Regular paper

Journal of the KIECS. pp. 151-158, vol. 11, no. 2, Feb. 29 2016, t. 70, pISSN 1975-8170 | eISSN 2288-2189
http://dx.doi.org/10.13067/JKIECS.2016.11.2.151

CPU-GPU환경에서 효율적인 메인메모리 접근을 위한 융합 프로세서 구조 개발

박현문* · 권진산** · 황태호*** · 김동순****

A Development of Fusion Processor Architecture for Efficient Main Memory Access in CPU-GPU Environment

Hyun-Moon Park* · Jin-San Kwon** · Tae-Ho Hwang*** · Dong-Sun Kim****

요 약

이기종시스템 구조(HSA)는 두 유닛의 각각에 메모리 폴(pools)이 가상메모리를 통해 공유할 수 있게 됨에 따라 CPU와 GPU 아키텍처의 오랜 문제를 해결하였다. 그러나 물리적 실제 시스템에서는 가상메모리 처리를 위해 GPU와 GPU 사이의 빈번한 메모리 이동으로 병목현상(Bottleneck)과 일관성 요청(Coherence request)의 오버헤드를 갖게 된다. 본 연구는 CPU와 GPU간의 효율적인 메인 메모리 접근방안으로 퓨전프로세서 알고리즘을 제안하였다. CPU가 요청한 처리할 메모리 영역을 GPU의 코어에 맞게 분배·제어해주는 기능으로 작업관리자(Job Manager)와 Re-mapper, Pre-fetcher를 제안하였다. 이를 통해 CPU와 GPU간의 빈번한 메시지도 감소되고 CPU의 메모리주소에 없는 Page-Table 요청이 낮아져 두 매체간의 효율성이 증대되었다. 제안한 알고리즘의 검증 방안으로 QEMU(:short for Quick EMUlator)기반의 에뮬레이터를 개발하고 CUDA(:Compute Unified Device. Architecture), OpenMP, OpenCL 등의 알고리즘과 비교평가를 하였다. 성능평가 결과, 본 연구에서 제안한 융합 프로세서 구조를 기존과 비교했을 때 최대 198%이상 빠르게 처리되면서 메모리 복사, 캐시미스 등의 오버헤드를 최소화하였다.

ABSTRACT

The HSA resolves an old problem with existing CPU and GPU architectures by allowing both units to directly access each other's memory pools via unified virtual memory. In a physically realized system, however, frequent data exchanges between CPU and GPU for a virtual memory block result bottlenecks and coherence request overheads. In this paper, we propose Fusion Processor Architecture for efficient access of main memory from both CPU and GPU. It consists of Job Manager, Re-mapper, and Pre-fetcher to control, organize, and distribute work loads and working areas for GPU cores. These components help on reducing memory exchanges between the two processors and improving overall efficiency by eliminating faulty page table requests. To verify proposed algorithm architectures, we develop an emulator based on QEMU, and compare several architectures such as CUDA(Compute Unified Device Architecture), OpenMP, OpenCL. As a result, Proposed fusion processor architectures show 198% faster than others by removing unnecessary memory copies and cache-miss overheads.

키워드

CPU-GPU, GPGPU, Uniform Memory Access, HSA, Fusion architecture.
CPU-GPU, 범용 GPU, 공유 기억 장치 접근, 이기종 시스템, 융합 구조

* 전자부품연구원 IoT플랫폼연구센터(kimagu@ keti.re.kr)
** 교신저자 : 전자부품연구원, IoT 플랫폼연구센터
*** 전자부품연구원 책임연구원(taeo@keti.re.kr)
**** 전자부품연구원 책임연구원(dskim@keti.re.kr)
• 접 수 일 : 2015. 12. 22
• 수정완료일 : 2016. 02. 13
• 게재확정일 : 2016. 02. 24
• Received : Dec. 22, 2015, Revised : Feb. 13, 2016, Accepted : Feb. 24, 2016
• Corresponding Author : Jin-San Kwon
IoT Platform Research Center
Email : jinsan.kwon@keti.re.kr

Ⅰ. 서 론

3D 처리를 하던 GPU는 스마트디바이스의 저전력·고성능 처리를 위해 OpenCL, CUDA, OpenMP 등 GPGPU (:General Purpose computing on GPU)를 도입하면서 기존 CPU가 맡았던 응용프로그램들의 계산을 담당하게 되었다[1.10-12]. 그러나 CPU는 사이클과 사이클 간의 시간을 최소화하는 방향으로 설계되었고 GPGPU는 GPU의 멀티코어를 이용해 CPU보다 전력대비 단위 시간당 처리량을 높여 탁월한 연산 성능을 극대화한다. CPU와 GPU의 메모리 접근 방식이 다를 수밖에 없으며, 두 유닛의 버스 간 연결로 인한 성능제약과 캐시(Cache), 복잡한 메모리 구조 모델 등 근본적인 하드웨어 구조의 차이로 인한 잦은 데이터 동기화, 메모리간의 복사와 주소 변환으로 컴퓨팅 성능의 한계가 존재했다. 이런 두 유닛간의 하드웨어 구조적 차이와 성능을 극복하고 증가하는 GPGPU 코어의 효율적인 사용을 위해 CPU-GPU간의 공용 메모리 컨트롤러를 통해 메모리 공간을 통일하는 헤테로지니어스(Heterogeneous)기반 이기종 시스템구조(HSA)가 제안되었다[1-4]. 헤테로지니어스한 CPU-GPU간 공유메모리접근 방법은 ARM, ATI, nVIDIA, 삼성 등에서도 활발하게 연구되고 있다[1]. 하지만 HSA도 CPU와 GPU의 두 유닛에 근본적인 아키텍처 차이로 공유한 내부 버스를 이용한 연속된 대용량의 데이터 처리 문제가 존재한다. 첫째는 개별 유닛의 L1, L2 캐시와 메인 메모리간의 빈번한 복사가 발생된다. 둘째는 서로 다른 메모리 복사와 쓰기속도가 다른 차이로 유닛간의 접근 지연이 발생한다. 이로 인해 두 유닛간의 캐시미스, 데이터 병목 그리고 캐시일관성의 오버헤드가 여전히 존재한다[3-4]

본 연구는 기존 문제의 해결 방안으로 CPU와 GPU간의 직접적인 데이터 복사 없이 CPU가 요청한 데이터의 주소 정보를 수신하고 GPU의 코어에 맞게 메모리 테이블 관리자 계층에서 상호 변환·분배·제어하는 퓨전프로세서 알고리즘을 제안하였다. 세부적으로 기능에 따라 작업관리자(Job Manager)와 리매퍼(Re-mapper)와 프리페처 (Re-fetcher)로 분류하여 CPU-GPU간 데이터 처리에 병목현상(Bottleneck)을 줄이고, 효율적인 캐시의 일관성을 유지로 메모리 복사, 캐시미스 등의 오버헤드를 최소화하였다. 또한 기존 OpenGL, OpenMP, GUDA과 제안된 융합알고리즘의 비교에서도 최대 198%의 성능향상을 확인할 수 있었다.

Ⅱ. 이기종 시스템 구조와 퓨전아키텍처의 비교

2.1 이기종 시스템 구조(HSA : Heterogeneous System Architecture)

기존 CPU와 GPU는 주소체계가 서로 다른 NUMA(:Non-Uniform Memory Access)을 갖는다. 하지만 양방향의 데이터를 고속화와 효율성을 위해 통합 공유메모리(Unified Shared Memory)가 제안되고 두 유닛의 공통적인 주소공간과 주소체계 변환에 대한 공통적인 처리의 필요성이 제기되었다[1,9].

그림 1과 같이 이기종 시스템 구조에서는 GPU는 CPU를 통해 처리해야 할작업들의 코드와 데이터를 컨트롤러를 통해 전달받게 된다. 이후, 물리적인 메인 메모리를 사용하기 위해 통합 가상화 메모리(Unified Virtual Memory)로 접근하게 된다. 이렇게 함으로써 프로세서가 캐시의 데이터를 갱신했을 때 메모리 일관성으로 에러가 발생하지 않고 시스템 메모리 전부를 GPU가 접근할 수 있게 된다.

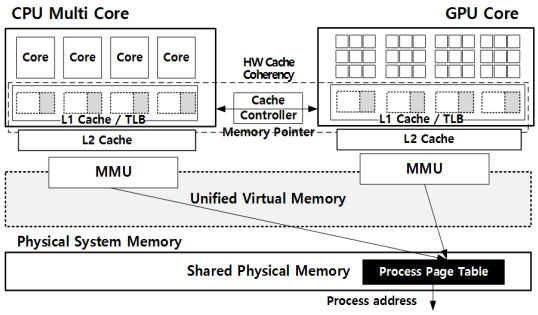


그림 1. 이기종 시스템에서 통합가상메모리 구조
Fig. 1 HSA unified virtual memory architecture

이기종 시스템구조에서 CPU와 GPU는 TLB (:Translation Lookaside Buffer) misses가 각각 5-15%, 29%씩 발생된다[4]. TLB는 최근에 사용된 주소 변환 정보를 저장하고 있는 Page Table에 대한

캐시(Cache)로 GPU, CPU에 각각 존재한다. TLB는 일반적으로 L1 캐시인 32~128KB보다 작은 사이즈로 4~8KB를 가지며 Intel 6세대코어 L1은 Cache Latency인 4 Cycles보다 짧은 1 Cycle을 갖는다[5-6]. 하지만 선행연구[4]와 같이 TLB misses가 발생하면 30 Cycles이상의 지연 오버헤드가 발생한다.

또한, 가상메모리구조인 HSA도 GPU의 멀티코어를 이용한 분할 및 순차적인 처리에서 TLB 변환 정보에 재사용이 상대적으로 낮고, GPU에서 처리할 데이터가 클 경우 TLB에서 통합 가상화 메모리의 페이지 테이블로 접근하는 횟수가 크게 증가하게 된다. 더욱이 많은 GPU 코어들이 각각의 TLB를 갖고 메모리버스에 접근할 때 더욱 많은 트래픽이 발생하게 된다. 따라서 큰 트래픽으로 인한 가상화된 메모리영역의 접근 지연이 발생하거나 실패가 발생한다. 접근 지연으로 CPU는 TLB에 가상 메모리 주소에 없는 Page Table을 요청하게 되고 TLB 실패(Cache miss)가 발생한다. TLB 실패로 페이지 테이블 접근과 생성이 반복되어 가상공통메모리에 대한 CPU와 GPU의 전체적인 성능이 감소하게 된다. 특히 GPU에서 개별 쓰레드를 처리할 때 인터럽트가 발생하게 된다. GPU가 별도의 MMU를 통해 가상주소를 사용하더라도 CPU에서 사용하는 주소영역과는 다르기 때문이다. 결국은 GPU가 바라보는 주소 공간을 CPU가 사용한 메인 메모리 페이지 테이블 이용하여 주소 공간을 매핑하고 메인메모리에서 일부의 복사가 이루어진다.

2.2 융합 구조(Fusion Architecture)

HSA는 가상메모리구조로 인해 4~12 cycle 정도의 오버헤드[4-6] 갖는다. 가상메모리 구조의 오버헤드를 줄이면서, 2.1장의 문제 해결 위해서는 GPU를 메모리 접근 제어와 함께 CPU의 공유메모리 로드를 줄여 TBL miss를 줄이고 캐시일관성 유지로 GPU의 효율성 높이는 방안이 요구된다.

CPU가 요청한 처리 영역을 별도의 관리자가 GPU의 코어에 맞게 분배·제어해준다면 CPU와 GPU간의 빈번한 메시지도 감소하고 CPU의 메모리주소에 없는 Page Table 요청이 매우 낮아진다. GPU는 할당받은 메인메모리의 데이터 주소에서 GPU L2 캐시 메모리로 복사하고 개별 코어에 맞게 처리한 최종결과를 CPU에 통보한다. CPU는 L2, L1 캐시 메모리에서 수행데이터를 제거하여 TBL miss를 최소화하게 된다. 앞서 기능을, 본 연구는 작업관리자(Job Manager), 리매퍼(Re-mapper), 프리페처(Re-fetcher)로 구현하고 그림 2와 같은 융합알고리즘(Fusion Architecture)을 제안하였다.

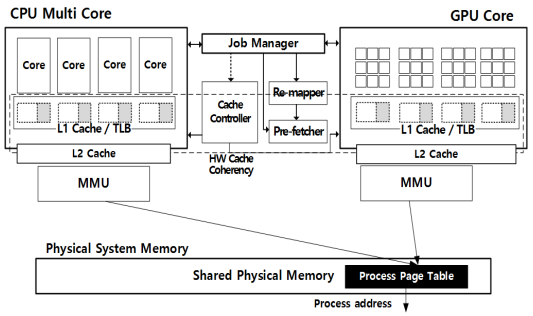


그림 2. 제안된 융합 알고리즘 구조
Fig. 2 Proposed fusion architecture

작업관리자는 CPU가 GPU에 작업을 위임하고 필요한 데이터 범위와 위치를 정의하면 GPU는 처리할 메모리 맵을 연속된 페이지에 할당(allocation)을 수행한다. 이때 CPU와 GPU의 서로 다른 메모리 맵으로 인한 캐시 일관성(Cache Coherency)과 메인 메모리의 쓰고 ·읽기 위해서는 지속적인 GPU과 CPU간 양방향에 메모리 맵의 관리가 이루어져야 하는데, 본 연구에 제안된 리매퍼, 프리페처는 GPU와 CPU의 접근 주소 정보를 호환성 있게 변환해주는 역할을 한다. 이를 통해 GPU의 접근빈도와 크기와 큰 영향 없이 처리할 수 있고 가상메모리 구조로 인한 오버헤드도 줄일 뿐만 아니라 메인메모리에 GPU가 직접 접근하여 빈번한 메모리 복사를 크게 줄일 수 있다.

III. 제안된 융합 프로세서 구조

3장은 제안된 퓨전아키텍처의 작업관리자와 주소 리매퍼 그리고 프리페처의 상세한 동작 설명하였다.

3.1 작업관리자(Job Manager)

그림 3은 작업관리자의 처리과정을 A), B), C), D)의 4단계로 나타내었다.

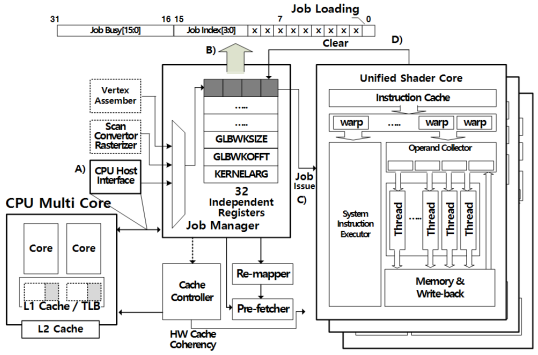


그림 3. 제안된 작업관리자 처리 과정

Fig. 3 Processing of job manager in proposed fusion architecture

A)는 CPU 호스트 인터페이스와 같이 Bus, Bridge를 통해 GPU를 구동할 수 있고, 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 어플리케이션의 L2 Memory Access Latency의 평균을 내어 범위(scalability)를 산출하고 할당될 Shader Core의 개수를 정의한다. Shader Core 개수는 연산속도에도 영향을 주지만 L1 Cache와 L2 Cache 간의 캐시 미스(Cache miss)에 큰 영향을 주기 때문에 최적화하는 과정도 중요하다. C) GPU의 각 코어 작업 할당을 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.

3.2 리매퍼와 프리페처

리매퍼와 프리페처는 앞서 작업관리자를 통해 같이 가상 메모리가 아닌 물리적인 시스템 메모리에 접근하기 때문에 오버헤드를 갖는다. 하지만 오버헤드가 4 Cycles 이내기 때문에 높은 시스템 효율을 갖는다. 또한, CPU가 사용하는 메인 메모리의 페이지 테이블 주소를 GPU에서 접근하기 때문에 캐시 미스도 줄어든다.

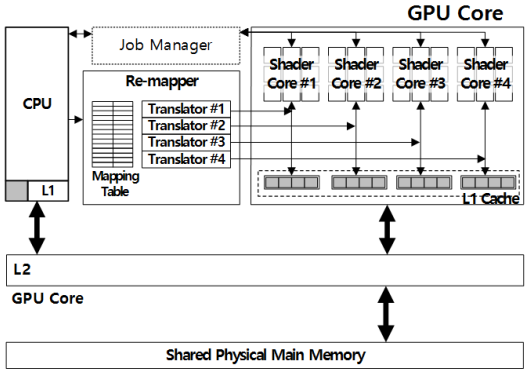


그림 4. GPU에서 융합 프로세서의 리매퍼 처리과정

Fig. 4 Processing of re-mapper processing based on GPU with proposed fusion architecture

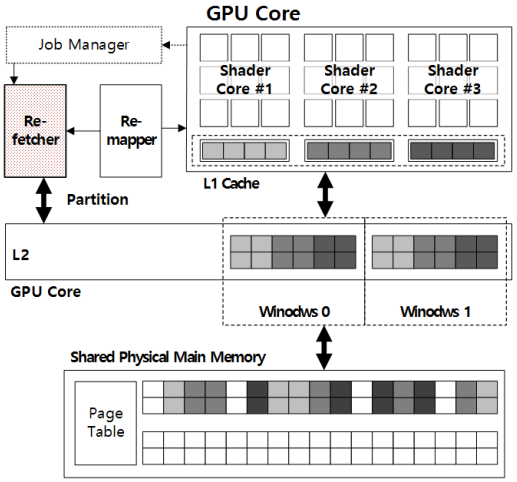


그림 5. 제안된 프리페처의 처리과정

Fig. 5 Processing of proposed re-fetcher schedule in proposed fusion architecture

그림 5와 같이 프리페처는 GPU가 처리할 일을 분할(Partition)하는 역할을 한다. L2의 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.

Ⅳ. QEMU를 통한 성능비교

4.1 QEMU에서 제안된 기본구조

본 아키텍처의 동작의 동작 검증과 응용프로그램을 위해 x86 PC 기반의 하드웨어 가상화 에뮬레이터인 QEMU에서 제안된 융합구조를 OpenCL, Open MP, CUDA 등의 플랫폼과 비교를 통해 검증하였다.

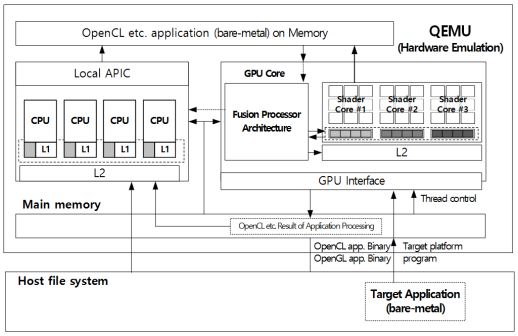


그림 6. QEMU기반에 융합구조 시뮬레이터

Fig. 6 Fusion architecture simulator based on QEMU

그림 6과 같이 QEMU에서 제공하는 기존의 CPU 시스템 버스-메모리 하드웨어 플랫폼에 GPU 코어 에뮬레이션 스레드와 제안된 Fusion Processor Architecture와 베어메탈(bare-metal)을 위한 OpenCL, Open MP 등에 커널 바이너리와 매개변수 데이터를 공급하는 인터페이스 장치를 추가하였다.

QEMU는 동작과 함께 GPU 코어 스레드를 초기화하고 인터페이스 장치의 레지스터 입·출력에 따라 QEMU 호스트와 타깃 플랫폼 사이에 메인메모리의 읽기·쓰기 작업을 중계하거나 호스트파일 시스템에 저장된 OpenCL, GL, MP 바이너리를 타깃 플랫폼의 메모리에 적재한다. 이를 통해 호스트와 QEMU 구현 환경이 서로 분리되어 데이터교환과 QEMU 안에 개발된 플랫폼 동작의 비교평가를 할 수 있다. 외부에서 응용프로그램에 맞춰 제작된 입력 바이너리 데이터를 제한된 구조에 따라 GPU, CPU간의 공유 메인 메모리 영역으로 전송할 수 있게 된다. 그림 6은 구조는 SoC를 위한 FPGA에 구현의 편리함뿐만 아니라 개발된 알고리즘의 타당성 검증에 유리하다. 제안된 영역을 다른 CUDA, HSA의 알고리즘으로 변경할 경우 동일한 자원에서 개별 알고리즘과 OpenCL, OpenMP 등을 성능비교 할 수 있다. 그림 7은 QEMU에서 CUDA, OpenCL, OpenMP. 융합알고리즘을 비교 평가하기 위한 에뮬레이터와 연동한 UI인 융합 구조 분석기(Fusion Architecture Analyzer)를 개발하고 개발 평가항목으로 벡터의 곱셈과 덧셈, 이미지 압축, 스테레오 이미지 등의 성능을 비교하였다.

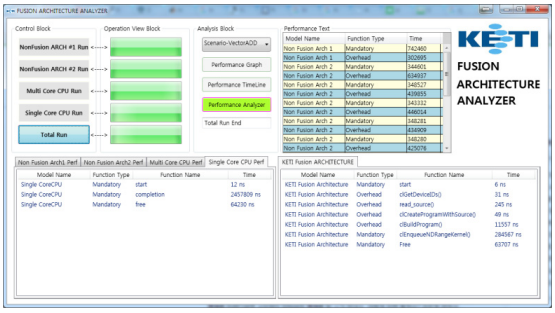


그림 7. QEMU 기반에 융합 구조 분석기

Fig. 7 A developed FAA(: Fusion Architecture Analyzer) based on QEMU

4.2 QEMU에서 성능비교

표 1. QEMU에서 특징적 기능

Table 1. The specification on QEMU

	System Information
CPU-RAM	i7-4790k / 32GB 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

NVIDIA의 모바일 코어인 Tegra X1와 직접적인 비교평가를 하면 좋겠지만, 대부분 제품이 태블릿으로 QEMU에서 비교할 수 없어 X1와 비슷한 Quardo K620로 비교 평가하였다. 표 1과 같이 X1과 차이점은 CUDA core가 384개로 높고 GPU 대역폭 또한 25.6GB/s 반면 벤치마크 GPU는 29.0GB/s로 10~15% 높은 성능을 갖는다. 그림 8과 9의 비교 그래프에서 y축의 Proposed는 본 본문에서 제시한 융합 구조에서 수행한 OpenCL 결과이고 두 번째는 멀티코어에서 OpenMP의 성능, 세 번째는 융합 알고리즘이 적용되지 않은 OpenCL 알고리즘이며, 4번째는 CUDA 성능 비교 알고리즘이다. x 축의 단위는 벤치를 위한 응용 이미지의 처리 성능지표로 ms의 단위를 지닌다.

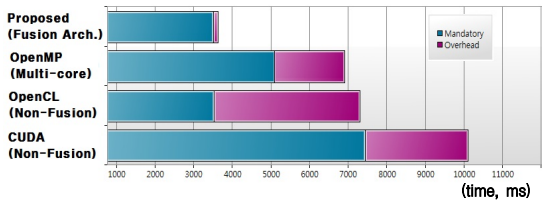


그림 8. 벡터 덧셈 성능 비교

Fig. 8 Vector addition benchmark

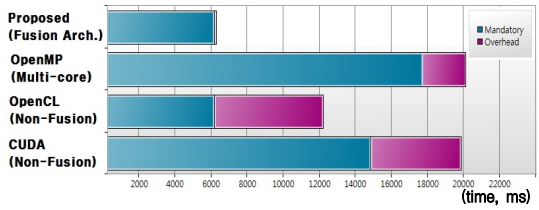


그림 9. 벡터 곱셈 성능 비교

Fig. 9 Vector multiplication benchmark

그림 8, 9에서 Mandatory 영역은 GPU-CPU의 연산영역이고, 빨간색 영역은 메인메모리와 L2 메모리의 복사, 캐시 미스와 같은 핸들링 오버헤드 영역이다. 그림에서 같이 제안된 알고리즘은 GPU-CPU 연동에서 메인메모리에서 직접 읽고 쓰기 때문에 오버헤드가 매우 작지만 Open, CUDA, OpenMP의 경우 가상메모리 영역으로 인한 공유메모리의 GPU와 CPU에 복사가 빈번하고 전체 오버헤드에 12~30%를 차지하였다. CUDA와 OpenMP의 선행연구와 같이 데이터 크기와 메모리 사이즈에 대한 오버헤드가 증가하였다[7-8]. 벡터의 곱셈과 덧셈에서 OpenMP에 비해 본 제안 알고리즘의 메모리 오버헤드가 낮은 것을 알 수 있다. 또한 그림 9의 OpenMP의 처리속도는 벡터 곱셈의 최적화 문제로 OpenCL에 비해서 낮게 나오는 것으로 예상한다. 여기서 특이한 것은 벡터의 곱셈 연산 성능이 덧셈보다 상대적인 오버헤드가 적은 것을 알 수 있다. 이는 GPU 코어가 곱셈에 최적화되어 있기 때문이다.

그림 10, 11은 이미지 압축 성능과 이미지를 스테레오 이미지로 성능 비교를 하였다. 이미지압축 성능에서 전체적인 처리속도는 CUDA나 OpenCL, OpenMP에 비해서 빠른 것으로 나타났지만 오버헤드가 발생되는 것으로 나타났다. 이미지 압축에서 메인메모리의 접근에 따른 메모리 복사가 발생하였다. 스테레오 이미지 성능비교는 OpenCL를 제외하고 제안된 알고리즘이 처리시간이나 오버헤드 측면에서 가장 우월한 것으로 나타났다.

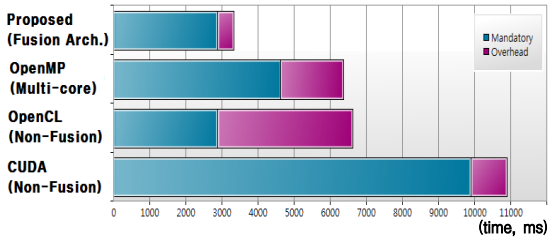


그림 10. 이미지 압축 성능 비교

Fig. 10 Image reverse benchmark

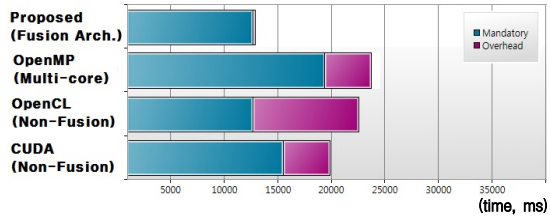


그림 11. 스테레오 이미지 필터 성능 비교

Fig. 11 Comparison of the performance of stereo image filter benchmark

V. 결 론

본 연구는 융합 알고리즘으로 이기종 시스템에서 메모리 접근 기법의 향상 방안으로 융합알고리즘을 제안하였다. HW로 처리되는 CUDA에 비교에서도 115%부터 198%까지 분야별로 높은 효율을 보여주었다. 하지만 이미지 압축과 같은 비교에서는 융합 알고리즘 또한 오버헤드가 많이 증가했으며, 4K H.265, 4K VP9과 같은 고해상도 영상에 처리를 위한 메인메모리 할당 방안을 최적화할 필요가 있다. 점차 발전하는 다중 계층 멀티코어 형태로 발전하는 모바일에서 CPU-GPU간의 계층적 캐시와 TSV기반 메인메모리의 효율적인 공유방안을 연구할 필요가 있다. 현재는 QEMU 없이 아키텍처 간 비교평가를 위한 FAA 기반에 시뮬레이터를 개발하고 있으며, 이와 함께 본 융합 알고리즘을 국내 멀티미디어 셋톱박스와 블랙박스용 반도체를 공급하는 기업에 기술이전 중에 있다.

감사의 글

본 논문은 산업통상자원부 산업융합원천기술개발사업으로 지원된 연구결과입니다. [10041664, 멀티 Shader GPU 통합형 멀티 코어 퓨전 프로세서 원천 기술 개발]

References

[1] J. Power, A. Basu, J. Gu, S. Puthoor, B. M. Beckmann, M. Dill, and D. Aood, “Heterogeneous system coherence for integrated CPU-GPU systems,” *Proceedings of the 46th Annual IEEE/ACM Int. Symposium on Microarchitecture. ACM*, California, USA, Dec. 2013. pp. 457-467.

[2] C. Balkesen, J. Teubner, G. Alonso, and M. T. Ozsu, “Main-memory hash joins on multi-core CPUs: Tuning to the underlying hardware,” *Data Engineering (ICDE), 2013 IEEE 29th Int. Conf. on. IEEE,* Brisbane, Australia, April 2013. pp. 362-373.

[3] B. Pichai, L. Hsu, and A. Bhattacharjee. “Architectural support for address translation on gpus: Designing memory management units for cpu/gpus with unified address spaces,” *ACM Special Interest Group on Programming Languages(SIGPLAN) Notices,* vol. 49 no.4, 2014, pp 743-758.

[4] G. Kim, M. Lee, J. Jeong, and J. Kim, “Multi-GPU system design with memory networks,” *Proceedings of the 47th Annual IEEE/ACM Int. Symposium on Microarchitecture. IEEE Computer Society*, Cambridge, United Kingdom, Dec. 2014. pp. 484-495

[5] J. Jeffers and J. Reinders, *High Performance Parallelism Pearls Volume Two: Multicore and Many-core Programming Approaches,* Waltham: Morgan Kaufmann, 2015.

[6] B. Hechtman, *A.* Blake, and J. Daniel, “Evaluating cache coherent shared virtual memory for heterogeneous multicore chips,” *Performance Analysis of Systems and Software (ISPASS), 2013 IEEE Int. Symposium on. IEEE*, Texas, USA, April. 2013. pp. 118-119.

[7] S. Potluri, H. Wang, D. Bureddy, A. Singh, C. Rosales, and D. Panda, “Optimizing MPI communication on multi-GPU systems using CUDA inter-process communication,” *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International. IEEE,* Shanghai, China, May 2012. pp. 1848-1857.

[8] S. Lin, Y. Liao, and Y. Hsu, “A Reliable and Secure GPU-Assisted File System,” *Algorithms and Architectures for Parallel Processing. Springer Int. Publishing,* vol. 8630, 2014, pp. 71-84.

[9] I. Singh, A. Shriraman, W. Fung, M. O’Connor, and T. Aamodt, “Cache coherence for GPU architectures,” *on High Performance Computer Architecture (HPCA), 19th International Symposium on*, 2013, pp. 578-590.

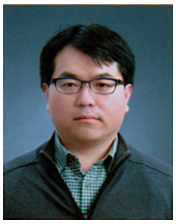
[10] S. Kim and Y. Choi, “Analysis of Human Activity Using Motion Vector and GPU,” *J. of the Korea Institute of Electronic Communication Sciences,* vol. 9, no. 10, 2014, pp. 1095-1102.

[11] J. Park, ”Comparison Speed of Pedestrian Detection with Parallel Processing Graphic Processor and General Purpose Processor,“ *J. of the Korea Institute of Electronic Communication Sciences,* vol. 10, no. 2, 2015, pp. 239-246.

[12] S. Lee and W. Jeong, “Design of the Entropy

Processor using the Memory Stream Allocation for the Image Processing," *J. of the Korea Institute of Electronic Communication Sciences,* vol. 7, no. 5, 2012, pp. 1017-1026.

저자 소개



박현문(Hyun-Moon Park)

2006년 국민대학교 전자통신학과 졸업(공학석사)
2010년 국민대학교 BIT학과 졸업(이학박사)
2010년 10월~2013년 12월 한국전자통신연구원
2014년 1월~현재 전자통신연구원 융합시스템연구본부 선임연구원
※ 관심분야 : IoT시스템, 임베디드시스템

권진산(Jin-San Kwon)

2011년 고려대학교 컴퓨터정보학과 졸업(이학사)
2013년 고려대학교 컴퓨터정보학과 졸업(이학석사)
2013년~현재 전자통신연구원 융합시스템연구본부 전임연구원
※ 관심분야 : 임베디드·웨어러블 시스템 SW

황태호(Tae-Ho Hwang)

1997년 한국외국어대학교 전자공학과(공학석사)
2013년 한국외국어대학교 컴퓨터공학과(공학박사)
2000년~현재 전자부품연구원 융합시스템연구본부 책임연구원
※ 관심분야 : RTOS, WPAN/WBAN, 임베디드 시스템 S/W

김동순(Dong-Sun Kim)

1997년 인하대학교 전자재료공학과 (공학석사)
2013년 인하대학교 전자재료공학과 미디어시스템(공학박사)
1999년~현재 전자부품연구원 융합시스템연구본부 책임연구원
※ 관심분야 : 임베디드 하드웨어, 멀티미디어 SoC Design