누적 버퍼를 활용한 효율적인 Latency Hiding기법*

이민우°, 한탁돈*
°*연세대학교 컴퓨터과학과
e-mail:{hellomw, hantack}@msl.yonsei.ac.kr

An Efficient Latency Hiding method using accumulation buffer

Min-Woo Lee°, Tack-Don Han*
°*Dept. of Computer Science, Yonsei University

● 요 약 ●

현재 cache의 성능 향상을 위한 많은 기법들이 제안되고 있으며, Latency Hiding 기법 역시 cache의 효율적인 사용을 위해 많은 연구가 진행 되어 왔다. write buffer를 사용한 write Latency hiding기법이나 multi threading을 사용한 Latency Hiding 방법 등 여러 기법들이 연구되어 왔으며, 지금도 Latency hiding을 위한 많은 연구들이 지속적으로 진행되고 있다. 본 논문 역시 효율적인 Latency Hiding을 위한 누적 버퍼를 제안한다. 본 논문은 누적 버퍼의 활용도를 조사하여 얼마나 효율적으로 Latency를 은폐했는지, 또 버퍼를 사용함으로써 얻는 다른 이점에 대해 집중적으로 연구하였다.

키워드: 캐시(cache), 지연시간 은폐(Latency Hiding), 누적 버퍼(accumulation buffer)

I. 서론

일반적으로 cache memory는 CPU가 필요로 하는 데이터가 cache memory에 있을 경우 cache hit를 통해서 memory에서 데이터를 가져오고 cache memory에 데이터가 없을 경우 cache miss를 호출하게 된다. cache miss가 발생하는 경우에는 memory로부터 데이터를 접근해 cache로 전송해오는 과정을 거쳐야 하기 때문에 Latency가 발생하게 된다. cache는 CPU의 경우 L1의 데이터를 읽는데 2 cycle 정도, L2의 데이터를 L1으로 읽어오는데 6~10 cycle 정도 필요한데 비해 memory의 데이터를 L2 cache로 옮기는 시간은 30-100cycle정도가 소모되므로 이는 cache의 성능 저하의 주된 원인이 된다. 따라서 연속적인 Hit가 이루어지면 이루어 질수록 cache의 성능이 좋아지는 것이 당연하다. 하지만 일반적으로 입력으로부터 요청되는 데이터가 항상 연속적으로 Hit하기는 어렵다. 그래서 이런 Latency Hiding에 대한 연구가 꾸준히 이루어지고 있는 상황이다.

Latency hiding에 대한 대표적인 기법들로 write- buffer, multi threading 등이 있으며 이들을 기반으로 많은 연구들이 진행되어 왔다. 하지만 꾸준히 증가하는 프로세서와 memory사이의 속도차이를 보안하기 위해서 지금도 많은 연구들이 진행되고 있다.

본 논문에서는 효율적인 Latency Hiding을 제안하고 있다. 일반 하드웨어에서 miss데이터를 버퍼에 저장한 뒤 memory에서 요청한 데이터가 오는 동안 새로운 입력을 받고 처리할 수 있도록 하여 요청한 정보가 도착하는 동안의 Latency를 은폐시킨다. 이로서 cache는 새로운 입력을 받을 수 있게 되고 Latency가 자연스레 은폐될 수 있는 것이다. 본 논문은 버퍼의 활용성과 버퍼 참조로 인해 감소하는 Cache 접근 횟수에 대해 집중적으로 연구하였다.

II. 관련 연구

1. Latency Hiding

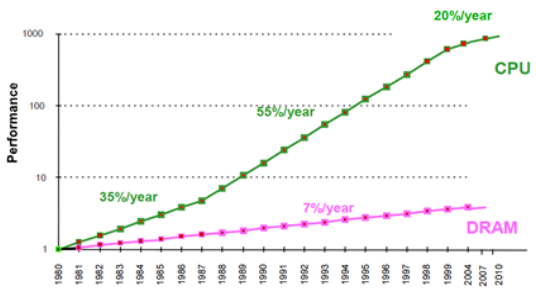

그림1. 프로세서와 memory사이의 속도 차이
Fig1. Speed gap between processor and memory

* 이 논문은 2011년 정부 (교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(한국연구재단-2011-No.2011-0027450)

Cache에서 Latency란 miss가 발생했을 경우 memory에서 데이터를 가져와야 할 때 발생되는 지연시간을 말한다. 이 지연시간 동안 cache는 새로운 입력 정보를 받을 수 없게 되고 miss가 발생한 데이터의 정보가 memory로부터 도착할 때까지 대기 상태가 된다. 이로 인해 Latency가 발생되는데, 이는 속도가 빠른 프로세서와 상대적으로 느린 memory 사이의 속도 차이로부터 발생된다. 현재 프로세서와 memory 사이의 속도 차이는 그림1과 같이 증가하는 추세이며 앞으로도 그럴 전망이다.

따라서 프로세서와 memory 사이의 Latency를 해결하기 위한 여러 방안이 제시 되어왔으며 Latency Hiding 또한 그 해결 방안 중 하나로서 많은 연구들이 진행되어 왔다. Latency Hiding으로 소개된 많은 방법의 대표적인 것들 중에는 write buffer, Multi-threading 등이 있다.

write buffer란 쓰기 요청에 대한 정보를 버퍼에 저장하여 프로세서가 요청이 완료될 때까지 대기하는 것으로부터 자유롭게 함으로써 Latency를 은폐시키는 기법이다. Multi Threading의 경우 긴 Latency 참조가 계산되었을 경우, 한 개 Thread에서 다른 Thread로 switching 하여 miss가 처리되는 동안 다른 유용한 작업을 수행할 수 있게 한 Latency Hiding기법이다. 하지만 switching cost가 memory 접근 penalty보다 크면 효율성이 많이 떨어지는 것이 단점이다.

2. 광선 누적 버퍼(RAB : Ray Accumulation Buffer)

Ray-tracing은 현재 활발히 연구가 진행 중인 그래픽 Render방식이다. Ray-tracing은 scan line기법에 비해 보다 정교한 Rendering이 가능하다는 점에서 많은 연구들이 진행되고 있다. 하지만 Ray- tracing은 정교한 Rendering이 가능한 대신 긴 시간의 Render time이 요구된다. 그런 이유로 아직 Ray-tracing의 Real-time Rendering은 이 분야의 연구 과제로서 여전히 많은 연구가 진행 되고 있다.

뿐만 아니라 Ray-tracing전용 하드웨어 구현 역시 Real-time Ray-tracing을 위한 접근법으로써 많은 연구가 진행 중이다. 광선 누적 버퍼는 이러한 Ray- tracing 전용 하드웨어에서 Latency를 감추기 위해 제안된 Hiding기법이다.

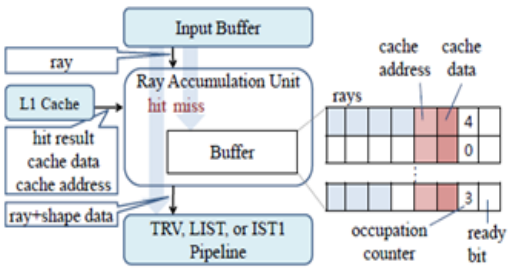

그림2. 광선 누적 버퍼의 구조
Fig2. Architecture of ray accumulation buffer

광선 누적 버퍼의 기본 개념은 miss가 난 Input Ray데이터를 버퍼에 저장해 놓은 다음 새로운 Input Ray를 처리함으로써 miss가 난 Input Ray의 정보가 memory로부터 도착할 동안에도 새로운 Input Ray를 받을 수 있다는 것이다.

물론 coherent한 입력에 대해서는 SIMD방식에서의 Latency hiding기법이 더 효율적일 수 있다. 하지만 난반사와 같은 incoherent한 입력에 대해서는 단일 Thread로 처리 하는 것이 더 빠를 수 있다. 난반사처럼 coherent한 입력이 아닌 incoherent하게 들어오는 입력의 경우 SIMD구조로 처리하게 된다면, 각각의 Thread는 서로 다른 데이터를 요구하게 될 것이다. 만약 하나의 Lane이 miss가 될 경우 다른 모든 Lane들 모두 대기 상태가 될 것이다. 이는 각각의 Ray들을 temporal locality를 부당하게 이용하는 것으로부터 예방하기 위해서이다. RAB버퍼는 이런 incoherent한 입력에 대해 보다 효율적으로 처리하기 위해 개발되었다.

III. 본론

1. 전반적인 구조

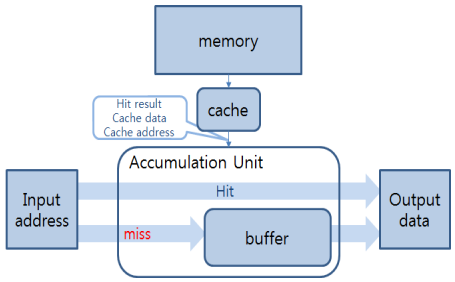

그림3. 누적 버퍼의 전반적인 구조
Fig3. Overall architecture of accumulation buffer

그림 3은 누적버퍼의 전체 구조를 설명하고 있다. 버퍼는 입력과 cache 그리고 출력데이터 사이의 중간다리 역할을 하고, 또한 memory로부터 데이터를 가져오기도 한다. 입력 주소는 hit/miss 여부에 따라 바로 출력 데이터로 내보내던지 memory에 요청해서 해당 정보를 가져 올 것인지 판단한다. 만약 어떤 입력 주소가 miss가 발생 된다면 해당 데이터가 memory에서부터 오는 동안 miss가 발생된 입력 Tag를 버퍼에 저장하고 데이터가 도착하게 되면 해당 데이터를 출력하게 된다.

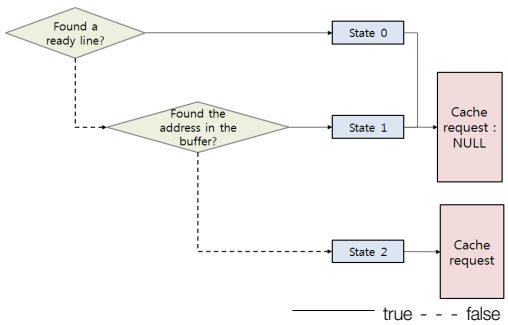

그림 4. cache 요청 이전 작업 흐름도
Fig 4. Work flow before cache request

그림4와 같이 누적 버퍼는 Latency동안 miss가 발생한 주소를 저장하고 다음 입력 주소를 받을 수 있게 한다. miss가 발생되어 버퍼에 저장되어 있던 주소는 해당 데이터가 memory로부터 도착하게 되면 Ready bit에 1을 넣어 출력 데이터를 내보낸다. 만약 준비된 line이 없다면 버퍼는 입력을 받고 만약 해당 주소의 데이터가 버퍼에 남아 있다면 데이터를 출력 데이터에 저장한 뒤 출력 데이터를 내보낸다. 주소는 같지만 데이터가 남아 있지 않다면 버퍼에 저장한 뒤 다음 입력 주소를 받는다.

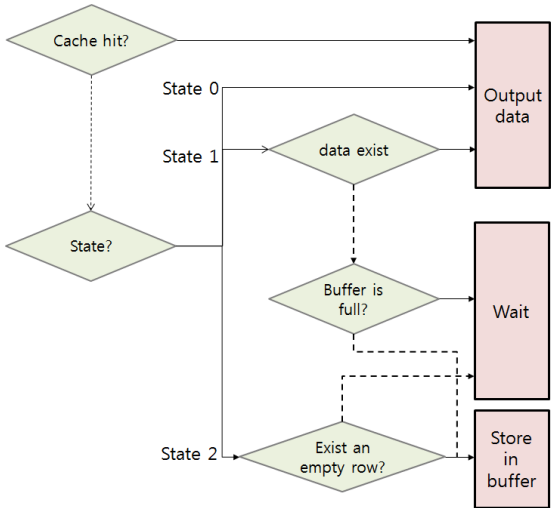

그림5. cache 요청 후 작업 흐름도
Fig5. Work flow after cache request

그림 5에서처럼 버퍼에 해당 주소 line이 없다면, cache에 요청을 한 뒤 hit/miss결과를 받는다. 만약 hit라면 출력데이터를 바로 내보내고, miss라면 memory에 데이터를 요청한 뒤 버퍼에 해당 주소를 저장한 뒤 다음 입력 주소를 받는다.

2. 버퍼 구조

버퍼는 n * m사이즈로 구성이 되며 한 행에 n개의 동일한 Tag와 주소의 데이터가 저장이 되며, 총 m개의 서로 다른 Tag의 주소가 저장된다.

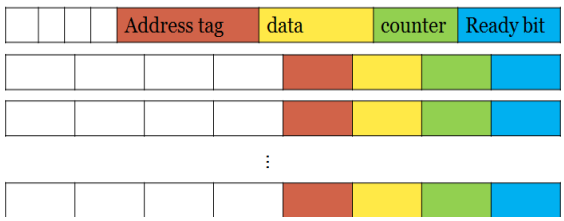

그림5. 누적 버퍼의 내부 구조
Fig5. inner organization of accumulation buffer

버퍼 라인의 구조는 주소 Tag와 데이터 그리고 주소 카운터, Ready bit로 구성된다. 주소 tag는 22bit로 주소의 태그를 저장하며 버퍼에 저장되는 데이터는 32byte로 가정하여 실험하였다. counter는 현재 라인에 저장되어 있는 동일한 tag를 가진 주소의 개수를 표시하고 있다. Ready Bit의 경우 버퍼에 저장된 cache miss되었던 주소 라인에 memory에 요청했던 데이터가 도착 했는가 안했는가를 구분하기 위한 flag bit이다.

표1. 누적 버퍼 사용 여부에 따른 cache 접근 횟수
Table1. Cache access count according to Accumulation Buffer use

	Data Type	cache hit	cache miss	buffer hit	buffer miss	buffer success	buffer fail	Total address
Trace 1	Read	5,194,160	30,272	4,084,584	23,918	1,115,159	771	5,224,432
Trace 2	Read	1,495,984	27,307	1,354,331	15,880	152,509	571	1,523,291
	Write	1,676,507	20,555	1,211,798	15,787	469,027	450	1,697,062
Trace 3	Read	780,321	14,377	696,389	8,229	89,806	274	794,698
	Write	794,009	9,728	571,083	7,349	224,984	231	803,737

3. 실험 결과 및 실험 환경

본 논문에서는 표1과 같이 3개의 Trace 파일을 사용하여 성능 조사를 실시하였다. 첫 번째 파일은 실제 RAB에서 사용하는 메모리 주소를 사용한 파일이고, 두 번째와 세 번째는 SPEC2006 파일을 사용하였다. 하지만 현재 Instruction에서의 버퍼 성능에 대한 의심이 남아 실험은 Read와 Write의 두 부분에 대한 성능 조사만 이루어졌다. cache는 현재 2 associative cache로서 block size는 32비트이며 세트 수는 1024개로 실험하였다. 또한 cache의 성능 향상을 위해 Victim cache를 사용하였다.

표2. 버퍼 사용 빈도 및 미스 감소율
Table2. Buffer use frequency and miss reduce rate

	Data Type	Buffer_success/ Total_address	Buffer miss/ cache miss
Trace1	Read	21.34%	79.01%
Trace2	Read	10.01%	58.15%
	Write	27.63%	76.80%
Trace3	Read	11.30%	57.23%
	Write	27.99%	75.54%

표2는 버퍼의 활용도와 miss 감소율을 조사한 결과이다. 위 결과와 같이 최소 10%에서 최대 27.6%의 입력을 버퍼 내부에서 처리한다. 이로 인해서 최대 cache miss의 58%까지 미스 횟수가 줄었다. Trace2와 Trace3의 경우 버퍼 활용도는 write가 높았지만 미스 감소율에 있어서는 Read 에서 낮은 성능을 보였다. Trace2와 Trace3 파일 보다 Read로만 구성되어 있는 Trace1파일이 높은 활용도를 보였다. 이는 Trace1파일이 다른 파일들 에 비해 이웃한 주소 참조가 많았고 버퍼와 cache에 Read만 저장하기 때문에 버퍼의 활용도가 높게 조사되었다. 만약 다른 Trace 파일들에 대해 Read cache와 write cache를 따로 둔다면 Trace1과 비슷한 성능을 낼 것이다.

IV. 결론

누적 버퍼의 사용으로 인해 cache miss의 상당량이 버퍼에 누적되었다. 따라서 많은 Latency가 버퍼의 사용으로 인해 은폐됨으로써 효율적인 Latency Hiding기법을 제시하였다. 뿐만 아니라 같은 memory를 참조하는 주소들끼리 함께 처리함으로 써 그에 따른 cache 접근 횟수의 감소 또한 실험 결과에서 나타났다. 본 논문은 단일 Thread 처리가 필요한 프로세서에서의 cache Latency Hiding 기법으로 incoherent한 주소를 참조하는 cache에 적용한다면 높은 효율을 기대할 수 있을 것이다.

참고문헌

[1] Jae-Ho Nah et al “T&I Engine: Traversal and Intersection Engine for Hardware Accelerated Ray Tracing” ACM Transactions on Graphics, Volume 30 Issue 6, December 2011

[2] Ando Ki “Memory Latency Hiding Techniques”, Electronics and Telecommunication Trand, Volume 13 issue 3, June 1988

[3] ALAN JAY SMITH ”Cache Memories” Computing Surveys, Vol. 14, No. 3, September 1982

[4] WHITTED, T. An improved illumination model for shaded display. Communications of the ACM 23, 6, 343.349. 1980.

[5] AILA, T., AND LAINE, S. “Understanding the efficiency of ray traversal on GPUs.” In HPG '09: Proceedings of the Conference on High Performance Graphics, 145–149.2009

[6] Norman P. Jouppi “Improving Direct-Mapped Cache Performance by the Addition of a Small Fully- Associative Cache and Prefetch Buffers” Computer Architecture, 364 - 373. 1990