s

칩의 GPU와 재원이 비슷한 K620을 선택하였다.

그림 13, 14는 연속된 벡터 덧셈과 연속 곱셈을 한 처리한 결과이다. Mandatory는 순수한 연산을 나타내며, Overhead는 메인메모리 공유 아키텍처에서 발생하는 메인메모리 오버헤드를 나타낸다. 앞장의 GPU코어에 따른 알고리즘에 비교에서 메모리접근지연으로 발생되는 오버헤드가 전체 시스템 계산시간에 영향을 주면서, 전체 수행시간이 지연되었다. 제안된 방식이 순수 연산에서는 다른 아키텍처와 비슷한 성능을 보인다. 하지만 OpenCL, CUDA 대비 메인메모리 오버헤드는 가장 적으며, Open MP는 다중 CPU 연산을 통해서만 계산하기 때문에

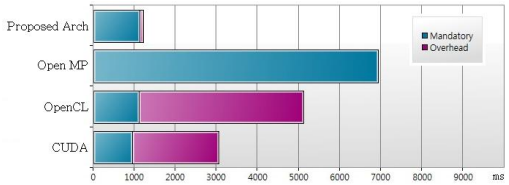


그림 13. 벡터 덧셈 성능 비교
Fig. 13 Vector addition benchmark

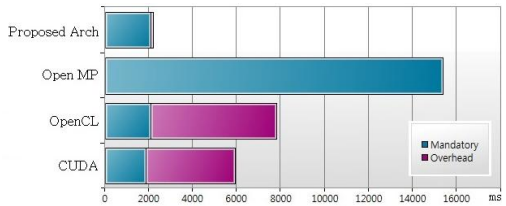


그림 14. 벡터 곱셈 성능 비교
Fig. 14 Vector multiplication benchmark

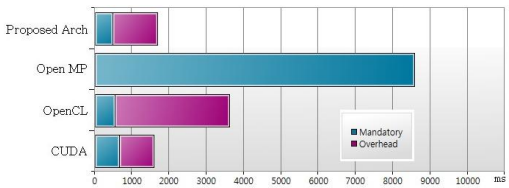


그림 15. 이미지 압축 성능 비교
Fig. 15 Image reverse benchmark

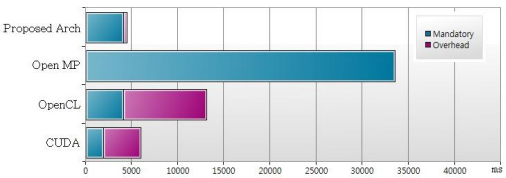


그림 16. 스테레오 이미지 성능 비교
Fig. 16 Comparison of the performance of stereo Image filter benchmark

GPU와의 공유메모리 인한 오버헤드가 없는 것을 알 수 있다.

제안된 알고리즘은 GPU-CPU 연동에서 메인메모리에서 직접 읽고 쓰기 때문에 오버헤드가 작지만 Open, CUDA는 경우 가상메모리 영역으로 인해 GPU-CPU의 공유메모리에 복사가 빈번함에 따라 오버헤드에 180~300%를 차지하였다. Open MP는 메모리 오버헤드는 없지만, 연산시간이 가장 긴 것

으로 나타났다.

그림 15, 16은 이미지 압축과 스테레오 이미지 변환으로 성능 평가를 하였다. 이미지 압축 성능에서는 OpenCL보다 빠른 처리속도와 메인메모리의 오버헤드도 적었다. 하지만 CUDA에 비해서 상대적인 메모리 오버헤드가 컸으며, 전체처리시간도 CUDA 아키텍처보다 3~4%정도 느린 것으로 나타났다. 본 제안의 기법은 앞장의 Compute-intensive의 알고리즘 비교와 그림 15의 결과로 볼 때 제안된 기법의 개선 필요가 있다는 것을 알 수 있다. 그림 16의 스테레오 이미지 필터의 벤치마크 결과에서 제안된 알고리즘이 150~330%가량 메모리 효율성이 높았다.

4가지 분석에서 Compute-intensive 관련된 성능 개선이 필요하며, 향후 연구에도 CPU 코어 중심에 처리에서 GPU의 효율성과 CPU의 메인메모리 접근의 효율성을 높이는 추가적인 알고리즘이 요구된다.

IV. 결 론

본 연구는 효율적인 메모리 접근 시스템으로 융합알고리즘과 GPU 코어 할당 알고리즘을 제안하였다. CPU-GPU간의 공유메모리 알고리즘에서 메모리 중심에 처리방식은 기존의 CUDA나 OpenCL보다 우월한 것으로 나타났다. 또한, GPU 주위의 연산방식의 메인메모리 공유 및 할당에서 효율성을 보였다. 벡터의 덧셈과 곱셈 그리고 스테레오 이미지 성능 비교에서는 최소 55%부터 최대 330%, 평균 220%의 높은 성을 보여줄 수 있었다. 하지만 이미지 압축 성능과 같이 CPU 중심에 GPU 공유메모리 아키텍처에서는 CUDA에 비해 메모리 효율성이 낮은 것으로 나타났으며, 앞으로는 결과분석에서 나타난 문제를 개선하는 연구를 진행할 예정이다.

References

[1] J. Lee, N.B. Lakshminarayana, Kim H., R. Vuduc, "Many-thread aware prefetching mechanisms for GPGPU applications," Proceedings of 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 213-224, 2010.

[2] P. Rogers, A.C. Fellow, "Heterogeneous system architecture overview," Proceedings of Hot Chips, Vol. 25. 2013.

[3] J. Power, A. Basu, J. Gu, S. Puthoor, B.M. Beckmann, M.D. Hill, D.A. Wood, "Heterogeneous system coherence for integrated CPU-GPU systems," Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 457-467. 2013.

[4] Y.H. Park, C.H. Kim, J.M Kim, "Implementation and Performance Evaluation of the Faddev-Leverrier Algorithm using GPGPU," IEMEK J. Embed. Sys. Appl., Vol. 8, No. 6, pp. 171-178, 2013 (in Korean).

[5] G.Y. Jeong, J.H. jeong, H.C. Lee, G.G. Jeon, J.H. Cho, "Efficient Implementation of Candidate Region Extractor for Pedestrian Detection System with Stereo Camera based on GP-GPU," IEMEK J. Embed. Sys. Appl., Vol. 8, No. 2, pp. 121-128, 2013 (in Korean).

[6] S. Che, M. Boyer, J. Meng, D. Tarjan., J.W Sheaffer, S.H. Lee, K. Skadron, "Rodinia: A benchmark suite for heterogeneous computing," Proceedings of IEEE International Symposium on Workload Characterization, pp. 44-54. 2009.

[7] J. Feehrer, P. Rotker, M. Shih, P. Gingras, P. Yakutis, S. Phillips, J. Heath, "Coherency hub design for multisocket sun servers with coolthreads technology," IEEE Micro, Vol. 29, No. 4, pp. 36-47, 2009.

[8] I. Singh, A. Shriraman, W. Fung, M. O'Connor, T. Aamodt, "Cache coherence for GPU architectures," Proceedings of IEEE 19th International Symposium on High Performance Computer Architecture, pp. 578-590, 2013.

[9] P. Hammarlund, R. Kumar, R.B. Osborne, R. Rajwar, R. Singhal, R. D'Sa, S. Gunther, "Haswell: The fourth-generation Intel core processor," IEEE Micro, Vol. 34, No. 02, pp. 6-20, 2014.

[10] K. Wang, X. Ding, R. Lee, S. Kato, X. Zhang, "GDM: Device memory management for gpgpu computing," Proceedings of The 2014 ACM international conference on Measurement and modeling of computer systems, pp. 533-545, 2014.

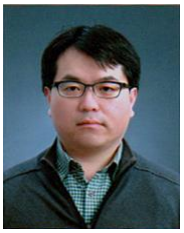
[11] O. Kayiran, N.C. Nachiappan, A. Jog, R., Ausavarungnirun, M.T. Kandemir, G.H. Loh,

C.R. Das, “Managing GPU concurrency in heterogeneous architectures,” Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 114-126, 2014.

[12] Rodinia Benchmark Group. The Rodinia Benchmark Suite version 3.1. 2015.

[13] H.M. Pack, J.S. Kwon, T.H. Gwang, D.S. Kim, “A Development of Fusion Processor Architecture for Efficient Main Memory Access in CPU-GPU Environment,” Journal of KIECS, Vol. 11, no. 2, pp. 151-158, 2016 (in Korean).

Hyun-moon Park (박 현 문)



He received the M.S. degree in Telecommunication Engineering from, Kookmin University, Korea, in 2006. He is currently a Ph.D. candidate in Business of Information Technology at Kookmin University, in 2010. He was a Research Engineer in SW-SoC Institute of ETRI. from 2010-2014. Now, Senior Research Engineer of Korea Electronics Technology Institute of IoT Platform Research Center. His research interest includes IoT Platform, and embedded system.

Email: kimagu@keti.re.kr

Jinsan Kwon (권 진 산)

He received the B.S. degree in Computer Information from, Korea University, Korea, in 2011.

2013 M.S, School of Computer Information at Korea University. Now, Associate Research Engineer of Korea Electronics Technology Institute of IoT Platform Research Center. His research interest includes embedded system, and wearable system.

Email: jinsan.kwon@keti.re.kr

Tae-ho Hwang (황 태 호)

He received the M.S. degree in Electronic Engineering from, Hankuk University of Foreign Studies, Korea, in 1997. He is currently a Ph.D. candidate in Computer Science at Hankuk University of Foreign Studies, in 2013. Now, Senior Research Engineer of Korea Electronics Technology Institute of IoT Platform Research Center. His research interest includes RTOS, WPAN/WBAN and embedded system software.

Email: taeo@keti.re.kr

Dong-Sun Kim (김 동 순)

He received the M.S. degree in Electronic Materials Science from, Inha University, Korea, in 1997. He is currently a Ph.D. candidate in Electronic Materials Science of Media Systems at Inha University, in 2005. Now, Senior Research Engineer of Korea Electronics Technology Institute of IoT Platform Research Center. His research interest includes multimedia SoC Design and embedded hardware.

Email: dskim@keti.re.kr