ISSN: 1226-7244 (Print)
ISSN: 2288-243X (Online)
논문번호 14-03-14

j.inst.Korean.electr.electron.eng.Vol.18,No.3,392～397,September 2014
http://dx.doi.org/10.7471/ikeee.2014.18.3.392

Memory Latency Penalty를 개선한 SIMT 기반 Stream Processor의 Memory Operation System Architecture 설계

An Implementation of a Memory Operation System Architecture for Memory Latency Penalty Reduction in SIMT Based Stream Processor

이 광 엽*

Kwang-Yeob Lee*

Abstract

In this paper, we propose a memory operation system architecture for memory latency penalty reduction in SIMT architecture based stream processor. The proposed architecture applied non-blocking cache architecture to reduce cache miss penalty generated by blocking cache architecture. We verified that the proposed memory operation architecture improve the performance of the stream processor by comparing processing performances of various algorithms. We measured the performance improvement rate that was improved in accordance with the ratio of memory instruction in each algorithm. As a result, we confirmed that the performance of stream processor improves up to minimum 8.2% and maximum 46.5%.

요 약

본 논문은 Memory Latency Penalty를 개선한 SIMT Architecture 기반 Stream Processor의 Memory Operation System Architecture를 제안한다. 제안하는 구조는 Non-Blocking Cache Architecture를 적용하여 기존의 Blocking Cache Architecture에서 발생하는 Cache Miss Penalty를 개선하였고 다양한 알고리즘의 처리속도를 비교하여 제안하는 Memory Operation System Architecture를 적용한 Stream Processor의 성능 향상을 검증하였다. 실험은 각 알고리즘의 Memory 명령어의 비율에 따라 향상된 성능을 측정하여 Stream Processor의 성능이 최소 8.2%에서 최대 46.5%까지 향상됨을 확인하였다.

Key words : Memory Latency, Non-Blocking Cache, Stream Processor, Cache Memory, SIMT

Ⅰ. 서론

최근 임베디드 기기들의 스마트화를 통해 다양한 고성능의 어플리케이션들을 지원함으로써 사용자의 요구를 만족시켜주고 있다.

이러한 고성능의 어플리케이션을 스마트 기기에서 원활히 동작시키기 위해 어플리케이션의 구조를 최적화하여 연산량을 최소화하는 방안과 하드웨어적인 측면에서 스마트 기기에 다양한 Co-Processor를 탑재함으로써 스마트 기기의 처리 속도를 향상시키는 방안을 함께 적용하고 있다.

스마트 기기의 여러 Co-Processor 중 하나인 SIMT(Single Instruction Multiple Thread) Architecture 기반 Stream Processor[1][2]는 수백, 수천 개 이상의 쓰레드를 통해 많은 연산량을 요구하는

* Dept. of Computer Engineering, Seokyeong University, kylee@skuniv.ac.kr 02-940-7745
※ Acknowledgment
This Research was supported by Seokyeong University in 2012.
Manuscript received Sep. 2, 2014; revised Sep. 22, 2014 ; accepted Sep. 23. 2014

다양한 어플리케이션을 병렬처리를 통해 처리함으로써 중앙 프로세서의 부담을 줄여주고 어플리케이션의 처리속도를 향상시키는 Co-Processor로 현재 활발한 연구가 진행되고 있다.

SIMT Architecture 기반 Stream Processor의 성능은 수백 사이클이 소모되는 외부 메모리로의 접근을 수행하는 Memory Operation System Architecture의 영향을 크게 받기 때문에 Stream Processor의 높은 성능을 기대하기 위해서는 Memory Operation System의 구조를 효율적으로 설계하는 것이 매우 중요한 부분이라 할 수 있다.

본 논문은 기존의 Memory Operation System Architecture를 개선하여 외부 메모리로 접근할 경우 발생하는 Latency[3]를 개선한 Memory Operation System Architecture의 설계를 제안한다.

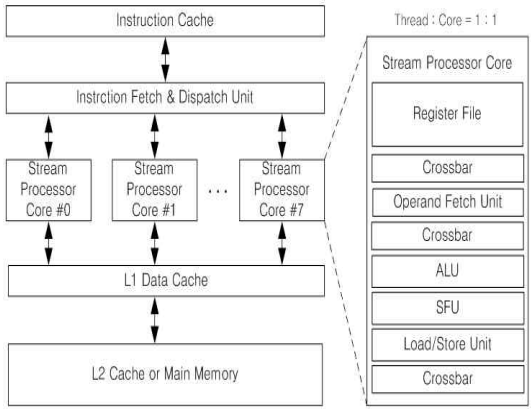

Fig. 1. SIMT based Stream Proceesr Architecture
그림 1. SIMT 기반 스트림 프로세서의 구조

II. 본론

1. SIMT Architecture 기반 Stream Processor

가. Multiple Thread & Load/Store Units

그림 1은 SIMT Architecture 기반 Stream Processor의 구조이다. SIMT Architecture에서 다 수의 Thread는 자신의 흐름과 데이터를 가지고 동일한 명령어를 수행하는 구조이며 각 Thread는 명령어를 전달받을 때 자신의 활성화 여부도 함께 전달받는다. 이 때, 활성화된 Thread를 Active Thread[4]라 한다.

그림1의 구조는 총 8개의 Thread를 가지고 있으므로 모든 Thread가 Active Thread일 경우까지 고려하여 모든 Thread가 각자의 데이터를 처리할 수 있도록 총 8개의 Load/Store Unit이 필요하다. 즉, Stream Processor의 Core는 Stream Processor가 가진 전체 Thread의 수만큼 보유하도록 설계해야 한다.

나. Memory Operation Interface

Stream Processor의 Core 내부의 Load/Store Unit의 Memory 접근 요청은 Core 외부에 존재하는 L1 Data Cache에서 처리하기 때문에 각 Load/Store Unit의 Memory 접근 요청 신호와 요청 주소번지는 Core 외부로 출력되어 L1 Data Cache에 전달된다. 이 때, 각 구조의 Stream Processor는 Load/Store Unit이 연속으로 Memory 접근 요청을 보내더라도 이를 모두 안정적으로 처리할 수 있도록 Memory Operation System Architecture를 설계해야 한다.

2. SIMT Architecture 기반 Stream Processor의 Memory Operation System Architecture

가. Multi-banked Memory Cache Architecture

L1 Data Cache는 Memory Operation Interface를 통해 통합된 Thread들의 메모리 접근 요청을 받아 Cache Unit 내에서 다시 각 Thread 별로 메모리 접근 요청을 나누어 처리한 뒤, 그 결과를 각 thread와 매핑되어 있는 Stream Processor Core에 전달한다. 이 때, Cache Memory 구현에 사용되는 SRAM의 특성상 동시에 한 번의 읽기와 한 번의 쓰기만 가능하므로 모든 Thread의 메모리 접근 요청을 동시에 처리하기 위해서는 최소 Thread 개수 이상의 Cache Memory가 필요하게 된다. 그러므로 SIMT Architecture 기반 Stream Processor의 L1 Data Cache는 동시에 모든 Thread의 메모리 접근 요청을 처리할 수 있도록 Thread 개수 이상의 Cache Memory를 가진 Multi-banked Memory Cache Architecture로 설계되어진다.

나. Blocking Cache Architecture

그림 2는 Blocking Cache의 동작 구조를 나타낸 것이다. Blocking Cache Architecture는 전통적인 Cache Architecture로써 Cache Miss가 발생하면 발생한 Miss를 처리할 때 까지 Cache의 동작을 멈췄다가 Miss가 모두 처리되었을 때 다시 동작을 하는 순차적인 처리 흐름을 가진다.

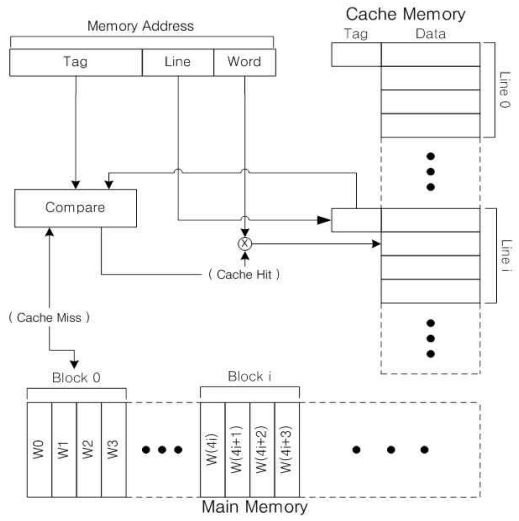

Fig. 2. Blocking Cache Architecture
그림 2. Blocking 캐시 구조

다. SIMT 기반 Stream Processor에서 Memory Operation System의 High level Architecture

그림 3은 기존 SIMT Architecture 기반 Stream Processor의 Memory Operation System Architecture의 High level Architecture[5][6] 블록도이다. 각 Core의 Load/Store Unit의 Memory 접근 요청이 Load/Store Unit과 L1 Data Cache의 데이터 전송을 안정적으로 하기 위해 설계된 Memory Operation Interface의 Queue를 거쳐 L1 Data Cache의 Data Request and Response Crossbar와 통신하고 캐시 컨트롤 로직에 의해 각 Thread가 접근해야 하는 Cache bank Memory로 접근한다. Blocking Cache의 구조로 설계되었기 때문에 Cache bank Memory로 접근한 결과가 Miss라면 L2 Request and Response Crossbar를 통해 L2 Data Cache 또는 메인 메모리에게 유효한 Data를 요청하며 유효한 Data를 전달받을 때까지 대기한다.

3. 제안하는 SIMT Architecture 기반 Stream Processor의 Memory Operation System Architecture

가. Proposed Non-Blocking Cache Architecture

그림 4는 본 논문에서 제안하는 non-Blocking Cache[7]의 동작 흐름이다. Non-Blocking Cache Architecture는 Cache Miss가 발생할 경우 메인 메모리로 접근하기 위해 소모되는 Memory latency에 의해 발생하는 파이프라인 스톨로 인해 Throughput이 감소하지 않도록 하기 위해 고안된 구조로 Lockup-free Cache라고도 한다.

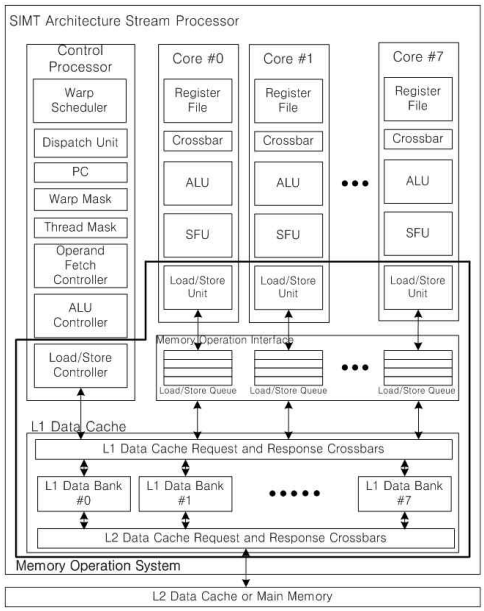

Fig. 3. A High level Block Diagram for Memory Operation System Architecture of Conventional Stream Processor
그림 3. 기존 스트림프로세서 메모리 동작 시스템의 High level 블록도

그림 5와 같이 본 논문에서 제안하는 Non-Blocking Cache의 구조는 Cache Miss가 발생하면 Miss가 발생한 명령어의 정보(접근할 메모리 주소 번지, Warp ID, Thread ID, Load의 경우에는 Load/Store Unit에 전달할 Register index)를 저장하고 있다. Non-Blocking Cache는 발생한 Cache Miss가 해결되기 전에 추가적으로 Cache Miss가 발생할 수 있기 때문에 Miss Handling Unit의 Miss Handling Queue에 Miss가 발생하는 순서대로 Miss가 발생한 명령어의 정보를 저장하고 순서대로 처리한다. 하지만 Cache Miss 발생여부와 관계없이 Miss Handling Queue가 가득 찬 상태가 아니면 Cache는 다음 메모리 접근 요청 명령을 수행한다. Cache Miss가 발생한 명령어의 다음 명령어가 Cache Hit라면 Out-of-Order 방식으로 Cache Hit된 데이터를 먼저 Load/Store Unit에 전달함으로써 Stream Processor의 파이프라인 스톨을 방지해 Throughput을 효과적으로 향상시킬 수 있다.

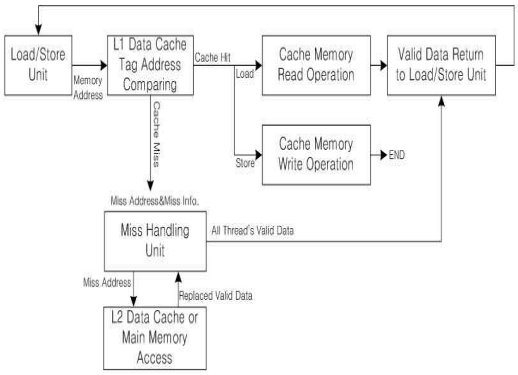

Fig. 4. Non-Blocking Cache Operation Flow
그림 4. Non-Blocking 캐시 동작 흐름

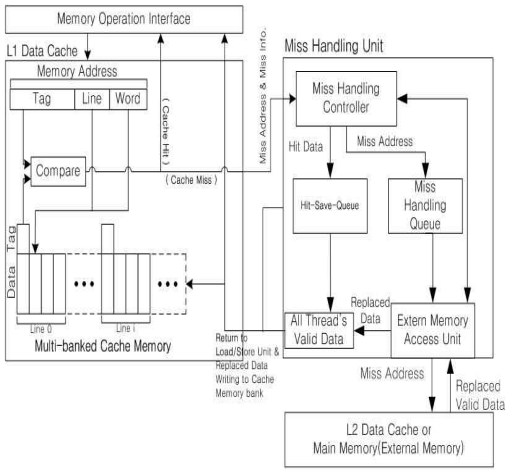

Fig. 5. Proposed Non-Blocking Cache Architecture
그림 5. 제안하는 Non-Blocking 캐시 구조

나. 제안하는 Memory Operation System의 High level Architecture

그림 6은 제안하는 SIMT Architecture 기반 Stream Processor에서 Memory Operation System의 High level 블록도이다. 기존 SIMT Architecture 기반 Stream Processor의 Memory Operation System Architecture와 달리 Non-Blocking Cache Architecture를 적용하여 설계하였다. 제안하는 Non-Blocking Cache는 Cache Miss가 발생할 경우 Miss가 발생한 명령어의 정보와 함께 Cache Hit가 된 Thread의 유효 데이터를 Miss Handling Unit의 Hit Save Queue에 저장해두고 Miss Handling Queue를 통해 Cache Miss로 인한 Miss Data가 유효 데이터로 교체될 때 교체된 데이터를 그림 7과 같이 Hit-Save-Queue의 유효 데이터와 결합하여 모든 Load/Store Unit이 유효데이터를 전달받을 수 있도록 보장하기 때문에 기존의 Load/Store Unit과 L1 Data Cache의 데이터 전송을 지원하기 위해 설계되었던 Memory Operation Interface의 Queue를 제거한 구조로 변경되었다.

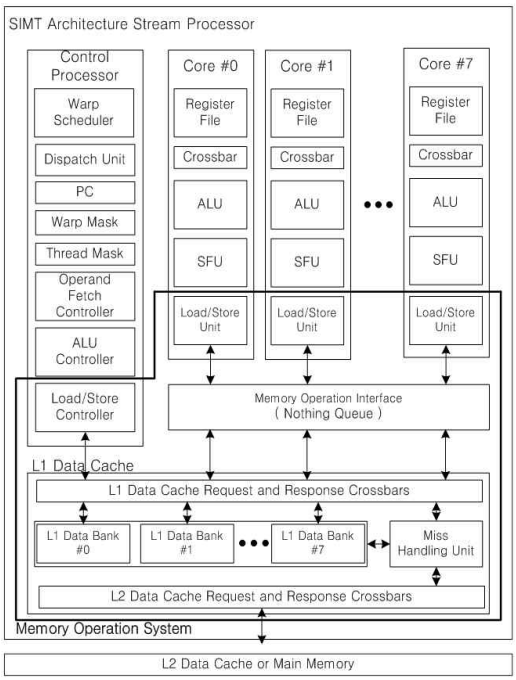

Fig. 6. Proposed Memory Operation System Architecture of Stream Processor
그림 6. 제안하는 스트림 프로세서의 Memory Operation System 구조

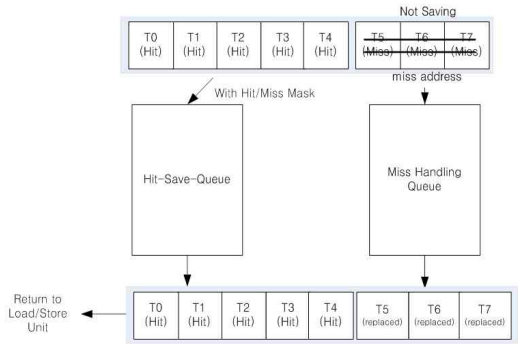

Fig. 7. Data Management of Hit-Save-Queue and Miss Handling Queue
그림 7. Hit-Save-Queue와 Miss Handling Queue의 데이터 관리

III 실험 및 결과

본 논문은 기존의 SIMT Architecture 기반 Processor의 Memory Operation System Architecture에서 발생하는 Cache Miss Penalty를 개선하기 위한 Memory Operation System Architecture를 제안한다. 그림 8은 1000개의 명령어 중 25%가 Memory 명령어로 발행되는 환경에서 기존의 Memory Operation System Architecture의 Cache Miss Ratio에 따른 Cache Miss Penalty Cycle과 제안하는 구조의 Cache Miss penalty Cycle의 비교치를 나타내며 표1은 이에 따라 제안하는 Memory Operation Unit의 향상된 성능을 표기한 것이다. Cache Miss Penalty Cycle은 일반적으로 Bus Interface의 traffic 상황에 따라 변하기 때문에 본 논문에서는 메인 메모리인 DRAM으로 접근할 때 소요되는 Latency를 120 Cycle로 고정하여 Cache Miss Penalty Cycle의 기준으로 사용하였고 Stream Processor의 전체 파이프라인은 20 스테이지로 구성하였다. 제안하는 Memory Operation System Architecture는 Non-Blocking Cache Architecure의 특성으로 인해 Cache Miss가 발생하는 집적도에 따라 명령어 전부를 수행하였을 때 측정되는 Cache Miss Penalty Cycle이 다르므로 최소 Cache Miss Penalty Cycle과 최대 Cache Miss Penalty Cycle, 그리고 평균 Cache Miss Penalty Cycle을 함께 측정하였다.[8]

표 2는 SIMT Architecture 기반 Processor로 수행한 알고리즘을 표기한 것이며 표 3은 기존 SIMT Architecture 기반 Stream Processor와 제안하는 Memory Operation System Architecture를 적용한 SIMT Architecture 기반 Stream Processor가 표 2의 알고리즘을 수행할 때, 각 알고리즘의 Memory 명령어 발행 비율에 따라 처리 성능을 비교하여 Memory Operation 성능 향상을 통한 Stream Processor 전체 성능의 향상률을 나타낸다.

Table 1. Performance improvement of Memory Operation by Cache Miss Ratio
표 1. 캐시 미스 효율에 따른 Memory Operation 성능 향상

	Memory Operation					
Cache Miss Ratio	3%		7%		10%	
performance Improvement (%)	Avg.	Max	Avg.	Max	Avg.	Max
	52.9	99.2	51.0	99.2	50.6	99.2

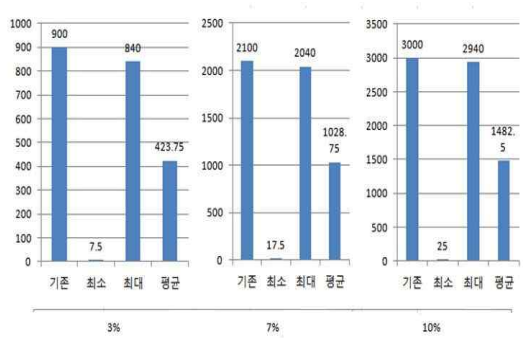

Fig. 8. Cache Miss Penalty Cycles for Continually Memory Instruction Fetching and Cache Miss Ratio
그림 8. 캐시 미스율과 지속적인 메모리 명령어 수행에 따른 캐시 미스 페널티 사이클

Table 2. Algorithms for Performance Measurement
표 2. 성능 측정을 위한 알고리즘

Name	Description
bilat	Bilateral image filtering
dither	Floyd-Steinberg image dithering
kmeans	KMeans clustering
rsort	Radix sort of array of integers
strsearch	Knuth-Morris-Pratt string search
viterbi	Viterbi decoder

Table 3. Performance Improvement of Total Operation by Cache Miss Ratio
표 3. 캐시 미스 효율에 따른 Total Operation 성능 향상

	Total Operation					
Cache Miss Ratio	3%		7%		10%	
Name (Mem inst. Ratio)	performance Improvement(%)					
	Avg.	Max	Avg.	Max	Avg.	Max
bilat(15)	8.6	17.5	8.3	17.5	8.2	17.5
dither(30)	18.9	42.4	18.1	42.4	17.9	42.4
kmeans(29)	18.1	40.4	17.4	40.4	17.2	40.4
rsort(32)	20.4	46.5	19.5	46.5	19.3	46.5
strsearch(15)	8.6	17.5	8.3	17.5	8.2	17.5
viterbi(29)	18.1	40.4	17.4	40.4	17.2	40.4

Ⅳ 결론

본 논문은 기존의 SIMT Architecture 기반 Stream Processor의 Memory Operation System Architecture의 Cache Miss Penalty Cycle을 개선하기 위해 Non-Blocking Cache Architecure를 적용한 Memory Operation System Architecture를 제안하였다.

설계한 L1 Data Cache의 Miss Handling Unit 내부에 Hit-Save-Queue를 두어 Stream Processor Core와 Cache 사이의 데이터 안정성을 보장하였다.

향상된 성능을 검증하기 위해 기존의 Memory Operation System Architecture와의 성능차이를 비교하였고 이를 통해 제안하는 Memory Operation System Architecture의 성능 향상률과 Stream Processor 전체 동작 성능의 향상률을 측정하였다.

그 결과, 다양한 알고리즘 수행에 대한 처리 성능이 기존의 Stream Processor 대비 최소 8.2%에서 최대 46.5%까지 향상되었고 지속적으로 메모리 명령어가 발행되는 환경에서 Cache Miss ratio에 따른 Cache Miss Penalty Cycle은 기존의 Memory Operation System Architecture 대비 최소 2%에서 최대 99.2%까지 감소하여 평균 약 51%의 감소율을 보였다.

References

[1] Sung Su Kim, “Table-based thread reconvergence mechanism on SIMT processor”, The Graduate School of Yonsei University, 2011

[2] Kwang-Yeob Lee, Tae-Ryong Park, "Method of Multi Thread Management based on Shader Instruction for Mobile GPGPU", Journal of IKEEE. Vol.16, No.4, 310~315, December 2012

[3] Jianmin Chen, Xi Tao, Jih-Kwon Peir, “Guided Region-Based GPU Scheduling: Utilizing Multi-thread Parallelism to Hide Memory Latency”, 2013 IEEE 27th International Symposium on, 441-451, 2013

[4] Xiaosong Ma, Gomes, B, Quittek, J.W. “Efficient fine-grain thread migration with active threads”, Parallel Processing Symposium 1998, 410-414, 1998

[5] Wilson W. L. Fung, Ivan Sham, George Yuan, Tor M., "DynamicWarp Formation and Scheduling for Efficient GPU Control Flow", MICRO 2007, 407--420 ,2007

[6] Ji Kim, Christoper Torng, Shreesha Srinath, "Microarchitectural mechanisms to exploit value structure in simt architectures", 40th ACM/IEEE Int'l Symp. on Computer Architecture (ISCA) , 2013

[7] Seungpil Lee, "Design of a non-blocking instruction and data cache controller for SMT microprocessors“, The Graduate School of Yonsei University, 2002

[8] J. A. Stratton et al. parboil, "A Revised Benchmark Suite for Scientific and Commercial Throughput Computing“, Technical report, UIUC, IMPACT-12-01, 2009

BIOGRAPHY

LeeKwangYeob (Life member)

1985. 8 Seogang University, Dept. of Electronics Engineering (BS)

1987. 8 Yonsei University, Dept. of Electronics Engineering(MS)

1994. 2 Yonsei University, Dept. of Electronics Engineering(Ph.D)

1989~1995. 2 Hyundai Electronics Inc., Senior Researcher

1995.3~ Seokyeong Univeristy, Dept. of Computer Engineering, Professor

<Research interests> Microprocessor, Embedded System, Image Processing