논문 2009-46SD-11-13

라이트 백 캐쉬를 위한 빠른 라이트 백 기법

(The Early Write Back Scheme For Write-Back Cache)

정 영 진*, 이 길 환**, 이 용 석***

(YoungJin Chung, Kilwhan Lee, and Yongsurk Lee)

요 약

일반적으로 3차원 그래픽 깊이 캐쉬와 픽셀 캐쉬는 메모리 대역폭의 효율적인 사용을 위하여 라이트 백(write-back) 캐쉬로 설계된다. 또한 3차원 그래픽 특성상 캐쉬 읽기 접근을 시도한 주소에 대한 캐쉬 쓰기 접근 혹은 읽기 접근이 발생하지 않고 캐쉬 쓰기 접근만 발생하는 경우가 많다. 캐쉬 메모리의 모든 블록이 사용되고 있는 상태에서 캐쉬 접근 실패가 발생하면 캐쉬 메모리 한 블록이 교체 알고리즘(replacement algorithm)에 의하여 한 블록을 라이트 백 동작을 실행하고 그 블록에 다른 데이터를 저장한다. 이러한 캐쉬 접근 실패 발생은 방출되는 캐쉬 메모리 한 블록의 데이터를 저장하기 위한 외부 메모리 쓰기 접근과 캐쉬 접근 실패를 처리하기 위한 외부 메모리 접근을 동시에 발생시킨다. 따라서 연속적인 캐쉬 접근 실패가 발생하는 경우 다량의 메모리 읽기와 쓰기 접근이 동시에 발생되어 메모리 병목현상을 유발시키고 이는 결국 메모리 접근 소요시간을 길어지게 한다. 이와 같이 연속적인 캐쉬 접근 실패는 캐쉬를 사용하는 프로세서나 IP의 성능 저하와 전력소비 증가를 유발한다. 본 논문에서는 캐쉬 사용 시 발생하는 메모리 병목현상을 최소화하기 위하여 빠른 라이트 백이라는 새로운 방법을 사용하였다. 이 방법은 캐쉬 메모리 블록에 들어있는 유효 데이터를 방출하는 시점을 조절하여 외부 메모리 접근이 다량으로 몰리는 것을 방지하는 것이다. 즉 같은 메모리 용량과 접근 성공율을 가지는 캐쉬의 성능을 증가 시킬 수 있는 방법이다. 이를 통하여 메모리 병목 현상을 완화시킬 수 있고 또한 캐쉬 접근 실패 시 소요되는 평균 메모리 접근 소요시간을 줄일 수 있다. 이러한 새로운 캐쉬 구조를 위한 실험은 ARM11, 3차원 그래픽 가속기 및 다양한 IP들이 내장되어 있는 SoC 환경에서 3차원 그래픽 가속기의 깊이 캐쉬와 픽셀 캐쉬에 적용하여 진행하였으며 여러 가지 실험 벡터를 이용하여 결과를 측정하였을 때 성능을 향상 시킬 수 있다.

Abstract

Generally, depth cache and pixel cache of 3D graphics are designed by using write-back scheme for effieient use of memory bandwidth. Also, there are write after read operations of same address or only write operations are occurred frequently in 3D graphics cache. If a cache miss is detected, an access to the external memory for write back operation and another access to the memory for handling the cache miss are operated simultaneously. So on frequent cache miss situations, as the memory access bandwidth limited, the access time of the external memory will be increased due to memory bottleneck problem. As a result, the total performance of the processor or the IP will be decreased, also the problem will increase peak power consumption. So in this paper, we proposed a novel early write back cache architecture so as to solve the problems issued above. The proposed architecture controls the point when to access the external memory as to copy the valid data block. And this architecture can improve the cache performance with same hit ratio and same capacity cache. As a result, the proposed architecture can solve the memory bottleneck problem by preventing intensive memory accesses. We have evaluated the new proposed architecture on 3D graphics z cache and pixel cache on a SoC environment where ARM11, 3D graphic accelerator and various IPs are embedded. The simulation results indicated that there were maximum 75% of performance increase when using various simulation vectors.

Keywords : 3D graphics, pixel cache, wirte-back, write buffer, memory bottleneck

* 정회원, *** 평생회원, 연세대학교 전기전자공학과
(Dept., Electrical & Electronics Eng. Yonsei University)
** 정회원, 삼성전자
(Samsung Electronics)
접수일자: 2009년3월24일, 수정완료일: 2009년11월16일

I. 서 론

최근 SoC(System On a Chip)분야에서 다양한 멀티미디어와 디스플레이의 발전으로 한 프레임에 처리해야 할 픽셀의 수가 증가하면서 시스템 버스를 통하여 외부 메모리를 접근하는 빈도가 높아지고 있다. 이러한 이유로 메모리 병목현상이 점점 심해지고 있고 이로 인한 메모리 데이터 접근 시간 증가가 프로세서와 주요 SoC IP들의 성능 저하와 모바일 환경에서 중요한 전력 소비 증가를 유발시킨다.

이를 해결하기 위하여 프로세서와 멀티미디어 IP는 다양한 종류와 레벨의 온 칩 캐쉬(on chip cache)를 사용하여 외부 메모리 접근 빈도를 줄이려는 노력을 하고 있다. 그리고 온 칩 시스템 버스는 좀 더 높은 성능을 위하여 네트워킹이 단순하지만 성능이 떨어지는 쉐어드(shared) 버스 구조에서 네트워킹은 복잡하지만 높은 성능을 낼 수 있는 포인트 투 포인트(point to point) 버스 구조로 바뀌고 있다. 또한 메모리 컨트롤러는 메모리의 페이지 히트(page hit)와 메모리 뱅크 인터리빙(bank interleaving)등을 효율적으로 이용 할 수 있도록 메모리 접근 순서를 스케쥴링(scheduling)을 하여 메모리의 사용 효율을 높이기 위한 노력을 하고 있다. 고성능 프로세서와 멀티미디어 IP를 설계하기 위해서는 일정한 크기 이상의 메모리 대역폭(bandwidth)을 확보해야 메모리 병목현상 없이 메인 메모리와의 접근을 할 수 있지만 일반적으로 모바일 환경에서는 사용할 수 있는 대역폭이 제한적이다. 따라서 대량의 데이터 전송은 메모리 병목현상이 심화시켜 IP의 성능 저하는 물론 시스템 전체의 성능을 저하시킬 수 있다. 이러한 병목현상은 최근 임베디드용 시스템 버스가 멀티플 아웃스탠딩 트랜젝션(multiple outstnading transaction)과 아웃오브 오더 엑세스(out-of-order access)를 지원하면서 더욱 심각해지고 있다. 또한 이러한 현상은 모바일 환경에서 중요한 요소 가운데 하나인 전력 소모를 증가시킨다.

일반적으로 캐쉬를 설계할 때 중요한 요소로는 캐쉬 메모리의 용량, 블록의 크기, 구조, 쓰기 정책(write policy), 블록 교체 정책(replacement policy) 등을 고려하여 설계해야 한다. 그리고 캐쉬의 성능 측정을 위해서 캐쉬 접근 성공률, 평균 메모리 접근시간, 캐쉬 접근 실패의 경우 메모리 접근 소요시간 등을 계산해야 한다. 캐쉬의 평균 메모리 접근시간과 캐쉬 접근 실패

Average memory access time =
Hit rate * hit time + miss rate * miss penalty

Miss penalty =
Memory latency + transfer time + delivery time

그림 1. 평균 메모리 접근 시간과 캐쉬 접근 실패에 따른 손실

Fig. 1. Average memory access time and miss penalty.

패널티는 그림 1과 같이 나타낼 수 있다. 여기서 전송시간(transfer time)은 메모리에서 캐쉬 컨트롤러로 보내는 시간을 나타내고 도달시간(delivery time)은 캐쉬 컨트롤러에 전달된 데이터가 실제로 데이터를 필요로 하는 IP까지 전달되는 시간을 의미한다. 캐쉬의 성능은 캐쉬 접근 성공률과 캐쉬 접근 실패 패널티에 따라 좌우되고 중 캐쉬 접근 성공률은 캐쉬 메모리의 용량, 캐쉬 블록의 크기, 캐쉬의 구조 그리고 캐쉬 접근을 시도하는 데이터의 패턴에 따라 좌우되며, 캐쉬 접근 실패 손실은 시스템 버스의 구조, 메모리 컨트롤러의 성능 그리고 메인 메모리의 성능에 따라 좌우 된다. 최근에는 이와 같이 성능에 영향을 주는 캐쉬 성능을 향상시키기 위하여 논 블록킹 캐쉬, 다단계 캐쉬(multi level cache) 그리고 프리패칭(pre-fetching) 방법 등을 사용하여 버스 및 메모리에 트래픽이 집중되는 현상을 완화시키려 하는 다양한 연구가 진행되고 있다.[5~6].

본 논문에서는 캐쉬의 접근 실패 손실을 줄이기 위하여 빠른 라이트백 방법을 제안하고 이 방법을 3차원 그래픽 하드웨어 가속기에 사용되는 깊이 캐쉬와 픽셀 캐쉬에 verilog HDL을 사용하여 구현한 후 적용 전후의 성능을 비교하고자 한다.

II. 본 론

그림 2는 3차원 그래픽 하드웨어 가속기의 블록 다이어그램(block diagram)이다. 3차원 그래픽 가속기는 입력되는 정보 단위를 기준으로 크게 기하(geometry) 처리 단계와 래스터라이제이션(rasterization) 처리 단계로 나뉠 수 있다. 기하 처리 단계는 입력된 물체의 정점(vertex)들에 대한 위치 변환 , 색상 계산 등을 수행한다. 래스라이제이션 단계는 기하 연산이 수행된 정점들의 정보를 가공하여 2차원 화면에 출력하기 위한 픽셀 정보로 변환한다. 픽셀 변환 작업은 실제와 유사한 표현 질감을 표시하기 위한 텍스쳐 맵핑(texture

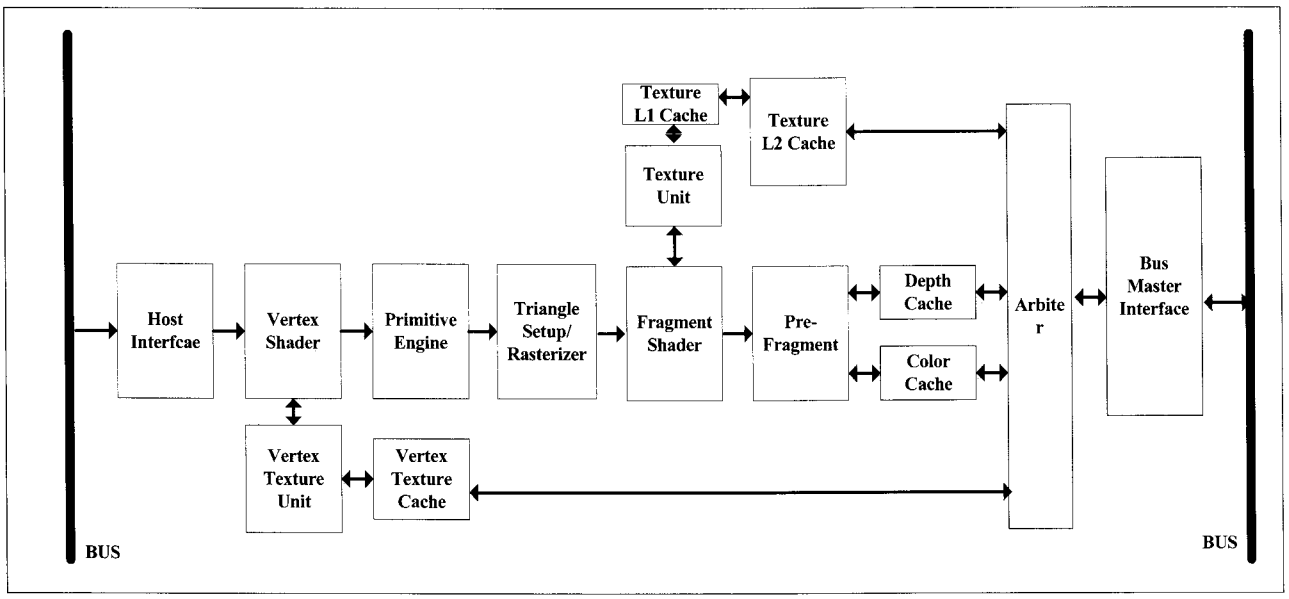


그림 2. 3차원 그래픽 가속기 블록 다이어그램
Fig. 2. Block diagram of 3D hardware accelerator.

mapping) 단계를 거치고 다양한 연산을 통하여 픽셀 정보를 가공한 다음 그 가공된 픽셀을 화면에 출력하기 위한 픽셀 쓰기 단계로 구성된다[4]. 래스터라이제이션 처리단계에서 필요한 메모리 전송양은 3차원 그래픽 가속기 전체 메모리 전송양의 많은 부분을 차지하고 있다. 그 중 가장 메모리 전송양이 많이 필요한 부분은 픽셀 처리 파이프라인 단계의 텍스쳐 메모리 접근과 프레임 버퍼 메모리 접근에서 일어난다. 최근에는 좀 더 다양하고 현실감 있는 이미지를 표현하기 위하여 여러 이미지의 텍스쳐를 사용하고 요구하는 해상도 또한 높아지고 있다. 이로 인하여 점점 더 필요한 메모리 전송양은 크게 증가하는 추세이다.

그림 3은 3차원 그래픽 하드웨어 가속기의 퍼-프레그먼트(per-fragment) 파이프라인을 나타낸 것이고 이 중 메모리 접근이 필요한 경우는 깊이와 스텐실 검사 그리고 알파 혼합과 논리 연산이다. 깊이 검사는 메모리에서 깊이 값을 읽어와서 현재 진행되는 프레그먼트의 깊이 값과 비교한 후 깊이 검사가 통과되는 프레그먼트의 깊이 값만 저장한다. 그리고 알파 혼합과 논리 연산은 메모리에서 프레그먼트의 픽셀 정보 읽기 후 데이터를 가공하여 다시 저장한다. 따라서 3차원 그래픽에서는 캐쉬 읽기 접근이 시도된 주소의 쓰기 접근이 시도될 확률이 높다는 특징을 가지고 있다. 즉 캐쉬 블록이 더티가 되는 현상이 많이 발생하며 이로 인한 라이트 백 동작이 빈번히 발생한다.

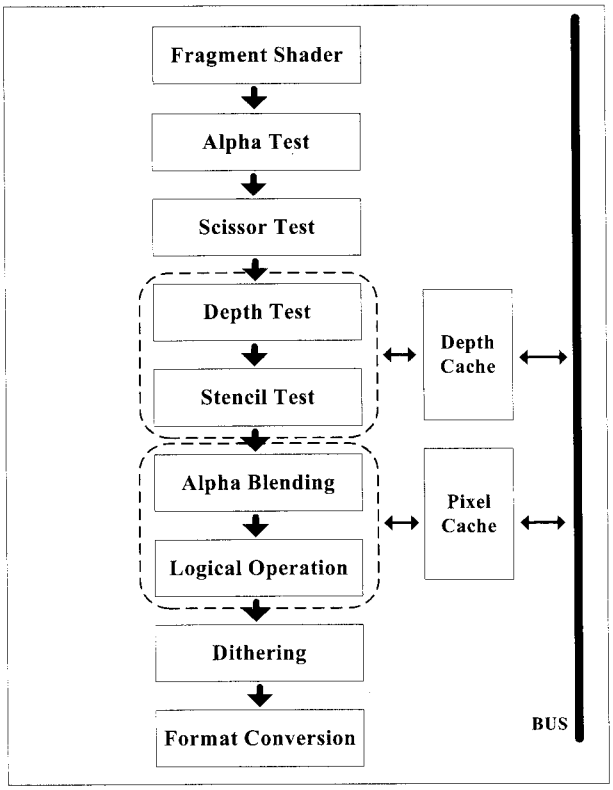


그림 3. 퍼 프래그먼트 파이프라인
Fig. 3. Per-fragment pipeline.

3차원 렌더링 시 정교하고 복잡한 모델링을 하고 다수의 텍스쳐를 사용하고 여러 텍셀(texel)을 이용하여 텍스쳐 필터링을 수행하고 이를 고해상도의 화면으로 구성한다면 텍스쳐 메모리와 프레임 버퍼 메모리의 전

송량은 기하급수적으로 증가하게 된다. 그러나 사용가능한 메모리 대역폭은 한계가 있기 때문에 요구하는 정보를 메모리 지연 없이 빠르게 전송하기가 어렵다. 이러한 메모리 대역폭 문제를 해결하기 위해와 3차원 그래픽 가속기에서는 텍스쳐 캐쉬, 깊이 캐쉬, 픽셀 캐쉬 등 다양한 캐쉬를 사용하고 있다[2~3]. 그 중 픽셀 캐쉬는 알파 혼합이나 논리 연산이 선택되는 경우에만 메모리 읽기와 쓰기 동작이 모두 수행되고 그 경우를 제외한 나머지 경우에서는 단지 메모리 쓰기 동작만이 수행된다. 하지만 한 프레임에서도 알파 혼합과 논리 연산 기능이 수행되는 경우와 수행 되지 않는 경우가 반복이 될 수 있다. 따라서 캐쉬의 동작 모드를 알파 혼합이나 논리 연산이 수행되는 경우는 일반적인 캐쉬 처럼 동작을 하는 캐쉬 모드로 동작하고 나머지 경우에는 메모리 읽기 접근이 없는 쓰기 버퍼 모드로 동작을 하도록 설계를 하면 캐쉬 성능 향상에 기여 할 수있다. 캐쉬 설계 시 사용되는 블록 교체 알고리즘은 가장 오랜 시간 동안 사용되지 않은 블록을 교체하는 LRU(Least Recently Used) 알고리즘과 정보가 입력된 순서대로 교체 순서가 결정이 되는 FIFO(First In First Out) 알고리즘 그리고 교체 되어야 하는 블록이 임의로 결정되는 랜덤(random) 알고리즘이 가장 많이 사용된다. 그 중 LRU 알고리즘은 가장 많은 성능을 향상 시킬 수 있지만 하드웨어 구현이 복잡하다는 단점이 있고 FIFO 알고리즘은 가장 간단한 구조로 구현이 용이하나 최고의 성능을 낼 수 없다는 단점이 있다.

3차원 그래픽 하드웨어 가속기의 텍스쳐링(texturing) 엔진의 경우에는 이미지의 개선을 위하여 다양한 필터링 방법을 사용한다. 주변 4개의 텍셀을 이용하는 바이리니어(bi-linear) 필터링, 밉맵(MIPMAP)과 주변 텍셀을 사용한 트라이리니어(tri-linear) 필터링 등이 가장 널리 사용되고 있다. 이러한 필터링을 사용하는 경우에는 근접한 위치의 텍셀 정보를 많이 사용한다. 따라서 텍스쳐 캐쉬를 설계할 때는 시간적 지역성과 공간적 지역성이 모두 중요하기 때문에 LRU 알고리즘을 사용하는 것이 성능 향상을 위해서 도움이 된다[1].

3차원 그래픽의 래스터라이제이션 진행 방향은 일반적으로 왼쪽에서 오른쪽 방향으로 진행이 되고 한 스캔에 대한 렌더링이 끝나면 아래쪽의 다음 스캔으로 넘어가서 렌더링을 수행한다. 개체의 크기가 커질수록 스캔에서 처리해야 하는 픽셀의 수가 많아지기 때문에 캐쉬

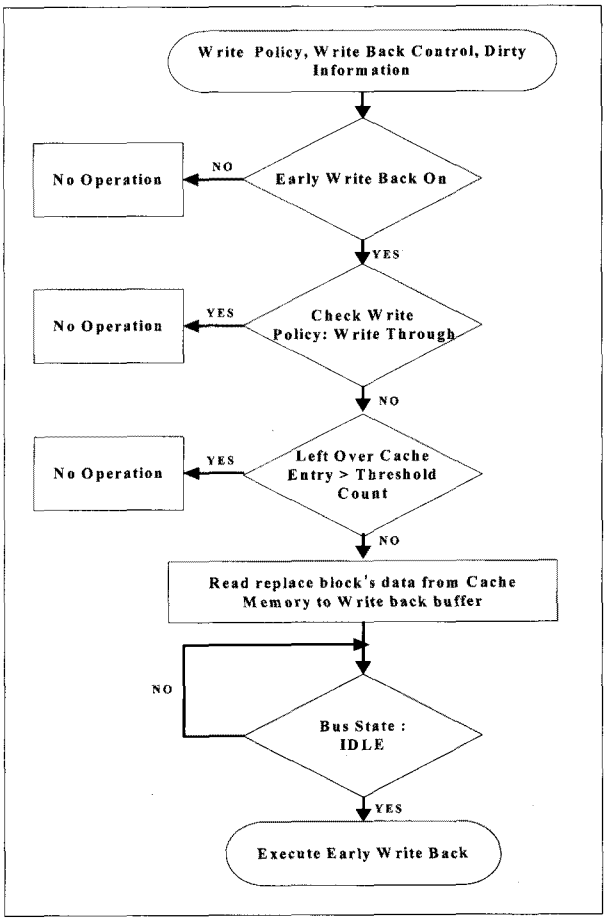


그림 4. 빠른 라이트 백 순서도
Fig. 4. Flow chart of early write-back operation.

한 블록에 저장된 데이터가 시간이 흐른 뒤 다시 사용될 가능성이 낮다. 왜냐하면 일반적으로 캐쉬 메모리의 용량이 작기 때문에 이 주소가 다음에 접근될 때는 이미 캐쉬에서 방출되었을 확률이 높기 때문이다. 즉 시간적인 지역성보다 공간적인 지역성이 중요하게 된다. 따라서 픽셀 캐쉬의 교체 알고리즘은 하드웨어 구현이 간단한 FIFO 알고리즘을 사용하여도 좋은 성능을 얻을 수 있다. 모바일 SoC 환경에서는 여러 가지 IP들이 외부 메모리를 공유하여 사용하고 또 이들이 동시에 메모리 접근을 시도하기 때문에 만약 한 IP에 있는 캐쉬에서 여러 번은 메모리 접근이 발생하면 메모리는 위와 같은 다양한 IP에서 시도하는 메모리 접근과 더불어 캐쉬에서 요청하는 메모리 접근이 더해져서 결국 다량의 메모리 접근이 집중적으로 몰리기 때문에 캐쉬에서 요청한 데이터에 대한 응답은 많은 클록 주기 후에나 처리 되게 되고 이는 결국 캐쉬 접근 실패 손실을 증가시켜 캐쉬의 성능을 저하시키게 된다. 이러한 캐쉬 접근 실패에 대한 손실을 줄이기 위하여 본 논문에서는 메모

리 쓰기 접근이 발생하는 캐쉬 라이트 백의 시점을 조절하는 방법을 사용 하였다. 즉 라이트 백에 의한 메모리 쓰기 접근이 발생하는 시점을 시스템이 사용되지 않는 시점에 미리 수행하는 것이다. 이와 같은 방법을 사용하면 캐쉬 접근 실패가 발생한 경우 생기는 메모리 접근이 짧은 시간에 다량으로 집중되는 것을 방지할 수가 있게 되어 병목 현상을 완화 시킬 수 있다. 이 방법을 좀 더 구체적으로 살펴보면 캐쉬의 모든 블록이 사용되는 상태가 되면 캐쉬 접근이 없어도, 즉 캐쉬의 접근 실패가 발생하지 않아도 미리 캐쉬 블록 교체 정책에 의해 다음 캐쉬 접근이 실패했을 경우 메인 메모리로 써져야 하는 블록의 데이터를 미리 메모리로 라이트 백 하는 것이다. 이러한 캐쉬 라이트 백 동작을 본 논문에서는 빠른 라이트 백 기법이라 명하였다. 이 방법의 장점은 메모리 병목 현상을 줄일 수 있게 되어 캐쉬 접근 실패가 발생한 경우에 캐쉬 접근 실패 손실을 줄일 수 있다는 것이다.

그림 4는 위에서 설명한 빠른 라이트 백 기법의 순서도를 나타낸 것이다. 캐쉬 컨트롤러는 쓰기 정책, 빠른 라이트 백의 수행여부, 라이트 백을 시행하는 시점을 소프트웨어 사용자로부터 입력 받는다. 만약 사용자가 빠른 라이트 백을 원하지 않으면 이 동작을 수행하지 않는다. 또한 사용자가 빠른 라이트 백을 수행하려 하여도 만약 쓰기 정책이 라이트 쓰루인 경우에는 이 동작을 수행하지 않는다. 이러한 경우를 제외하면 빠른 라이트 백이 수행되는데 이 때 사용자는 캐쉬의 블록이 얼마나 남았을 때 빠른 라이트 백을 수행할지 결정해 주어야 한다. 만약 빠른 라이트 백 된 블록이 교체되기 전에 캐쉬 읽기 접근 성공이 발생하면 라이트 백 한 블록이 캐쉬 메모리에 여전히 남아 있기 때문에 캐쉬 접근 성공율에 지장을 주지 않지만 쓰기 접근이 성공하는 경우에는 그 블록을 한 번 더 메모리에 라이트 백을 해야 하기 때문에 불필요한 메모리 쓰기 동작이 발생하게 된다. 그러므로 항상 많은 개수의 블록을 미리 라이트 백 하는 것이 아니라 소프트웨어 사용자가 응용 프로그램(application)에 따라서 캐쉬에 들어오는 주소 패턴을 예측하여 미리 라이트 백 되는 블록의 개수, 즉 임계 값을 선택할 수 있게 하였다. 일반적으로 3차원 모델의 개체의 크기가 작은 경우 임계 값은 '0'으로 설정하고 개체의 크기가 큰 경우에는 조금 더 큰 값으로 설정해 주면 좀 더 좋은 성능을 낼 수 있다. 일반적으로 3차원 그래픽 픽셀 캐쉬에서는 개체의 크기가 크면 주로 공간적인 지역성만 존재하게 되어 빠른 라이트 백의 임계값이 큰 경우에도 효율적으로 동작하지만 개체의 크기가 작은 경우에는 시간적 지역성도 생기기 때문에 임계 값을 작게 설정하는 것이 더 효율적이다. 캐쉬 메모리의 사용 가능한 블록의 개수가 사용자가 결정한 수보다 작으면 캐쉬 접근이 없어도 다음번에 교체 되어야 할 블록의 정보를 라이트 백 버퍼로 전송한 다음에 시스템 버스가 사용 가능한지 확인한다. 만약 버스가 사용 가능하면 라이트 백 동작을 실행하고 사용가능 하지 않으면 가능할 때까지 기다리다가 라이트 백을 실행한다. 이러한 빠른 라이트 백 방법은 집중적인 메모리 접근을 분산 시켜주는 역할을 하기 때문에 모바일 SoC와 같은 다양한 IP들이 외부 메모리를 공동으로 사용하는 시스템에 더욱 더 적합하다. 그림 5는 실제로 설계한 빠른 라이트 백 방법을 적용한 픽셀 캐쉬의 블록 다이어그램이다. 빠른 라이트 백 방법을 적용하기 위해 몇 가지 블록이 추가 되었다. 빠른 라이트 백 조건 조사(early write back condition check), 빠른 라이트 백 결정 블록(early write back decision block), 빠른 라이트 백 실행 상태 기계(early write back execution state machine) 등 몇 가지의 블록이 추가되었다.

본 논문에서는 3차원 그래픽 가속기의 픽셀 캐쉬에 빠른 라이트 백 기법을 적용하고 그것을 ARM11 CPU가 집적되어 있는 SoC 플랫폼에 집적하여 이 방법의 적용 전과 후의 3차원 그래픽 가속기의 성능을 비교하고자 한다.

III. 실 험

그림 5는 실험에 사용된 SoC 환경의 블록 다이어그램이다. 본 논문의 정확한 실험을 위하여 ARM11 CPU를 집적한 SoC 플랫폼을 사용하였다. 이 플랫폼에는 ARM11를 비롯한 버스 마스터 IP들과 버스 슬레이브 IP들이 집적되어 있고 시스템 버스는 좀 더 높은 성능을 얻기 위하여 멀티플 아웃스탠딩 트랜젝션이 지원되는 AMBA3.0 AXI 프로토콜을 사용하였다. 실험에 사용되는 SoC의 모든 모든 IP는 verilg HDL을 사용하여 구현되어 있고 외부 메인 메모리는 512Mbit DDR SDRAM verilog 모델을 사용하여 실험을 진행하였다. 실험 벡터는 Synopsys사의 VERA를 사용하여 작성하였고 VERA 인터페이스를 통하여 ARM 모델을 구동시켰다.

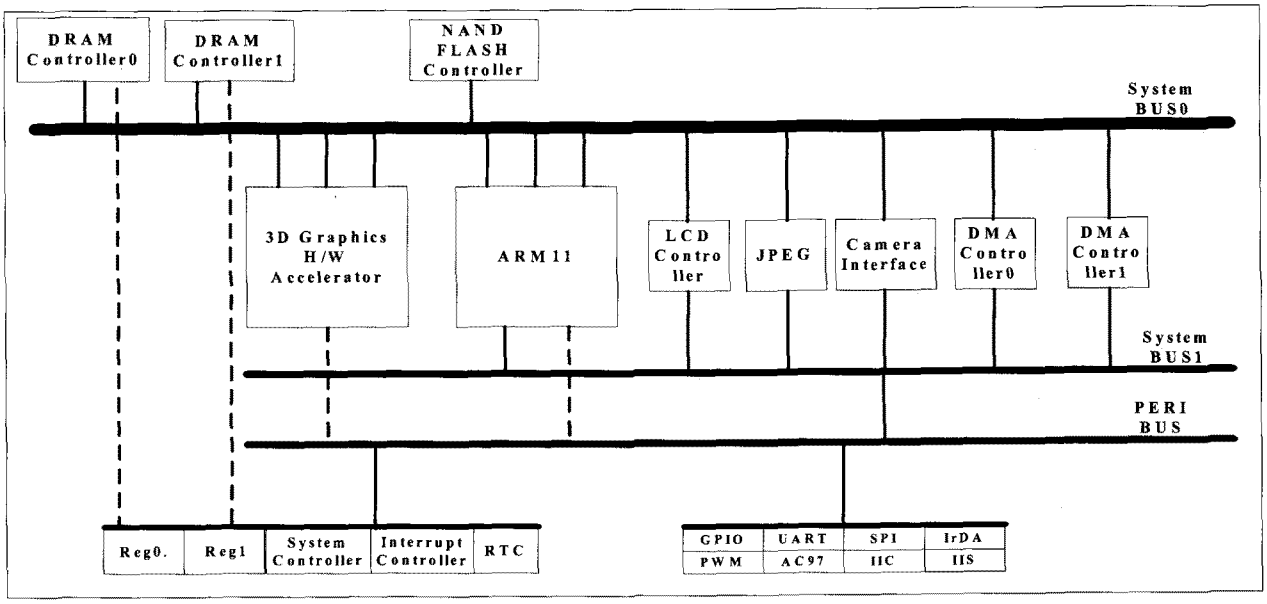


그림 5. 실험 환경 SoC 블록 다이어그램
Fig. 5. Simulatuion Environment SoC block diagram.

표 1. 첫 번째 실험의 빠른 라이트 백 적용 전후 결과 비교
Table 1. Result of the first experiment.

	Before		After	
	single pipeline	dual pipeline	single pipeline	dual pipeline
fillrate (MPixel/sec)	94.58	156.79	162.91	208.53

실험의 비교는 일반적인 라이트 백 방법을 가지는 캐쉬와 본 논문에서 사용한 빠른 라이트 백 방법을 사용하는 캐쉬를 각각 동일한 3차원 그래픽 가속기에 집적하여 그 가속기가 3차원 모델을 렌더링하여 화면을 채우는 속도를 측정하여 비교하였다.

실험은 크게 두 가지로 진행하였다. 첫 번째 실험은 하나의 커다란 3차원 개체를 렌더링을 하는 것이다. 이 경우는 깊이 검사, 알파 혼합 그리고 로직 연산 동작이 수행되지 않기 때문에 픽셀 캐쉬 쓰기 접근만 발생한다. 두 번째 실험은 정교한 3차원 모델을 렌더링 하는 것인데 이 경우에는 깊이 캐쉬와 픽셀 캐쉬에 읽기 접근과 쓰기 접근이 모두 발생한다.

본 논문에서 언급한 빠른 라이트백 방법은 3차원 그래픽 하드웨어 가속기에 내장 되어 있는 깊이 캐쉬와 픽셀 캐쉬에 구현 하였고 빠른 라이트 백이 수행되는 시점은 캐쉬 메모리가 모두 사용되는 경우, 즉 임계값을 첫 번째 실험은 '1', 두 번째 실험은 '0'으로 설정하고 실험을 진행하였을 때 가장 좋은 성능을 볼 수 있었다. 첫 번째 실험에서는 하나의 개체를 사용하여 렌더링을 진행하는 경우는 공간적 지역성만 높기 때문에 오랜 시간 사용하지 않은 캐쉬 블록을 다시 사용할 확률이 적기 때문에 임계값이 '1'일 때 가장 좋은 성능을 볼 수 있었고 두 번째 실험에서는 첫 번째 실험과는 다르게 시간적 지역성이 증가하기 때문에 빠른 라이트 백이 수행된 블록에 대한 읽기 접근이 성공하는 경우는 성능에 영향을 주지 않지만 그 블록에 다시 한번 쓰기 접근이 성공할 가능성이 발생하기 때문에 임계 값을 '0'으로 선택하였을 때 좋은 성능을 볼 수 있었다. 따라서 소프트웨어 사용자는 배경 이미지를 렌더링하는 경우에는 빠른 라이트 백 수행 임계 값을 약간 큰 값으로 정교한 모델을 렌더링하는 경우는 임계 값을 작은 값으로 선택을 하는 것이 성능을 높이는데 유리하다.

픽셀 캐쉬의 용량은 4KByte, 구조는 8-way set-associative, 쓰기 정책은 라이트 백, 블록 교체 정책은 FIFO 방식을 사용하였다. 빠른 라이트 백 방법 적용 후 합성 결과 캐쉬의 면적은 기존 캐쉬 컨트롤러 면적 보다 2% 미만으로 증가하여 추가적인 하드웨어 부담은 거의 없다고 볼 수 있다.

본 논문의 실험에서 동작하는 IP는 다음과 같다. 전체 환경 제어 및 3차원 정점 데이터 정보 전송 역할을

하는 ARM11, 렌더링 진행을 위한 3차원 그래픽 가속기, 데이터 전송을 위한 DMA 컨트롤러, 실험 결과 이미지를 보기 위한 LCD 컨트롤러, 외부 DDR SDRAM과의 데이터 전송을 위한 DRAM 컨트롤러 그리고 3차원 그래픽 가속기 및 여러 IP들을 CPU가 제어하기 위한 인터럽트 컨트롤러 등도 같이 동작한다. 이러한 환경은 실제 3차원 그래픽 가속기를 이용하여 렌더링을 진행하면서 LCD화면으로 렌더링 결과를 디스플레이하는 디스플레이 컨트롤러 그리고 메모리 컨트롤러가 동작한다. 일반적으로 그래픽 랜더링 시 그래픽 IP의 메모리 접근이 전체 IP의 메모리 접근의 대부분을 차지하고 있고 또한 3차원 가속기가 렌더링을 진행하고 있는 경우에는 다른 IP가 시간 축에서 동시에 메인 메모리 접근을 시도하는 양은 3차원 그래픽 IP의 데이터 접근 양에 비하면 작은 편이다. 하지만 3차원 가속기에서도 쉐이더(shader)의 메모리 접근, 텍스쳐 캐쉬의 메모리 접근, 깊이 버퍼의 메모리 접근, 프레임 버퍼의 메모리 접근 등 다양한 메모리 접근이 여러 파이프라인에서 동시에 일어난다. 특히 좀 더 실감 있는 이미지를 높은 해상도로 렌더링하는 경우에는 전체 IP 중 3차원 그래픽 IP가 사용하는 메모리 접근 비율이 더욱 더 증가할 수 있다. 첫 번째 실험은 한 개의 개체를 이용하여 화면전체를 일정한 색상으로 렌더링을 진행하는 것이다. 즉 화면의 필 레이트(fill rate)를 측정하는 실험을 진행 하였다. 이 실험은 매우 간단하면서도 화면 지우기나 배경 이미지 렌더링 시 많이 사용되기 때문에 이 실험에서 속도를 높이는 것은 의미가 있다. 실험 벡터는 간단한 객체를 이용하여 320x240 해상도의 이미지를 렌더링하는 것이고 이 실험에 사용된 CPU 클록 주파수는 166MHz 그리고 메모리에 저장하는 픽셀 형태는 RGBA 32-bit이다. 또한 버스 트래픽에 따른 효과를 실험하기 위하여 3차원 그래픽 가속기의 래스터라이져 파이프라인이 1개인 경우와 2개인 경우에 각각 실험을 하였다. 이 실험에서는 텍스쳐가 사용되지 않았고 또한깊이 검사, 알파 혼합이 진행되지 않기 때문에 메모리 접근은 오직 픽셀 캐쉬의 프레임 버퍼 쓰기에만 시도된다. 실험 결과 픽셀 캐쉬 접근 성공률은 캐쉬의 용량이나 구조의 변화가 없었기 때문에 93.8%로 빠른 라이트 백과 상관없이 동일하게 측정되었다. 하지만 성능은 표 1과 같이 래스터라이져 파이프라인의 개수에 따라 각각 약 72%, 약 33%가 증가하였다. 실험 후 분석 결과 파이프라인의 수가 증가하면 메모리 병목 현상이 심해져서 성능 향상이 줄어들었음을 알 수 있었다. 이 실험의 경우 캐쉬 접근 성공률이 높기 때문에 메모리 접근 수가 많은 편은 아니다. 하지만 메모리 접근이 짧은 시간에 집중되어 발생하면 메모리 병목 현상으로 인한 성능 감소가 발생하고 메모리 접근을 시간 축에서 분배 시키면 메모리 접근 수가 같아도 더욱 뛰어난 성능을 얻을 수 있다는 것을 이 실험으로 증명하였다. 또한 이 실험을 통하여 메모리 접근을 시도하는 양이 증가 할수록 전체적인 시스템 성능저하로 인한 빠른 라이트 백 효과도 감소한다는 것을 알 수 있다. 이러한 시스템 성능 저하는 메모리 대역폭 증가시키거나 캐쉬의 용량을 늘려 요구하는 메모리 대역폭을 감소 시켜 해결할 수 있다. 하지만 본 논문은 캐쉬 용량이나 구조의 변화 없이 새

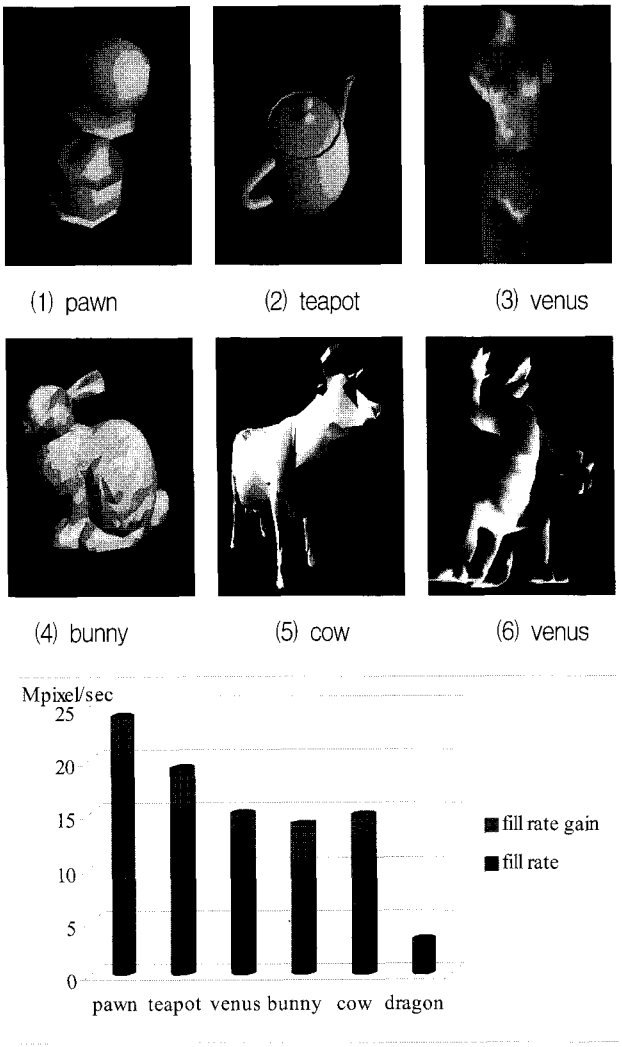


그림 6 두 번째 실험의 시뮬레이션 결과 이미지와 성능 비교

Fig. 6. Results of the second experiment and the result images.

로운 알고리즘을 이용하여 캐쉬의 성능을 증가시키는데 초점을 맞추고자 한다.

두 번째 실험은 다양한 3차원 모델을 렌더링 하였을 때 빠른 라이트백 방법의 효과를 측정하고자 한다. 이 실험을 위하여 드래곤(dragon), 카우(cow), 버니(bunny), 비너스(venus), 티폿(teapot), 폰(pawn) 6가지의 모델을 사용하였고 각 모델에 대하여 이 방법의 적용 전과 후의 성능을 측정하였다. 6가지의 모델은 수천 개 혹은 그 이상의 정점으로 구성되어 있으며 각 모델의 픽셀 수 는 대략 14,000개에서 42,000개 정도가 된다.

3차원 그래픽 가속기에서 버스 트래픽을 만드는 캐쉬는 깊이 캐쉬와 픽셀 캐쉬가 있다. 깊이 캐쉬는 외부 메모리, 즉 Z 버퍼에 있는 깊이 값을 읽는 동작과 깊이 검사를 통과한 깊이 값을 쓰는 동작을 한다. 즉 캐쉬 읽기 접근이 발생하고 그 데이터를 이용하여 깊이 검사를 수행한 후 검사를 통과하면 같은 주소의 깊이 데이터를 갱신하는 캐쉬 쓰기 접근이 발생한다. 그리고 픽셀 캐쉬에서는 읽기 동작을 수행하여 픽셀 값을 읽어 온 후 알파 혼합과 논리 연산 수행을 하고 그 픽셀 값을 다시 메모리에 쓰는 동작을 수행하고 이러한 기능들이 사용되지 않는 경우는 프레임 버퍼 쓰기 동작만 발생한다. 즉 같은 주소의 캐쉬 읽기 접근 후의 쓰기 접근이 발생하거나 캐쉬 쓰기만 발생한다. 각 모델들의 3차원 모델링이 많은 수의 개체를 사용하여 정교하게 되어 있기 때문에 깊이 캐쉬나 픽셀 캐쉬의 읽기 접근과 쓰기 접근의 수도 많아지고 접근 성공률 또한 첫 번째 실험보다 많이 떨어지게 된다. 따라서 캐쉬 접근 실패의 빈도가 높아지고 버스의 트래픽 또한 증가하게 된다. 그림 7은 6가지 모델에 대하여 렌더링을 진행한 실험 결과를 나타낸 그래프 이다. 그림 7의 성능 비교 그래프에서 볼 수 있듯이 빠른 라이트 백 방법 적용 시 적용 전에 비하여 모델에 따라 최소 9.9%, 최대 20.4% 의 필 레이트 성능이 증가하는 것을 볼 수 있다. 그리고 6개의 모델 평균 성능은 15.1% 필 레이트가 증가하였다. 렌더링하는 픽셀의 수는 폰(pawn) 모델이 13878개로 가장 작고 카우(cow) 모델이 41312개로 가장 많다. 하지만 필 레이트 성능은 폰 모델이 가장 좋고 드래곤(dragon) 모델이 가장 나쁘게 측정 되었다. 렌더링을 진행하는 픽셀의 수는 카우 모델이 가장 많지만 드래곤의 모델링이 더 복잡하게 되어 있기 때문에 드래곤 모델의 캐쉬 접근 성공률이 카우 모델 보다 낮아 필 레이트가 가장 낮게 측정이 되었다.

Ⅳ. 결 론

본 논문에서는 캐쉬에 접근 실패가 발생한 경우에 생기는 메모리 접근 소요 시간을 줄이기 위하여 빠른 라이트 백이라는 새로운 캐쉬 라이트 백 방법을 제안하고 이를 verilog HDL을 통하여 3차원 그래픽 가속기에 사용되는 깊이 캐쉬와 픽셀 캐쉬에 구현 하였다. 또한 빠른 라이트백의 실제 효과를 측정하기 위하여 ARM11을 비롯한 여러 IP들이 내장되어 있는 SoC 플랫폼에서 이 방법에 대한 실험을 진행 하였다.

빠른 라이트 백 방법을 적용하기 전 실험 결과를 보면 캐쉬 접근 실패가 발생하는 경우 순간적으로 메모리 접근이 다량으로 발생하게 되어 메모리 병목 현상이 심해지고 결국 이로 인하여 원하는 데이터 전송을 위한 메모리 접근 소요 시간이 길어지는 것을 볼 수가 있었다. 이를 빠른 라이트 백 방법을 적용한 후 살펴보면 순간적으로 많은 양의 메모리 접근이 발생하는 것을 방지하게 되어 메모리 병목 현상을 완화시키는 것을 볼 수 있었다. 결국 캐쉬 접근 실패가 발생하는 경우 원하는 메모리 접근을 좀 더 빠른 시간에 할 수 있게 되어 3차원 그래픽 가속기의 성능을 향상 시킬 수 있었다. 이 방법의 적용으로 몇 천 게이트 수 정도의 영역이 증가하였지만 약 300만 게이트 정도의 크기가 되는 3차원 그래픽 가속기의 성능을 크게 증가시킬 수 있었다. 실험 결과를 보면 간단한 객체를 이용하여 화면을 전체를 일정한 색상으로 렌더링하는 경우는 최대 72% 그리고 6가지 모델에 대해서는 평균 15.1% 향상을 볼 수 있었다.

결론적으로 CPU를 비롯한 여러 가지 IP들이 외부 메모리를 공유하는 모바일 환경에서 외부 메모리 접근이 순간적으로 집중되는 것이 성능 저하를 유발시킨다는 것과 외부 메모리 접근을 분산 시켜 주는 것이 성능 향상에 도움이 된다는 것을 증명하였다. 그리고 3차원 그래픽 IP 뿐만 아니라 다른 멀티미디어 IP에서도 사용되는 응용 프로그램에 적합하게 빠른 라이트 백이 실행되는 캐쉬 블록의 개수를 소프트웨어를 사용하여 최적화한다면 임베디드 환경에서 많은 면적을 필요로 하지 않고도 추가적으로 캐쉬의 성능을 향상시킬 수 있을 것으로 사료된다.

참 고 문 헌

[1] Z.S, Hakura and A. Gupta, "The design and analysis of a cache architecture for texture mapping" Proceeding of the 24th International Symposium on Computer Architecture, pp. 108-120,June 1997.

[2] Kil-Whan Lee, Woo-Chan Park, Il-San Kim, and Tack-Don Han, "A Pixel Cache Architecture with Selective Placement Scheme based on Z-test Result," Microprocessors and Microsystems, Vol. 29, Issue. 1, pp. 41-46, Feb. 2005.

[3] Woo-Chan Park, Kil-Whan Lee, Il-San Kim, Tack-Don Han, and Sung-Bong Yang, "An Effective Pixel Rasterization Pipeline Architecture for 3D Rendering Processors," IEEE Transactions on Computers, Vol. 52, No. 11, pp. 1501-1508, Nov. 2003.

[4] Tomas Akenine-Mö.ller, Eric Haines, "Real-Time Rendering," A K PETERS, 2002.

[5] H. Igehy, M. Eldridge, and K. Proudfoot, "Pre-fetching in a texture cache architecture" Proceedings of 1998 SIGGRAPH/Eurographics Workshop on Graphics Hardware, pp. 133-142, Aug 1998.

[6] Yong Surk Lee, "A Secondary Cache Controller Design for a Hign-End Microprocessor", IEEE Journal of Solid-State Circuits, Vol. 27, No. 8, pp1147-1146, August, 1992.

저 자 소 개

정 영 진(정회원)

2000년 한양대학교 전기공학과 학사

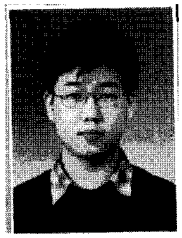
2000년 1월 삼성전자 메모리 사업부 DRAM 설계실.

2001년 11월 삼성전자 SystemLSI 사업부 SOC 연구소.

2008년~현재 삼성전자 SystemLSI 사업부 SOC개발실 책임연구원

2008년~연세대학교 전기전자공학과 석사과정 재학

<주관심분야 : 3차원 그래픽, H.264, 메모리 시스템, SOC>



이 길 환(정회원)

1996년 연세대학교 컴퓨터과학과 학사

1998년 연세대학교 컴퓨터과학과 석사

2004년 연세대학교 컴퓨터과학과 박사

2004년~현재 삼성전자 SystemLSI 사업부 SOC개발실 책임연구원

<주관심분야 : 그래픽 프로세서, 메모리시스템, 컴퓨터시스템>

이 용 석(평생회원)

1973년 연세대학교 전기공학과 학사

1977년 University of Michigan, Ann Arbor 전기공학과 석사

1981년 University of Michigan, Ann Arbor 전기공학과 박사

1993년~현재 연세대학교 전기전자공학과 교수

<주관심분야 : 마이크로 프로세서, 네트워크 프로세서, 고성능 연산기 설계, SOC>