메모리 시스템 구조 분석을 위한 시뮬레이터

강동우, 최종무
단국대학교 컴퓨터 학과
{kangdw, choijm}@dankook.ac.kr

Bank Level Simulator to Analysis Memory System

Dongwoo Kang, Jongmoo Choi
Dept of Computer Science, Dankook University

요 약

최근의 컴퓨터 시스템은 멀티 코어를 기반으로 병렬성 향상을 추구 하고 있지만 코어의 개수가 증가함에 따라 메모리가 새로운 병목 지점으로 지적되고 있다. 메모리 시스템은 가상 메모리, 물리 메모리, 뱅크 메모리 3계층으로 나눌 수 있으며, 각 계층은 상호연관 관계가 있어서 분석하기에 어려움이 있다. 본 논문에서는 이를 위해 계층 구조를 지원하는 시뮬레이터를 제안한다. 제안하는 시뮬레이터는 총 5개의 구성 요소로 이루어져 있으며, CPU 개수, 캐시 정책, 뱅크 개수등 다양한 설정을 지원한다. 또한 시뮬레이터를 통하여 운영체제 수준의 물리 메모리 관리자가 메모리 접근 지연에 영향이 있음을 보인다.

1. 서론

최근의 컴퓨터 시스템은 Intel사의 i7, Xeon 그리고 AMD의 Opteron과 같은 멀티 코어를 사용하여 병렬성 향상을 기반으로 시스템의 성능 향상을 추구하고 있다. 다수의 코어를 활용하기 위하여 대용량의 메모리를 장착하고 있으며, 대용량의 메모리는 다수의 뱅크로 구성된다. 각 뱅크에는 다수의 행과 버퍼로 사용하는 한 개의 로우버퍼가 존재한다. 만약 다수의 코어가 동시에 다른 행이지만 같은 뱅크에 접근하게 될 경우 로우버퍼 충돌이 발생한다. 로우버퍼 충돌은 메모리에 접근 할 때 마다 로우버퍼를 갱신하는 비용(Precharge Operation과 Activate Operation)이 발생하게 되며, 충돌이 발생하지 않은 경우에 비해 약 2배 정도의 시간이 소요되어 시스템의 성능 지연이 발생하게 된다.[1] 즉, 최근의 멀티 코어 기반의 컴퓨터 시스템에서는 메모리가 성능의 병목 지점이 되고 있다. 이러한 메모리 접근과 관련되어 쉽게 시스템의 성능 분석을 할 수 없는 건 가상 메모리, 물리 메모리, 뱅크 메모리로 이루어지는 계층 구조로 인함이다. 본 논문에서는 이러한 문제를 해결하고자 메모리 계층 구조를 지원하는 메모리 시스템 시뮬레이터를 제안한다. 제안하는 시뮬레이터는 응용 수준에서 접근하는 가상 메모리부터 운영체제 관점의 물리 메모리, 그리고 메모리 컨트롤러 입장의 뱅크 메모리까지의 유기적인 관계에 대한 시뮬레이션을 제공한다.

※ 이 논문은 2012년도 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임(No. 2012R1A2A2A01014233)

본 논문의 구성은 다음과 같다. 2절에서는 시스템 메모리 계층 구조에 대해 이야기 하고, 3절에서는 메모리 계층 구조 시뮬레이터에 대한 설계 요소들에 대한 설명과 실험 결과를 논한다. 이후 4절에서는 결론을 맺는다.

2. 배 경

컴퓨터 시스템에서 사용하는 메모리는 그림 1과 같이 3계층으로 나눠서 살펴 볼 수 있다. 먼저 응용 수준에서는 연속적인 공간을 제공해 주는 가상 메모리를 사용한다. 운영체제 수준에서는 연속적인 물리 메모리 주소를 기반으로 메모리를 할당/반납해주며, 응용이 접근하는 가상 메모리와 물리 메모리를 사상시키기 위해 페이지 테이블을 구성하는 기능을 제공한다. 메모리 컨트롤러 입장에서는 응용이 접근하려는 가상 주소를 MMU (Memory Management Unit)를 통해 변환된 물리주소를 받는다. 메모리 컨트롤러에 들어온 물리 주소를 뱅크/행/열에 대한 주소로 변환 후 실제 메모리에 접근을 수행한다.

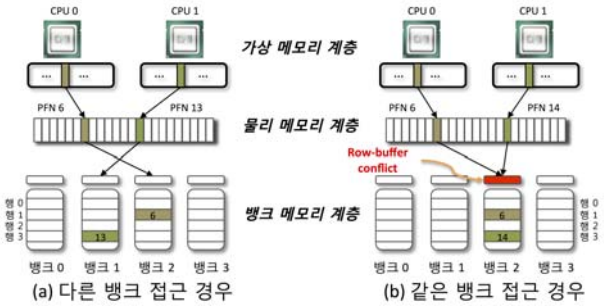

그림 1 3계층 메모리 구조

그림 1 (a)의 경우, CPU 0과 1은 특정 가상 메모리 주소를 기반으로 메모리에 접근하게 된다. 각각의 가상 메모리는 운영체제가 할당해 준 물리 메모리인 페이지 프레임 번호 6과 13에 사상되어 있다고 가정한다. 이러한 물리 메모리들은 뱅크/행/열에 대한 주소로 변환되어 특정 뱅크의 행/열에 접근하게 된다. 이 때 해당되는 행은 모두 로우 버퍼로 복사가 일어나며(Activate Operation) 메모리 컨트롤러는 각 뱅크의 로우 버퍼로부터 데이터를 읽어서 각 CPU에 전달한다. 이와 반대로 그림 1(b)와 같은 경우, 그림 1(a)와 같은 가상 메모리 주소라 하더라도 같은 뱅크에 해당하는 물리 메모리를 할당 받은 경우이다. 이러한 상황에서는 2개의 CPU가 동시에 접근을 수행하기 때문에 메모리 컨트롤러는 각 코어의 요청이 들어올 때 마다 해당 뱅크의 로우버퍼에 있는 데이터를 기존의 행에 저장한 후(Precharge Operation) 자신이 접근하려는 행을 로우버퍼에 저장하는 동작(Activate Operation)을 수행한다.

즉, 운영체제에서 응용의 메모리 요청을, 같은 뱅크에 있는 로우에 대한 주소로 할당하게 되면, 메모리 접근 시 로우 버퍼 충돌이 발생하게 된다. 다시 말해 응용 수준에서 메모리 접근으로 인한 지연은 접근하려는 가상 메모리에 대한 물리 메모리의 주소가 큰 영향을 미치는 것을 알 수 있다. 결국 메모리 접근에 대한 지연은 같은 뱅크에 대한 접근 유무에 따른 분석이 필요로 하며, 이를 위해 할당 받은 물리 메모리가 어떤 뱅크에 대한 접근인지 유기적인 분석 할 수 있는 환경이 필요로 하다. 하지만 기존의 DRAMsim[2]과 같은 시뮬레이터들은 이러한 계층 구조에 대한 지원을 하지 않기 때문에 응용 수준에서 발생하는 메모리 접근으로 인한 시스템의 성능 분석을 효과적으로 분석할 수 없다. 이에 본 논문에서는 유기적 계층 구조를 지원하는 시뮬레이터를 제안한다.

3. 메모리 시스템 시뮬레이터 설계 및 실험 결과

계층 구조 분석을 위해 본 논문에서는 뱅크 수준의 메모리 시뮬레이터를 개발하였다. 시뮬레이터는 총 5개의 구성요소로 이루어져 있다.

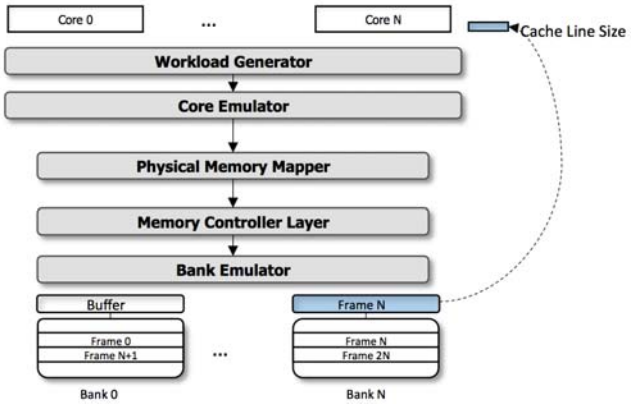

그림 2 메모리 시스템 시뮬레이터 구조

각 구성 요소는 다음과 같은 특징을 가지고 있다.

1. Workload generator : 응용 수준에서 가상 메모리에 대한 순차 접근과 비순차 접근을 수행한다. 또한 pin tool[3]을 통해 실제 워크로드들의 수행을 추출하여 재수행 기능을 제공한다.

2. Core emulator : 워크로드의 개수와 동일한 CPU 개수를 설정하고 CPU의 캐시와 MMU를 에뮬레이션 한다. CPU의 명령어 캐시와 데이터 캐시의 크기를 설정할 수 있으며, 캐시 알고리즘은 set-associative 방식으로 구동된다. 또한 Workload Generator를 통하여 발생되는 가상 메모리 주소를 MMU를 통해 페이징을 수행하여 물리 메모리 주소로 변환한다. 만약 접근하려는 가상 메모리에 대한 물리 주소가 할당 받지 않았을 경우, 기존 시스템과 마찬가지로 페이지 폴트를 발생시키고 Physical Memroy Mapper에게 새로운 물리 메모리를 요청한다.

3. Physical Memory Mapper : 물리 메모리를 관리하는 컴포넌트로 Lazy Buddy System을 통해 설정된 크기의 물리 메모리를 할당/반납 해 준다. 각 CPU마다 소유할 수 있는 Bucket 크기는 31로 고정한다.

4. Memory Controller Layer: 메모리 컨트롤러는 크게 3개의 기능을 지원한다. 먼저 물리 메모리 주소를 기반으로 발생한 메모리 접근에 대한 요청을 FR-FCFS(First Ready-First Come First Served) 스케줄러를 통해 처리하며, 뱅크에 대한 주소 변환을 수행한다.

5. Bank emulator : 각 뱅크에 대한 Activate Operation과 같은 동작들과 로우버퍼를 상태 머신을 기반으로 에뮬레이션 해주는 구성요소다.

본 논문에서 제안하는 시뮬레이터는 다양한 결과를 분석할 수 있다. 먼저 각 뱅크마다 발생하는 각 동작들에 대한 횟수와 명령어 캐시와 데이터 캐시에 대한 접근 횟수 및 미스 횟수, 그리고 총 명령어 수와 메모리에 접근한 횟수를 분석할 수 있다. 2절에서 언급한대로 물리 메모리 할당자에 따른 로우버퍼 충돌에 대한 영향력을 분석하기 위하여 SPEC CPU 2006 Benchmark 중 메모리 집중적인 soplex와 libquantum[4]을 pin tool을 통해 명령어와 접근한 가상 메모리 주소를 추출 후, 시뮬레이션을 수행하였다. 그림 3은 기존 운영체제의 메모리 관리자인 Lazy Buddy System과 임의로 할당해 주는 방식을 비교한 결과이다. 기존의 메모리 관리 기법은 임의 방식의 메모리 관리 기법보다 약 5%의 로우버퍼 충돌이 많이 일어난 것을 알 수 있으며, 이로 인해 성능 저하를 불러일으킬 수 있음을 보여주고 있다.

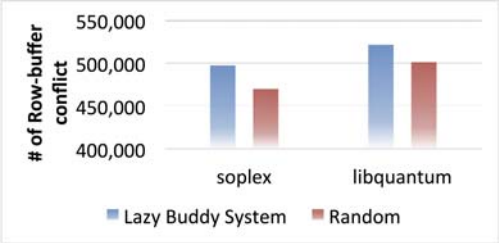

그림 3 로우버퍼 충돌 횟수 분석

4. 결론 및 향후 연구

메모리 시스템은 3개의 계층 구조가 얽혀있는 형태로 구성된다. 이러한 복합적인 구조는 시스템 분석을 어렵게 하기 때문에 본 논문에서는 이러한 환경을 기반으로 다양한 설정을 지원하며 테스트를 수행할 수 있는 시뮬레이터를 제안하였다. 또한 기존의 운영체제 수준의 물리 메모리 관리자가 메모리 접근의 지연을 발생시킬 수 있음을 보였다.

향후 다양한 메모리 관리 기법을 개발 및 테스트를 진행할 예정이며, XOR 인터리빙과 같은 여러 기법들을 시뮬레이터에 적용할 계획이다.

참고문헌

[1] H. Choi, J. Lee, and W. Sung, “Memory Access Pattern-Aware DRAM Performance Model for Multi-core Systems”, In Proceedings of the 2011 IEEE International Symposium on Performance Analysis of Systems & Software, ISPASS ’11, pages 66 - 75, 2011.

[2] S. P. Muralidhara, L. Subramanian, O. Mutlu,Kandemir, and T. Moscibroda. “Reducing memory interference in multicore systems via application-aware memory channel partitioning”, MICRO-44 ’11, pages 374-385, New York, NY, USA, 2011. ACM.

[3] Chi-Keung Luk, Robert Cohn, Robert Muth, Harish Patil, Artur Klauser, Geoff Lowney, Steven Wallace, Vijay Janapa Reddi, and Kim Hazelwood, “Pin: building customized program analysis tools with dynamic instrumentation”, PLDI, 2005

[4] John L. Henning, “SPEC CPU2006 benchmark descriptions”, SIGARCH Comput. Archit. News 34, 4, 2006