캐시메모리

한국전자통신연구원
컴퓨터 · 소프트웨어기술연구소
기 안 도

제1절 개요

컴퓨터의 기본 구성은 프로세서와 메모리와 입출력 장치 그리고 이들을 연결하는 버스라 할 수 있을 것이다. 프로세서가 연산을 할때 필요한 정보(데이터)를 메모리로부터 읽거나 쓰기를 하며 이때 데이터는 버스를 통해 전달된다. 따라서 컴퓨터의 성능은 얼마나 빨리 연산을 하느냐에 달려있고, 이는 프로세서가 얼마나 빠르게 필요한 데이터를 공급 받느냐에 달려 있다 하겠다.

요즘의 컴퓨터들은 상용 마이크로 프로세서를 사용하고, 메인메모리에는 DRAM(Dynamic Random Access Memory)을 사용한다. 최근까지의 마이크로프로세서를 빠르게 그리고 동시에 여러 명령어들을 수행하는 것에, DRAM은 용량 증가에 기술적인 발전이 집중되었다. 이러한 기술적인 발전의 차이로 인해 프로세서의 속도에 비해 메모리의 속도는 상당히 느린 편이다.

예로써, 최근의 마이크로 프로세서들은 보통 200~400MHz로 동작하고 빠르게는 1GHz까지 동작하는 것도 있지만, 빠른 DRAM의 일종인 SDRAM(Synchronous DRAM)은 통상 100MHz로 동작하는 것이 고작이다.

비록 100MHz로 동작하는 SDRAM이라도 처음어드레스가 주어지고 그에 해당되는 데이터가 출력되기까지는 통상의 DRAM과 동일한 50~60nsec의 시간이 소요된다. 따라서 대단히 빠르게 동작하는 마이크로 프로세서가 그 만큼 자주 그리고 많은 데이터를 요청할때 필요한 데이터가 제때 공급되지 못하면 프로세서 입장에서 상당한 시간을 기다리게 될 것이다.

예로써 200MHz로 동작하는 마이크로 프로세서와 응답지연이 50nsec(50 x 10 ^ (-9초))인 보통의 DRAM을 사용한 메인메모리로 구성된 간단한 컴퓨터에서 프로세서가 매클럭마다 한 개 명령어를 수행하고 평균적으로 10개 명령어중 한 개가 메모리를 참조한다고 가정해보자.

이러한 컴퓨터에서 어떤 프로그램이 실행되어 1시간만에 끝났다면 이중 30분 이상을 프로세서는 단순히 끝났다면 이중 30분 이상을 프로세서는 단순히 메모리에서 데이터가 오길 기다리는데 사용한 꼴이 된다. 즉 반이상을 빈둥거리고 논 것이

나 마찬가지란 뜻이다.

보통의 동작환경에서 고성능 마이크로 프로세서는 매클럭마다 한 개 이상의 명령어를 수행하고 이 명령어 중 10% 이상이 메모리를 참조하므로 어떻게 보면 프로그램 수행의 대부분을 그냥 기다리는데 사용하게 될 수도 있다는 것이다.

이러한 문제를 풀기위해 가장 광범위하게 사용되고 있는 것으로 캐시(Cache)가 있다. 이는 프로세서 속도에 준하는 응답속도로 동작하고, 프로세서와 메인메모리 사이에 위치하는 고성능 메모리다. 캐시는 고속응답속도를 유지해야되고, 고가이기 때문에 그 용량을 제한 받는데, 최근의 컴퓨터들은 수메가 바이트 크기까지 사용하기도 한다.

자주 사용하는 데이터를 캐시에 가져다 놓고 프로세서가 메인메모리를 직접 참조하는 빈도수를 줄임으로써 프로세서가 데이타를 기다리는 시간을 줄인다는 것이 그 동작 원리다. 우리의 일상생활에서 비슷한 예를 찾을 수 있다.

도서관에서 신간 잡지와 자주 찾는 책을 이용자가 쉽게 빨리 찾을 수 있도록 개가식 열람실에 비치하고 그렇지 않은 것들은 지하창고에 보관하여 찾는 사람이 있으면 가져와서 대출하는 방식으로 운영하는 것이 한예가 될 수 있을 것이다.

이때 개가식 열람실은 캐시에 지하창고는 메인메모리에 해당한다 하겠다.

제2절 캐시메모리의 동작과 구성

캐시메모리가 제 몫을 할 수 있는 것은 프로그램의 특정부분이 수행되는 동안 프로그램은 데이터 전체를 참조하기 보다는 이보다 훨씬 작은 부분을 참조하는 경향이 있기 때문이며 이러한 성질은 다음 두가지로 분류된다.

- 지역적 집약성(spatial locality): 프로그램이 수행중 어떤 데이터를 참조한 후 이것과 인접한 데이터를 참조하는 경향이 있다.
- 시간적 집약성(temporal locality) : 프로그램이 수행중 어떤 데이터를 참조한 후 가까운 장래에 같은 데이터를 또 참조하는 경향이 있다.

따라서 프로그램이 참조하는 부분을 고속 캐시메모리에 유지시킨다면 프로그램이 보기에 마치 전체데이터 영역이 고속으로 참조되는 것과 같아진다.

즉 프로세서가 어떤 데이터를 참조할때 해당 데이터 뿐아니라 그 근처의 데이터도 같이 캐시메모리에 가져다 놓음으로써 한번의 메모리 참조로 프로세서가 요청하는 여러번의 참조를 만족시키고, 아울러 캐시메모리에 가져다 놓은 데이터를 가능한 오래 유지함으로써 같은 데이터를 또 참조할때 메인메모리 참조없이 프로세서가 요청하는 메모리 참조중 90~95% 이상을 만족시키는 것으로 보고 되고 있다.

앞에서 설명한 가상 컴퓨터에 캐시메모리를 사용하고 이 캐시가 메모리 요청중 90%를 만족시킨다면 1시간 걸리던 프로그램이 30분 정도 걸리게 되고 이중 약 10% 정도만 메모리 응답을 기다리는데 사용된다.

즉 프로그램 실행 속도가 두배 정도 빨라진 것과 같아지고 아울러 프로세서를 사용한 효율도 높아지게 된다. 캐시메모리와 통상의 메모리는 정보를 저장한다는 공통점외에는 그 구성부터 다르다. 통상의 메모리는 인가되는 어드레스를 디코딩하여 특정 기억셀을 선택하여 그곳의 내용을 읽거나 쓰는 형태로 구성된다. 즉 특정 기억셀은 반드시 한개 어드레스만이 할당된다. 반면 캐시메모리는 (그림 1)에서와 같이 크게 두개의 메모리로 구성된다.

한쪽은 태그메모리로써 어드레스의 일부를 다른 한쪽은 데이터 메모리로써 실제 데이터를 저장한다.

어드레스(A)가 인가되면 그 일부(B)로 태그메모리에 저장된 정보(어드레스 D)를 읽어 이것을 나머지 어드레스(C)와 비교하여 같은지를 본다.

만약 C와 D가 같다면 데이터메모리에 저장된 데이터가 인가된 다른 어드레스(A)에 해당하는 것이고 그렇지 않다면 다른 어드레스에 해당하는 것이 된다.

따라서 캐시메모리의 구성은 동일한 기억셀에 특정어드레스가

아닌 여러 어드레스의 데이터를 저장할 수 있게 한다.

이러한 특성은 용량이 작은 캐시메모리가 이보다 훨씬 용량이 큰 메모리의 모든 내용을 자유롭게 저장 가능하게 하는 것이다. 통상 캐시메모리를 구성하느데는 비교기가 포함된 상용 태그메모리를 사용하고 데이터메모리는 고속 SRAM(Static Random Access Memory)을 사용한다. 캐시메모리의 크기는 항상 메인메모리보다 작으므로 어떤 데이터가 캐시메모리에 저장되려면 우선 빈 방을 만들어야 된다. 빈 방을 만드는 방법에 따라 캐시를 구분할 수 있다.

즉 주소에의해 특정 한 개 방이 고정적으로 지정되는 방법(direct map)과 주소에의해 두 개이상 방 중 선택할 수 있는 방법(set associative)그리고 캐시메모리내 모든 방 중 아무 방이나 선택하는 방법(full associative)이 있다. 당연히 후자로 갈수록 성능향상에 도움이 된다.

그 이유는 여러개중 프로세서가 사용할 확율이 적은 쪽을 선택할 수 있기 때문이다. 프로세서가 메인메모리를 참조하는 경우는 크게 명령어 참조와 데이터 참조 두가지 경우가 있다.

특히 명령어 참조는 읽기만 수행하고, 데이터 참조는 읽기와 쓰기 모두 수행한다. 이러한 이유로 프로세서용 캐시는 명령어 캐시와 데이터 캐시로 구분된 두 가지 캐시를 갖기도 한다. 또는 이들 두 참조가 동일한 캐시메모리를 공유하도록 구성하기도 한다.

일반적으로 프로세서 내부에 있는 일차캐시는 분리형이고, 프로세서 외부에 있는 캐시들은 공유형이다. 캐시가 쓰기를 어떻게 처리하느냐에 따라 그게 두가지로 구분한다.

첫째, 항상쓰기(write-through)방식은 캐시에 해당 데이터가 있든 없든 항상 메인메모리에 쓰기를 하는 방식이다.

둘째, (write-back) 방식은 메인메모리에 쓰기를 하는 대신 캐시에만 쓰기를 하고 나중에 꼭 필요한 경우에 한해 메인메모리에 쓰기결과를 반영하는 방식이다.

후자가 전자에 비해 좋은 성능을 가능하게 하는데 이는 동일 주소에 대한 여러번의 쓰기가 메모리 참조 없이 캐시에서 만족될 수 있고 덤으로 그 주소에 대한 읽기 또한 만족시킬 수 있기 때문이다.

캐시메모리의 가장 큰 장점은 하드웨어적인 구성만으로 프로그램이 실행되는 환경을 캐시 유무와 전혀 무관하게 보장하며, 캐시가 있으므로해서 전체 프로그램 수행 환경이 마치 아주 빠른 것처럼 보여준다는 것이다.

따라서 일반 사용자는 캐시가 있으나 없으나 사용환경은 동일하게 된다.

제3절 다단캐시구성

고속 프로세서의 동작속도에 부응하도록 충분히 빠른 캐시메모리를 구성하는 것이 불가능하진 않지만 용량과 속도를 모두 만족시키는데는 어려움이 있어 캐시메모리를 다단으로 구성하는 것이 일반적이다.

즉 프로세서와 동일한 속도로 동작하는 비교적 용량이 작은 일차캐시와 속도면에서는 조금 느리지만 용량이 큰 이차캐시로 구성하는 것이다.
(그림 2)에 여러가지 캐시 구성의 예를 보였다.

우리주위에서 흔히 볼 수 있는 개인용 컴퓨터에 내장되어 있는 Pentium 계열의 마이크로 프로세서의 캐시 크기는 〈표1〉와 같다. 대체적인 추세가 마이크로 프로세서 내부에 일차와 이차 캐시를 내장하는 것과 크기가 점점 증가한다는 것을 알 수 있다.

과거 컴퓨터가 사용한 메인메모리가 수KByte였던 것을 감안한다면 요즘음의 마이크로 프로세서가 사용하는 캐시메모리는 대단히 큰 용량이라 하겠다. 캐시메모리의 효용은 단일 프로세서 컴퓨터에서 보다 다중프로세서 컴퓨터에서 더욱 크다.

즉 다중프로세서 컴퓨터에서 캐시메모리는 메모리 참조 시간을 효과적으로 줄이면서 동시에 메모리를 참조하는 횟수를 줄여 네트워크 부하를 감소시키는데도

기여하기 때문이다.

하지만 다중프로세서에서 사용되는 캐시는 단일 프로세서 캐시에서는 없었던 캐시 데이터 동일성 유지 문제를 해결해야되는 부담이 있다. 이것은 동일한 데이터가 한 개이상 서로다른 프로세서 캐시에 존재가능하므로 한 쪽 복사본이 항상 최근 데이터로 유지될 수 있도록 해야 하기 때문이다.

즉 같은 데이터(같은 주소 할당된 정보)에 대하 여러 복사본이 여러 캐시에 존재할 수 있게 되므로 만약 각 프로세서들이 자신의 전용캐시에 있는 복사본에 대하여 자유롭게 변경(쓰기)을 시도하게 되면 동일 메모리 주소에 대한 동일하지 못한 상태를 야기하게 되어 결국 프로그램 수행의 결과가 오류를 범하게 된다. 캐시의 일관성을 깨뜨릴 수 있는 요인에는 쓸수 있는 데이터의 공유(sharing writable data), 프로세스 이주(process migration), 그리고 입출력 동작(input output activity) 등이 있다.

일반적으로, 프로세서의 임의의 주소에 대한 읽기의 복귀값(return value)이 같은 주소에 대한 가장 최근의 쓰기 값과 같음을 보장 할 때 캐시가 일관성을 유지한다고 정의한다.

이러한 것을 위해 캐시 일관성/동일성 유지 프로토콜(cache coherency protocal)이 필요하고 이 프로토콜이 메모리에 대한 일관된 상태를 보장하여 프로그램이 잘 수행됨을 보장하게 된다. 캐시동일성을 유지하기 위해서는 읽고 쓰는 명령 외에 캐시간의 동일성 유지명령(consistency command)을 필요로 한다. 예로써, 한 프로세서의 캐시에 있는 데이터를 다른 캐시에서 요구할 경우 이것이 공유되고 있음을 알리는 명령이 필요하다. 또다른 예로써, 특정 프로세서가 여러 캐시에서 공유되고 있는 데이터에 대해 배타적인 참조를 필요로 할 때 다른 캐시에 있는 복사본을 무효화 시켜야 하므로 무효화 명령이 필요하다. 캐시 동일성 유지 명령들은 동시에 여러 캐시에 영향을 주어야 하므로 프로세서들 사이의 상호연결망에 상당한 통신부담을 야기하게 된다.

상호연결망이 단순한 형태인 버스(bus)인 경우, 또는 프로세서의 수가 적은 경우 이러한 동일성 유지에 필요한 통신부담은 어느 정도 감수할 수 있겠으나 프로세서 수가 상당히 많아지는 경우 단순한 상호연결망으로는 전체적인 통신을 감당할 수 없게 되며, 더구나 복잡한 상호연결망에서는 동시에 여러 캐시에 영향을 주기란 어려운 일이다.

이러한 이유로 대규모 다중프로세서에서는 캐시에 저장되는 데이터별로 일정한 정보를 유지함으로써 선택적으로 캐시 동일성 유지 동작을 가능하게 하는 디렉토리 방법(directory method)이 사용되고 있다. 버스에 기반한 다중 캐시 시스템에서는 Encore Multimax, Sequent Symmetry, Pyramid MIServer, 국산 주전산기 TICOM-Ⅱ와 Ⅲ, SGI Challenge, Sun Enterprise 등이 있다. 디렉토리 방법은 2~4개 프로세서로 구성되는 작은 규모의 버스 기반 시스템을 상호연결망으로 묶는 시스템에서 일반적으로 많이 사용되고 있다. 그 대표적인 예가 CC-NUMA(Cache Coherent Non-Unifom Memory Access) 컴퓨터로써 SGI Origin 2000, Sequent NUMA-Q, Data General AViion 20000, HAL S-1 등이 있다.

CC-NUMA 컴퓨터들의 삼차캐시는 그 크기가 보통 128MByte 이상이거나 HAL

표1 : Intel Pentium Processor의 캐시 비교

	CPU 속도	일차캐시		이차캐시
		명령어캐시	데이터캐시	
Pentium	60~200MHz	8KB	8KB	없음
Pentium Pro	150~200MHz	8KB	8KB	256KB, 512KB, 1MB
Pentium MMX	133~266MHz	16KB	16KB	없음
Pentium Ⅱ	233~400MHz	16KB	16KB	512KB, 1MB, 2MB

S-1과 같이 삼차 캐시를 사용 않는 경우도 있다.

제4절 결론

날로 빨라지고 있는 마이크로프로세서와 점점 증가하는 메모리 용량 사이의 불균형중 속도차이에서 오는 부분은 캐시메모리가 해결하고 있다.

비록 대부분의 상용 마이크로프로세서가 내부에 캐시메모리를 내장하고 있으나 충분하지 못한 경우가 있고, 또 여러 프로세서를 사용해야될 경우 성능 향상을 위해 외부 캐시메모리를 사용하는 것이 일반적이다.

향후 마이크로프로세서를 채용하는 시스템에서는 캐시메모리가 필연적으로 사용될 것이며, 비록 일반 사용자는 이들의 존재와 동작에 대한 지식 없이 사용하겠지만, 시스템을 개발하거나 성능에 관심이 있는 분야에서는 반드시 캐시메모리에 대한 분석이 필요하다.

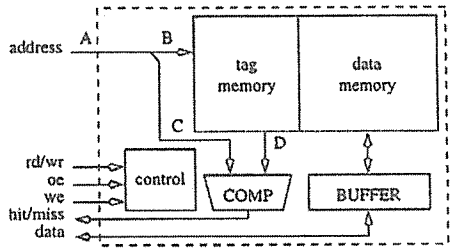
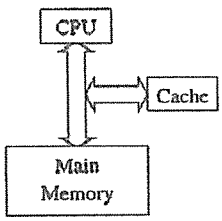
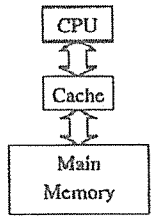


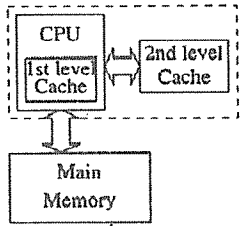
그림 1. 캐시메모리 구성의 일례



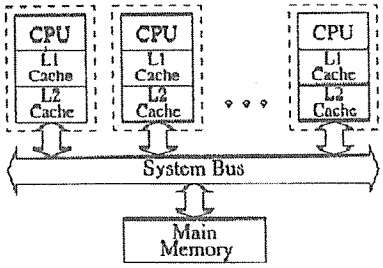
(a) Look-aside 캐시구성



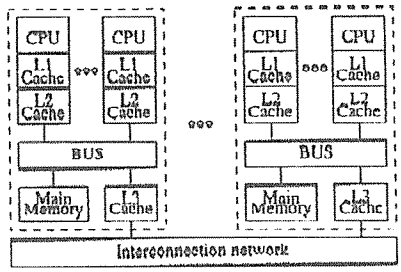
(b) Look-through 캐시구성



(c) Pentium® Pro프로세서의 2단 캐시구성



(d) 버스에 기반한 다중캐시 구성



(e) CC-NUMA캐시구성