주문형 메모리 시스템 설계를 위한 환경 개발

이 재 혁, 박 기 호, 이 길 환, 한 탁 돈
연세대학교 컴퓨터과학과 미디어시스템 연구실

Development of Research Environment for Application Specific Memory System

Jaehyuk Lee, Gi-Ho Park, Kil-Whan Lee and Tack-Don Han
Media System Lab., Department of Computer Science, Yonsei University

요 약

미세 회로 기술의 발전은 단일 칩에 집적될 수 있는 트랜지스터의 수를 지속적으로 증가시키고 있으며 이에 따라 설계의 복잡도 역시 크게 증가하고 있다. 이러한 설계 복잡도의 증가는 여러 기능 블록이 IP(Intellectual Property) 형태로 독립적으로 설계되어서 이들의 조합으로 새로운 시스템을 구성하는 시스템 온 칩(System On a Chip)과 같은 새로운 시스템 설계 방법에 대한 요구를 증가시키고 있다[1]. 이런 시스템 온 칩에 사용될 메모리 시스템 역시 기존의 표준화된 메인 메모리 이 외에 각각의 다양한 응용에 적합한 맞춤형(Application Specific Standard Products) 내장 메모리 시스템 구조에 대한 필요성이 대두되고 있다. 이와 같이 특정 응용에 적합한 메모리 시스템을 설계하기 위해서는 해당 응용 프로그램의 메모리 참조 패턴을 분석하고 특성화하여 이에 적합한 메모리 시스템을 설계할 수 있는 기본 정보를 제공해 주는 것이 필수적이다. 또한 이러한 정보에 따라 설계된 메모리 시스템에 대한 성능 평가할 환경도 함께 요구된다. 본 연구에서는 다양한 응용의 메모리 참조 특성을 분석하고 특성화하기 위하여 캐쉬 파라메터의 변화에 따른 캐쉬 접근 실패의 분포, 메모리 접근 영역의 분포, 참조 사이에 있는 유일한 참조의 수의 분포 등 다양한 정보를 제공해 주는 환경을 구축하였다.

1. 서론

지속적으로 발전하고 있는 미세 회로 기술은 2010 년경에는 하나의 칩에 8억 개 이상의 트랜지스터를 집적할 수 있을 것으로 예상되고 있다[9]. 그리고 8억 개의 트랜지스터를 효과적으로 사용하기 위한 설계 방법에 대한 연구가 활발히 이루어지고 있다.[1] 대표적 연구로는 메모리, 입출력 제어기, 그래픽 처리기 등 다양한 기능 블록들을 하나의 칩에 집적시키려는 시스템 온 칩(System On a Chip, SOC) 기술을 들 수 있으며, 이는 각각의 기능 블록들을 IP(Intellectual Property)의 형태로 독립적으로 설계하여 나중에 이들을 조합함으로써 설계의 복잡도를 낮추고 적은 비용으로 다양한 기능의 프로세서를 설계하고자 한다. 또한 메모리 부문도 기존의 표준화된 메인 메모리가 아닌 다양한 응용에 적합한 주문형 메모리 시스템(Application Specific Memory System)에 대한 요구가 증가되었다. 이런 주문형 메모리를 설계하기 위해서는 해당 응용 프로그램의 메모리 참조 패턴을 분석해 내고 특성화한 정보가 필수적이다.

본 논문에서는 다양한 응용에 대한 메모리 참조 특성의 분석과 특성화를 통해 메모리 시스템을 설계할 수 있는 정보를 제공하며 이에 따라 설계된 메모리 시스템의 성능을 평가할 수 있는 환경을 구축하였다.

본 논문의 구성은 다음과 같다. 2장에서는 메모리 설계를 위하여 개발된 기존의 툴에 대해 살펴보고 3장에서는 본 논문에서 구축한 툴의 구성에 대해 설명한다. 4장에서는 툴이 제공하는 다양한 정보와 구현상의 고려 사항을 살펴본다. 5장에는 결론과 앞으로의 연구 방향에 대해서 살펴보겠다.

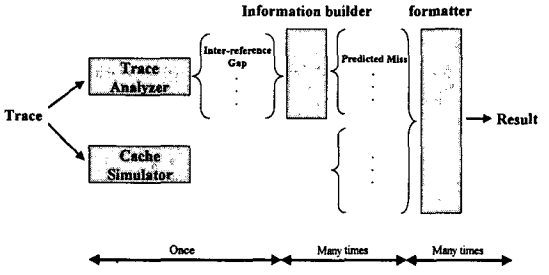
2. 관련 연구

프로세서와 메모리의 성능 격차를 효과적으로 줄이기 위한 캐쉬 메모리 시스템에 대한 많은 연구가 수행되어 왔다. 이러한 연구들에는 효율적인 캐쉬 메모리 시스템 구조에 대한 연구와 함께 응용 프로그램의 메모리 참조 특성을 분석하는 툴에 대한 연구 등이 있었다. 툴에 관한 연구 중에는 응용 프로그램의 지역성을 분석하기 위한 Cprof[3], Memspy[4], MTOOL[5] 등이 있다. Cprof[3]는 프로그램이 보다 많은 지역성(Locality)를 가지도록 수정할 수 있는 정보를 제공하며, MTOOL[5]은 병렬 프로그램까지 다룰 수 있는 장점을 가지고 있으며, Memspy[4]는 데이터 객체와 캐쉬 사이의 관계를 보여 줄 수 있는 기능을 가지고 있다. 위의 툴들이 UNIX를 기반으로 동작하는 반면 최근에 개발된 Etch[7]나 PatchWrx[8]는 Windows NT 환경에서 동작하도록 개발되었다. CVT[6]는 현재 캐쉬의 상태를 그래픽한 인터페이스를 통해 보여 줄 수 있고 SPLAT[2]는 프로그램이 가지고 있는 데이터 지역성(Data Locality)을 컴파일시의 정보와 간단한 프로파일링(Profiling)으로 빠르게 계산을 해 준다. 그러나 이런 툴들은 주문형 메모리 시스템을 위해 다양한 기본 정보를 제공하지는 못하며, 각각의 툴들은 입력과 출력 방식이 다르기 때문에 하나의 툴에서 나온 정보를 다른 툴에서는 이용할 수 없다는 단점이 있다.

3. 툴의 구성

캐쉬를 디자인하기 위해 필요한 여러 인자들의 값을 구하기 위해서는 매우 긴 시간이 필요하다. 사용하기에 편리한 툴이 되기 위해서는 결과를 얻어내기까지 너무 많은 시간이 걸리면 안 된다. 시간을 줄이기 위해 본 툴은 여러 단계를 거쳐 필요

한 정보를 얻어낸다. 앞 단계에서는 다음 단계에서 필요한 정보만을 제공하고 다음 단계에서는 앞의 정보를 이용해서 여러 번 원하는 인자를 바꾸어가며 값을 계산해 낼 수 있다. 툴의 구성은 그림 1과 같다.



[그림1] 툴 구성도.

3.1 트레이스 분석기와 캐쉬 시뮬레이터

트레이스 분석기(Trace Analyzer)는 트레이스를 입력으로 받아 분석을 한다. 이 부분은 시간이 많이 걸리는 부분이다. 여기서는 다음 단계인 정보 생성기(Information builder)에서 필요한 기본 데이터를 생성한다. 예를 들면 각 참조간의 유일한 참조의 수를 뽑아 내서 저장한다. 여기서 구한 참조 수에 대한 분석 결과는 정보 생성기에 의해서 처리되어서 각 캐쉬 사양에 따라서 미스를 예측할 수 있다. 이 과정은 캐쉬의 사양을 변화시키며 여러 번 반복할 수 있다. 이와는 조금 다르게 캐쉬 시뮬레이터(Cache Simulator)는 트레이스를 통해 실제 캐쉬를 시뮬레이션하고 그 결과는 정보 생성기를 거치지 않고 바로 형식변환기(Formatter)에 의해서 처리된다. 트레이스 분석기와 마찬가지로 캐쉬 시뮬레이터는 한 번만 이루어진다. 형식변환기는 앞에서 구한 결과를 다양한 형식으로 바꾸어 보여 준다.

3.2 정보 생성기

트레이스 분석기(Trace analyzer)에서 나온 데이터를 이용해서 실제 정보를 구성한다. 인자를 다양하게 변화시키면서 여러 번 반복 수행 할 수가 있다. 예를 들어 참조 정보를 이용해서 캐쉬 블록 크기와 캐쉬 크기 그리고 연관도를 변화시키면서 미스를 예측해 볼 수 있다. 이 과정은 단순한 계산 과정이 주를 이루기 때문에 빠르게 이루어 질 수 있다. 따라서 여러 번 반복을 해도 시간적인 측면에서 큰 부담이 되지는 않는다.

3.3 형식 변환기

정보 생성기나 캐쉬 시뮬레이터에 의해 구성된 데이터에서 원하는 정보만을 얻어 온다. 방대한 양의 데이터를 다루기 때문에 원하는 부분만을 빠르게 선택하는 메커니즘을 제공한다. 원하는 정보에 대한 그래프를 그리기 위해 x축과 y축에 사용할 변수를 정해서 그 정보만을 얻어 올 수 있다.

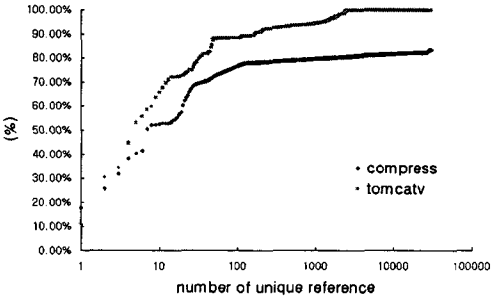
4. 툴이 제공하는 정보 및 구현 시 고려 사항

본 장에서는 툴이 제공하는 정보 및 구현 할 때 고려 사항에 대해 살펴본다.

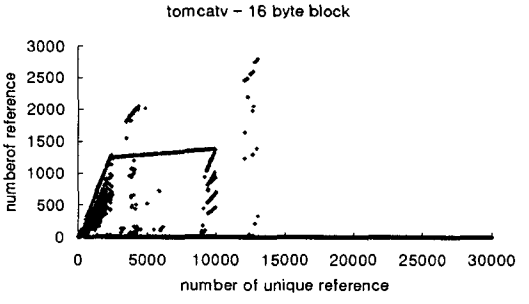
4.1 구현시의 고려 사항

그림 2는 tomcatv, compress의 트레이스 분석의 한 부분을 보여 준다. x축은 같은 참조 사이의 유일한 참조의 수를 나타내고 y축은 전체 트레이스에서 차지하는 부분을 누적해서 보여주고 있다. x축의 값이 100일 때 compress의 경우 78%, tomcatv의 경우 89%가 된다. 이것은 compress의 모든 메모리 참조 중에서 78%는 다른 주소의 참조가 100번 참조되기 전에 다시 그 주소가 참조된다는 것을 나타낸다. 마찬가지로 tomcatv의 경우에는 89%가 다시 참조된다. 이 정보를 이용해서 부분적으로 시간적 지역성(Temporal Locality)를 유추해 볼 수 있다.

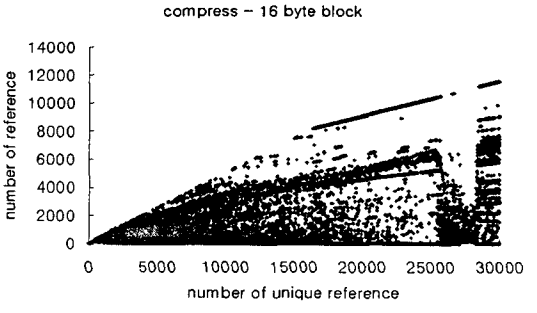
다음 그림 3은 전체 참조(트레이스)를 같은 주소가 다시 참조되었을 때 그 사이의 유일한 참조의 개수로 분류를 해서, 그 개수에 따라 전체 트레이스를 구분 지어서 각각의 수에서 평균적으로 존재하는 유일한 블록의 수를 블록의 크기를 변화시키면서 보여준 것이다. 그림 3(a)은 tomcatv에 대한 것이고



[그림2] 같은 참조 사이에 존재하는 유일한 참조 수의 분포



(a) Tomcatv



(b) Compress

[그림 3] 동일한 메모리 참조 사이의 유일한(unique) 참조의 수

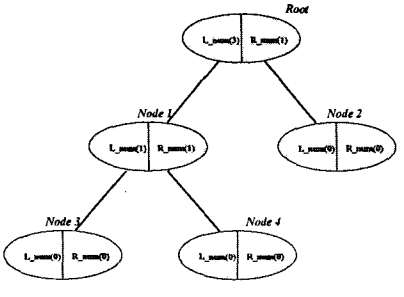
그림 3(b)는 compress에 대한 것이다. tomcatv와 compress는 확연히 다른 특성을 보여 주고 있음을 볼 수가 있다. tomcatv는 x축의 변화에 따라서 y축의 값이 한 곳으로 모이는 경향이

두르러진다. 크게 직선을 이루는 것을 볼 수가 있다. 이것은 tomcatv가 배열과 같은 큰 데이터 객체를 반복적으로 접근하는 루프를 가지고 있음을 알려 준다. 서로 구별되는 유일한 참조의 수가 증가하면서 그 안에 포함되는 참조의 수가 커지지만 전체 배열을 포함하지는 못하기 때문에 그래프에서 직선의 형태로 나타나게 된다. 반면에 compress의 경우는 x축의 증가에 따라 y값이 넓게 퍼져서 존재함을 알 수가 있다. 중간에 보이는 직선은 compress도 큰 배열을 접근하는 루프를 가지고 있다는 것을 나타낸다. 그렇지만 compress는 직선들 사이에 넓게 퍼져 있는 값들이 많다. 이것은 큰 배열 외에도 개별적으로 접근되는 많은 데이터 객체가 있다는 것을 암시한다.

4.2 구현시의 고려 사항

참조간의 유일한 참조의 수를 구하는 것은 CPU에 많은 부하를 준다. 1억 개의 길이를 가진 프로그램의 트레이스에서 위의 정보를 구성하기 위해서는 Sun UltraSparc의 경우 13-15시간이 걸린다. 많은 프로그램에 대해서 정보를 얻어내기 위해서는 상당한 정도의 시간이 요구된다. 이번 장에서는 시간의 순서대로 발생하는 이벤트의 열에서 같은 이벤트 사이에 존재하는 서로 구별되는 이벤트의 수를 구하기 위한 보다 효율적인 데이터 구조와 알고리즘에 대해서 살펴보고자 한다.

같은 주소의 참조 사이에 있는 유일한 참조의 수를 세기 위해서는 전에 그 주소를 참조했는지를 구별하기 위해서 누적해서 저장을 하고 검색을 해야 한다. 또한 그 사이의 유일한 참조의 수를 계산하기 위해서는 각 참조마다 순서를 저장하고 그 순서에 따른 순번을 얻어 낼 수 있어야 한다. 가장 많은 시간을 요구하는 부분은 각 참조마다 이전에 참조된 주소인지를 검색하는 부분과 순번을 알아내는 부분이다. 전자의 경우는 해쉬 테이블을 사용하면 O(1)의 시간에 해결할 수 있지만 후자를 함께 처리하는 것은 매우 힘든 일이다. 해쉬 테이블에 순차적인 순서를 유지하는 링크를 첨가해서 유지하는 방법이 있지만 이것은 Working Set의 크기를 N이라고 할 때 O(N)의 시간이 걸리고 N이 커지면 상당한 정도의 시간이 걸리게 된다. compress의 경우 10개의 노드를 사용했을 때 전체 트레이스의 90% 정도만이 그 안에 들어오게 된다. 여기서는 이 문제를 해결하기 위해 O(logN)의 시간에 문제를 해결할 수 방법에 대해서 살펴보겠다.



[그림4] 데이터 구조

기본적인 구조는 AVL 트리를 따른다. 그림 4에서 볼 수 있듯이 AVL 트리의 기본 노드에 L_num과 R_num 필드를 추가하였다. 이 필드들은 각각 왼쪽과 오른쪽 자식 트리에 있는 전체 노드의 수를 나타내게 된다. 노드를 추가하거나 삭제할 때 이루어지는 일은 AVL 트리와 같다. 노드의 추가 또는 삭제 후 로테이션이 일어나면 변형된 트리 구조에 맞게 위 두 필드를 갱신시켜주면 된다. 그림 5는 트리에서 순차적인 순번을 얻어내는 C로 작성된 pseudo code이다. 동작 원리를 이해하기 위해서 그림 4를 참고해서 get_rorder(Node4)의 값에 대해서 살펴보자. 그림 4에서 알 수 있듯이 그 값은 2이다. 알고리즘에 따라 살펴보면 Node4를 찾고 난 후에 rorder값은 0이 되고 Node1은 LEFT_LINK가 아니므로 그냥 지나치게 된다. Root 노드에 왔을 경우에는 오른쪽 자식의 노드 수 1과 자신 1를 더해 rorder값은 2가 되게 된다.

```
int
get_rorder(node* nd) {
   node *p;
   int rorder=0;

 /* find_node() makes the link from the node p to the ROOT */
   p = find_node(nd);
   rorder = p->R_num;
   p = next_link();
   while( p != NULL ) {
     if( FOLLOW_LEFT_LINK() == TRUE ) {
       rorder += p->R_num + 1;
     }
   }

   return rorder;
}
```

[그림5] 순번을 구하는 알고리즘

5. 결론

미세 회로 집적 기술의 발전으로 SOC(System On a chip)이 가능해짐에 따라 메모리 시스템의 설계에도 변화가 요구된다. 본 논문에서는 다양한 응용에 대해서 메모리의 참조 특성을 분석하고 특성화해서 메모리 시스템을 설계할 수 있는 정보를 얻고 그 정보에 의해서 설계된 메모리 시스템의 성능을 평가하는 환경을 구축하였다. 향후에는 Java나 3D 응용에 대해서 실제적으로 적용을 함으로써 부가적으로 환경을 개선해 나가는 것이 필요하다.

참고 문헌

[1] Trevor N. Mudge, "Strategic Directions in Computer Architecture," *ACM Computing Surveys*, Vol 28, No. 4, Dec. 1996, pp. 671-678.

[2] F. Jesus Sanchez and Antonio Gonzalez, "Fast, Accurate and Flexible Data Locality Analysis," *Int'l. Conf on Parallel Architecture and Compiler Techniques(PACT'98)*, Oct. 1998.

[3] Alvin R. Lebeck and David A. Wood, "Cache Profiling and the SPEC Benchmarks: A Case Study," *IEEE Computer*, Vol. 27, No. 10, Oct. 1994. pp. 15-26.

[4] Gupta, M. Martonosi and T. Anderson, "MemSpy: Analyzing Memory System Bottlenecks in Programs," *Performance Evaluation Rev.*, Vol. 20, No. 1, June 1992, pp. 1-2.

[5] A.J. Goldberg and J. Hennessy, "Performance Debugging Shared-Memory Multiprocessor Programs with MTOOL," *Supercomputing 91*, 1991, pp. 480-490.

[6] Eric van der Deijl, Gerco Kanbier, Oliver Temam and Elena D. Granston, "A Cache Visualization Tool," *IEEE Computer*, Vol. 30, No. 7, July 1997, pp. 71-78.

[7] T. Romer, G. Voelker, D. Lee, A. Wolman, W. Wong, H. Levy and B. Bershad, "Instrumentation and Optimization of Win32/Intel Executables Using Etch," *USENIX Windows NT Workshop*, August 1997, pp. 1-8.

[8] Jsaon Casmira, David Kaeli, and David Hunter, "Tracing and Characterization of NT-based System Workloads," First Workshop on Computer Architecture Evaluation Using Commercial Workloads, Feb. 1998.

[9] Semiconductor Industry Association. *The National Technology Roadmap for Semiconductors*. San Jose, CA, 1994