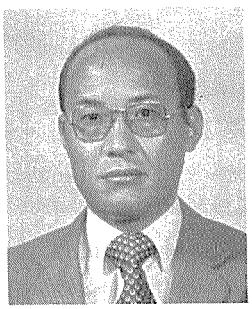
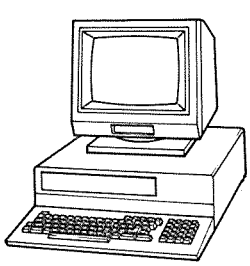


吳 吉 祿
韓國電子通信研究所
주전산기개발본부장 / 工博

다중 프로세서 시스템용 캐쉬 메모리 설계에 관한 연구



1. 개 요

Tightly Coupled Multiprocessor Computer System (TCMCS)에서 캐쉬 메모리를 사용하는 목적은 크게 두가지로 생각할 수 있는데, 첫째 프로세서의 평균 메모리 접근 시간을 줄이는 것과 둘째 프로세서와 메모리 사이의 데이터 전송량을 줄이는 것이 있다. 특히 버스를 이용하여 메모리와 프로세서들을 연결하는 TC-MCS의 경우, 버스는 시스템 내에서 가장 중요한 자원이 되고 버스의 데이터 전송 속도가 시스템의 성능에 좌우하게 됨으로 캐쉬 메모리는 버스의 부담을 줄이는 목적이 더욱 강조되고 있다.

한셈틀 하드웨어 version. 1 (HW. 1) 〔1〕은 32비트 마이크로 프로세서 (MC68020 〔8〕)와 VMEbus 〔9〕 등 표준부품 (off-the-shelf component)을 이용한 TC-MCS이다. 그림 1은 프로세서와 메모리를 중심으로 나타낸 블럭구조로서 각 프로세서 보드 마다 캐쉬 메모리를 갖고 있다.

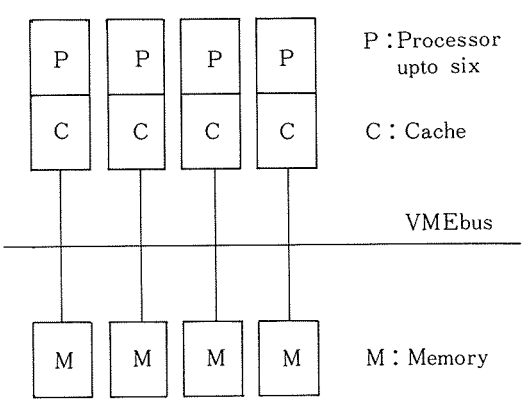


그림 1. 한셈틀 하드웨어 V. 1의 프로세서 및 메모리 구조

본고에서는 그림 1과 같은 캐쉬 메모리를 설계할 때 고려해야할 설계 기준과 그 기준에 따른 설계과정에 대해 기술하며, 설계한 캐쉬 메

모리의 성능을 버스 이용률과 평균 접근 시간 측면에서 평가한다. 2장에서는 HW. 1의 캐쉬 메모리의 설계 기준에 대해 설명하며, 기준에 의해 설계한 캐쉬 메모리의 형태별 사양에 대해서 기술한다. 3장에서는 성능에 절대적인 영향을 미치는 설계 변수의 결정과정을 성능평가와 함께 기술한다. 끝으로 내용의 요약과 문제점 그리고 앞으로의 연구방향에 대해서 언급한다.

2. 사양 및 설계 기준

표1은 HW. 1의 캐쉬 메모리 사양을 요약한 것으로 그와 같은 결정을 하는데 고려한 설계 목표는 가장 빠르게 그리고 가장 적게 버스를 사용하여 프로세서가 원하는 데이터를 제공하는 캐쉬 메모리를 설계하는 것이다. 물론 위와 같은 설계 목표에는 현재의 구현 기술과 비용 그리고 주변 회로와의 호환성 등의 제약점이 따른다.

표1. HW. 1의 cache사양 요약

Data Memory Size	8 Kbytes
Block Size	16Bytes
Tag Organization	Directed mapping
Tag Memory Size	Dual 512×16
Hit ratio	77percent
Hit cost	133nsec
Miss cost	975nsec
Write cost	267nsec
Cache type	Write-through cache No separate cache Physical address cache
Consistency protocol	Bus monitoring
Error Protection	Tag memory parity check
Implementation Technology	CMOS Static RAM+TTL

HW. 1 캐쉬 메모리를 설계하는데 있어서의 기준(design criteria)은 다음과 같다.

○접근 시간의 최소화
○hit ratio의 최대화
○miss에 따른 지연 시간의 최소화
○캐쉬 메모리 consistency overhead의 최소화
○구현 난이도 및 비용의 최소화
○주변 회로와의 호환성

앞의 설계 기준 중 앞의 3가지는 종래의 캐쉬 메모리에서도 강조되었던 것 들이며, 네째는 다중 프로세서 시스템이기 때문에 필요한 기준이 된다. 다섯째의 것은 비용과 구현 기술을 감안한 성능 향상을 고려해야함을 말하고 있고, 끝으로는 HW. 1은 표준 부품을 가능한한 많이 사용하는 것을 지향하고 있으므로 필요한 설계 기준이다.

캐쉬 메모리는 Write 주기를 처리하는 방법에 따라 크게 두가지 종류로 나뉘어지는데, Write가 발생함과 동시에 메인 메모리로 데이터를 전달하는 Wirte-thru 형태와, 쓰여지는 데이터를 캐쉬 메모리에 간직하는 Write-back (or copy-back) 형태가 있다. 전자의 것은 프로세서의 Write 주기마다 항상 메인 메모리를 사용하기 때문에 버스 사용을 줄이는데 한계가 있지만 (프로세서의 Write주기의 부분 이하로 낮출 수 없음), 캐쉬 메모리 Consistency (coherency) 문제 해결이 용이하고 구현하기 쉽기 때문에 현재 상품화된 대부분의 MCS가 채택하고 있는 형태이다. 후자의 형태는 프로세서의 Write 주기에 관계없이 캐쉬 메모리의 성능에 따라 현저히 버스 이용률을 줄일 수 있는 장점을 갖고 있지만 Consistency 문제 해결이 용이하지 않다. HW. 1의 캐쉬 메모리는 Write-thru 형태로 하는데, 그 이유는 구현이 용이하며 버스 사용 성능에 있어서도 6개의 프로세서를 지원하는 것이 목표인 HW. 1에 크게 제약을 주지 않기 때문이다.

캐쉬 메모리의 성능을 가장 크게 좌우하는 설계 변수는 데이터를 저장하는 메모리의 크기와 블럭 크기로서, 이 변수들을 정하는 것은 목표 성능을 현재의 구현기술로 맞추는 작업이다. 이때, 버스의 전송 프로토콜, 메모리의 접근시간, 현재 사용할 수 있는 메모리의 직접도 (Static RAM) 등을 고려하여 정하게 되는데, 3장에서 성능과 연결하여 결정과정을 기술한다. HW. 1 캐쉬 메모리는 전체 8 Kbyte에 16byte의 블럭 크기를 갖는다.

Write-thru 캐쉬 메모리의 Consistency 문제는 버스상의 Write 주기를 Monitor 함으로써 쉽게 해결할 수 있다. 즉, 버스상에 Write 주기가 발생하면 Bus monitor가 캐쉬 메모리에 해당 번지의 데이터를 갖고 있는가를 확인하게 된다. 그러나 이 과정에서 여러개의 프로세서를 한 버스상에 위치하게 됨으로써 Write 주기가 많이 발생하게 되어 프로세서의 데이터 처리를 위해 캐쉬 메모리 사용이 Bus monitor를 위한 캐쉬 메모리 접근에 의해 간섭을 받게 된다. 따라서 버스 위에 프로세서가 많아지면 버스 이용률의 증가에 의한 기다리는 시간(queuing delay)과 Bus monitor를 위한 지연때문에 각 프로세서는 제 성능을 발휘할 수 없게 된다. 이와 같은 문제를 해결하기 위해 Bus monitor를 위한 Tag memory와 프로세서의 데이터 요청을 처리하기 위한 Tag memory를 별도로 두는 방법을 채택한다. 두 Tag memory의 변형은 항상 동시에 일어나도록 하여 이들 사이의 동시성 문제를 해결한다.

캐쉬 메모리는 Associative메모리 (Contents Addressable Memory)의 구현으로써 Tag memory의 구조에 따라 Fully associative,Set associative 그리고 Directed mapping으로 구분된다. 첫번째 방법은 현재의 구현기술로 보아 실질적으로 구현이 불가능하고 (Translation Lookaside Buffer 등에 구현되고 있음), 나머지 두 방법이 현재 많이 사용되고 있다. Set associative 방법은 Directed mapping방법과 Fully associative 방법의 중간 형태로 몇개의 Tag memory set를 갖고 있는가에 따라 n-set associative라고 말한다. 보통 구현이 용이하기 때문에 2-set과 4-set이 많이 쓰이는데 (VAX11/780는 2-set[10]), Directed mapping 방법보다 많은 부품이 들어가는데 (같은 크기의 캐쉬 메모리를 구현했을 때 2 set associative가 Directed 보다 약 2배의 Tag memory, Comparator가 필요함), 성능에 있어서는 그 만큼의 향상을 내지 못하고 있다 [5]. HW. 1 캐쉬 메모리는 구현상의 어려움과 보드의 공간, 성능 등을 고려하여 Directed mapping 방법으로 Tag memory를 구성한다.

그 밖에 캐쉬 메모리를 설계하는데 있어서 고려하는 사항으로 캐쉬 메모리를 제어하는 어드레스에 관한 사항이다. 최근의 모든 시스템에서는 두가지의 어드레스를 갖게 되는데, 프로세서가 생성하는 가상 어드레스(Logical or Virtual)와 변환과정을 거쳐 실제 메모리의 위치를 나타내는 Physical 어드레스가 그것이다. 이 두가지 어드레스 중 어떤 것으로도 캐쉬 메모리를 구현할 수 있는데, 전자의 것은 MMU (Memory Management Unit)의 어드레스 변환이 없이 캐쉬 메모리를 사용할 수 있는 장점이 있는 대신, Synonym Problem[2]이 발생하기 때문에 이의 해결을 위하여 하드웨어의 구현이 복잡해지거나 소프트웨어에 제약이 가해지는 단점을 갖고 있다. 후자의 것은 구현상 간단하지만 매 메모리 접근때 MMU의 어드레스 변환 시간을 기다려야 하기 때문에 전자의 것 보다 접근시간이 긴 단점이 있다. 그러나 Synomym 문제의 심각성과 구현성의 난이도 등 때문에 상품화가 된 거의 모든 시스템들은 Physical 어드레스 캐쉬 메모리를 구현하고 있다(ELXSI는 소프트웨어에 제약을 가한 Virtual address 캐쉬 메모리[11]). 한셈틀 하드웨어 ver. 1의 캐쉬 메모리도 Physical 어드레스를 사용한다. 그밖에 고려한 사항은 Fetch algorithm, 인스트럭션과 데이터를 분리하는 캐쉬 메모리 형태 등 여러가지가 있는데 대부분 보편적으로 이용되고 있는 방법들을 채택한다.

3. 성능 평가

다중 프로세서 캐쉬 메모리의 성능은 평균접근시간(AAT, Average Access Time)과 버스 사용비(BTR, Bus Traffic Ratio = 캐쉬 메모리를 사용할 때의 버스 사용량/캐쉬 메모리를 사용하지 않았을 때 버스 사용량)로 나타낼 수 있다. 즉, 프로세서가 원하는 데이터를 가장 빨리, 가장 적게 시스템 버스(혹은 메인 메모리)를 사용하여 공급할 수 있는 캐쉬 메모리가 가장 우수한 성능을 갖는다고 할 수 있다. 한셈틀

하드웨어 ver. 1이 요구하는 캐쉬 메모리의 성능은 MC68020가 1.7 MIPS 이상 (6개를 꽂았을 때)으로 들 수 있도록 프로세서의 데이터 요구를 처리해야 하고, 6개의 프로세서 보드를 VMEbus에 꽂을 수 있도록 낮은 버스 이용률 (Bus Utilization = 단위 인스트럭션 수행 중 버스 사용 시간/단위 인스트럭션 수행 시간)을 나타내야 한다. 또한 6개를 연결하여 10MIPS의 성능을 보이기 위해서는 프로세서를 더 했을 때 발생하는 추가부담(하드웨어 측면의 추가 부담, 예, 캐쉬 메모리 Consistency overhead, Queuing delay, …)을 극소화해야 한다.

본장에서는 위와 같은 목표를 달성하기 위한 캐쉬 메모리의 설계 변수를 결정하는 과정과 근거에 대해 평균 접근 시간과 버스 이용률로 나누어서 기술한다. 결정과정에서 사용할 각종 데이터는 사용할 버스의 사양과 메모리 보드의 사양을 근거로 추정하였고, MC68020 인스트럭션의 특성(평균 수행시간, 크기, …), 수행시 특성(run time statistics) 등은 [4]를 참고하였다. [4]에서 제공하는 각종 데이터는 Trace-driven simulation을 이용하여 산출한 것으로 시뮬레이션과 함께 측정(Measurement) 과정을 거쳐 두개의 결과를 비교 검토한 것이다.

가. 평균 접근 시간(AAT, Average Access Time)

평균 접근 시간(AAT)은 프로세서의 데이터 요구에서부터 데이터가 프로세서에 전달될 때까지의 평균 시간으로, 이는 캐쉬 메모리를 사용하는 프로세서의 성능을 결정하게 된다. AAT는 Hit ratio(hr=1-miss ratio, 프로세서가 원하는 데이터가 캐쉬 메모리에 있을 확률), Hit cost(hc, 캐쉬 메모리에 프로세서가 원하는 데이터가 있을 때 접근시간)와 miss cost(miss가 발생했을 때 접근시간)에 의해서 결정된다. (공식 1). (공식 1)은 캐쉬 메모리의 AAT를 계산하기 위한 일반적인 식이며 이를 우리가 구현할 캐쉬 메모리(write-thrutype)에 보다 구체화시키면 (공식 2)와 같다. Write-thru 캐쉬 메모리의 경우 프로세서의 Write 주기(wf, write fraction, 프로세서의 메모리 접근중 write일 확률)는 무조건 Miss로 처리하여 버스를 사용하기 때문에, (공식 2)는 hr에 포함되지 않는 경우를 Miss와 write 부분으로 분리하여 계산한 것이다.

$$AAT = hr \times hc + (1-hr) \times mc \quad \cdots\cdots \text{(공식 1)}$$

$$AAT = hr \times hc + (1-hr-wf) \times mc + wf \times wc \quad \cdots\cdots \text{(공식 2)}$$

$$MIPS = \frac{cf \times 1.08}{acpi + (rpi \times (ATT/(cp-2))} \quad \cdots\cdots \text{(공식 3)}$$

% cf : clock frequency
% wc : write cost
% cp : clock period (cp = 1/cf)
% rpi : average memory reference per instruction
% acpi : average clocks per instruction

AAT가 결정되면 그 캐쉬 메모리를 사용하는 프로세서(MC68020)의 성능(MIPS, Million Instruction Per Second)을 (공식 3)에 의해 구할 수 있다. cf는 프로세서가 사용하는 Clock의 주파수를 나타내며, 인스트럭션 당 평균 Clock 수와 인스트럭션 당 평균 메모리 접근 회수는 [3]에 의하면 MC68020의 인스트럭션 캐쉬 메모리가 64%의 hr를 나타낼 때 7.159와 1.201이 된다. 전체의 값에 1.08을 곱한 이유는 실제 측정치와 시뮬레이션에 의한 데이터의 차이를 교정하기 위한 것이다. 측정치와 시뮬레이션에 의한 결과가 차이를 보이는 이유는 시뮬레이션에 의한 성능 평가에서는 프로세서 내부에서 일어나는 겹치기 수행(Pipelined or overlapped execution)에 의한 수행시간이 단축되는 현상을 계산할 수 없기 때문이다. 현재의 MC68020의 최대 Clock은 16.67 MHz로 (공식 2)를 역으로 계산하면, 1.7 MIPS의 성능을 얻기 위해서는 AAT가 293 nsec 보다 적어야 함을 알 수 있다.

그러나 최대 Clock speed로 돌렸을 때 어드레스의 변환 등을 고려할 때 Wait state 없이 돌리기 위한 주변회로의 구현이 어렵기 때문에 HW. 1은 Clock을 약간 늦추어 No wait로 동작할 수 있도록 16MHz로 한다. cf를 16MHz로 할

때 AAT를 계산하면 303 nsec를 얻을 수 있다. 그러면 위와 같은 AAT를 나타내기 위한 캐쉬 메모리의 설계 변수를 알아본다.

AAT는 (공식 1)과 같이 hr, hc, mc, wf, wc에 의해 계산된다. hr는 캐쉬의 크기에 가장 크게 좌우되며, 블럭 크기에도 많은 영향을 받는다. 캐쉬의 크기는 그 캐쉬 메모리가 낼 수 있는 대략 hr범위를 결정하게 되는데, 보다 자세한 것은 블럭 크기, Tag memory(TM)의 Associativity, Fetch algorithm 등에 의해 결정된다.TM의 구조는 Directed map방식, Fetch algorithm은 Demanded fetch 등 일반적으로 사용되는 간단한 방식을 채택한다. 블럭 크기는 메인 메모리와 캐쉬 메모리 사이에 전송되는 데이터의 기본단위로서, 크게 하면 Spatial locality에 의한 hr가 증가하게 되고, 일정 크기를 넘어 가면 Temporal locality에 의한 hit가 현저히 감소하게 되어 전체 hr가 감소하게 된다. 따라서 블럭 크기도 캐쉬 메모리의 전체 크기와 더불어 hr에 중요한 역할을 한다. wf(write fraction = write회수/메모리 접근 회수)은 수행할 프로그램에 따라 크게 달라질 수 있으나 여기에서 사용하는 값은 여러 프로그램을 시뮬레이션하여 계산된 평균 값인 20%로 한다. wc는 Write를 빠른 시간내에 할 수 있도록 Write buffer를 사용하는 방법과 바로 버스에 프로세서가 직접 수행하는 두가지를 고려한다.

표 2는 MC68020를 사용했을 때 (인스트럭션 캐쉬 메모리의 hr는 64%로 가정) 여러가지 캐쉬 메모리 크기와 블럭 크기에 대한 hr을 나타내고 있는 것으로 현재의 하드웨어 집적도로써 표준버스에 구현할 수 있는 크기(8K, 32K)를고려한 것이다. 또한 Tag memory는 Direct map 방식이고, Write-through 캐쉬 메모리를 대상으로 시뮬레이션했기 때문에 Write는 모두 miss로 간주한 결과이다. MC68020는 내부 인스트럭션 캐쉬 메모리를 사용함으로써 메모리 접근 Locality가 일반적인 것 보다 떨어지기 때문에, 표 2의 값은 일반적인 On-chip 캐쉬 메모리를 사용하지 않는 경우의 hr 보다 4～6% 적은 값을 나타내고 있다.

표 2. 여러가지 캐쉬/블럭 크기에 대한 Hit ratio

BS/CS	8 Kbytes	16 Kbytes	32 Kbytes
4 Bytes	65.3	70.8	74.9
8 Bytes	71.8	75.0	77.8
16Bytes	75.1	77.2	78.9
32Bytes	76.8	78.4	79.7

캐쉬 메모리의 hc는 사용할 Static RAM 의 접근시간과 Cache Controller의 형태 그리고 주변회로의 지연시간 등에 의해 결정된다. MC 68020의 버스는 Clock에 의해 동기되어 동작함으로 hc는 66.6 nsec(=1/16 MHz) 단위로 계산되어야 하고, Physical 어드레스를 사용함으로 MC68851(PMMU, Paged Memory Management Unit)의 어드레스 변환을 위한 시간도 고려해야 한다. 또한 MC68020이 가장 빠르게 데이터를 가져갈 수 있는 시간은 버스 동작의 시작 후 2 clock 후이므로 133 nsec 이하의 hc는 의미가 없다. 본 캐쉬 메모리에서는 45 nsec의 SRAM을 사용하고 hit됐을 때 No wait state로 동작시키기 위하여 캐쉬 메모리 접근 시간을 어드레스 변환 과정과 가능한한 겹쳐질 수 있도록 구현한다. 그러므로 hc는 133 nsec가 된다.

앞에서 설명한 대로 블럭 크기는 캐쉬 메모리의 기본단위로서 크게 하면 Miss가 발생했을때 여러번의 메모리 사용을 필요로 하기 때문에 mc는 증가되지만 대신 hr가 증가되어 mr가 줄어들게 된다. 블럭의 최소 크기는 프로세서와 버스의 기본 단위인 32비트(4바이트)로 생각할 수 있다(4바이트 이하는 성능의 현저한 저하를 가져옴). 블럭을 크게 하면 하나의 프로세서가 비스를 오래 잡게 되어 TC-MCS 환경에 적합치 않을 뿐 아니라 버스 이용률도 크게 증가하여 여러개의 프로세서를 지원할 수 없게 된다. 본절에서는 최소단위인 4바이트와 VME bus의 블럭전송 프로토콜과 메모리 칩의 Nibble access mode [12]를 최대한 이용할 수 있는 16바이트, 두개만을 고려한다.

mc는 메모리의 접근 시간, 버스의 Arbitration시간, 다른 보드가 비스를 사용하고 있을 때 기다리는 시간, 버스 사용시의 추가 부담(드라

이버의 on/off시간) 그리고 Miss가 발생하므로 생기는 프로세서의 지연 시간 등으로 구성된다. 메모리 보드의 기본 접근시간은 300nsec, Nibble mode접근방식으로 16바이트를 연속 사용했을 경우 500nsec의 시간이 필요하다. Arbitration을 위해 필요한 시간은 데이터의 전송과 Arbitration이 병렬로 수행이 되므로 별도로 추가될 필요가 없고, 버스 사용시에 드라이버의 on/off시간으로 약 30nsec의 여유가 필요하다. Miss가 발생하면 Cache Controller는 MC68020에게 현재의 버스 주기를 끝내고 (Bus Error 처리) Miss처리가 끝난후 재 시도(retry cycle)를 하도록 하기 때문에 재 시도에 의한 7 Clock의 시간(266nsec)이 더 필요하게 된다.

끝으로 mc에 추가되어야 할 시간은 여러개의 보드들이 하나의 시스템 버스에 물려 있기 때문에 어느 한 보드가 버스를 요청했을 때 다른 보드가 버스를 사용하고 있으므로 생기는 지연 시간이다. 이 값은 다시 버스 이용률과 서로 관계가 되므로 보다 복잡한 해석이 필요하게 되는데, 여기에서는 우선 상호 간섭이 없는 것으로 간주하여 ATT와 버스 이용률을 계산하고 후에 오차를 감안하도록 한다. 결과적으로 4바이트와 16바이트의 블럭 크기를 가질 때 mc는 각각 796nsec와 996nsec가 된다.

(식 2)에 AAT를 표 2의 각 항별로 계산한 것이 표 3이다. Write주기는 Write-thru 캐쉬 메모리를 사용하기 때문에 항상 버스를 사용하게 된다. Write buffer를 구현할 경우 프로세서의 내부 수행과 버스의 사용시간을 동시에 진행시킬 수 있기 때문에 266nsec에 Write를 수행할 수 있다. Write buffer없이 구현할 경우 메모리의 Write 주기 150nsec와 버스를 사용하기 위해 기다리는 시간 100nsec와 자체부담 3 Clock을 합하여 약 450nsec의 시간이 소요된다. 표 3의 앞에 것은 Write buffer를 구현한 것이고 뒤의 것은 사용하지 않은 것이다. 표 3을 통하여 8 Kbyte이상의 캐쉬메모리는 1.7MIPS의 성능을 나타내기 위한 AAT를 만족함을 알 수 있다. Dual Tag Memory를 구현함으로서 Consistency 문제 해결을 위한 시간적 부담을 최대한 줄일 수 있지만 시스템이 운영되고 있을 때 두개의 프로세서에 의하여 공유되고 있는 데이터가 캐쉬 메모리에 존재하면 버스상의 Write 주기는 프로세서의 캐쉬 메모리 사용을 방해하게 된다. 그외 여러가지 특수한 경우의 발생을 고려하여 표 3의 값에 약간의 오차가 발생할 수 있음을 감안해 둔다.

표 3의 결과를 종합해 보면 Write buffer 를 사용하지 않은 8K/4bytes 캐쉬 메모리 형태는 1.7MIPS의 성능을 나타내기 위한 AAT의 값을 만족하고 있고, 그외의 형태는 10%정도의 오차를 감안해도 원하는 값을 나타내고 있다. 표 4는 각 캐쉬 형태별로 MC68020의 성능을 나타내고 있다.

표 3. 여러가지 캐쉬/블럭 크기에 대한 평균 접근 시간(nsec)

BS/CS	8 Kbytes	16 Kbytes	32 Kbytes
4 Bytes	245/289	208/253	180/225
16 Bytes	189/234	171/215	156/201

표 4. 여러가지 캐쉬/블럭 크기에 대한 MC68020의 성능(MIPS)

BS/CS	8 Kbytes	16 Kbytes	32 Kbytes
4 Bytes	1.83/1.67	1.97/1.80	2.10/1.90
16 Bytes	2.06/1.87	2.15/1.94	2.23/2.00

나. 버스 이용률

앞절의 분석에 따르면 8K바이트 이상의 캐쉬 메모리를 사용하면 MC68020은 1.7MIPS 이상의 성능을 나타냄을 알 수 있었다. 이와 같은 성능은 앞에서 언급했듯이 프로세서가 메모리를 사용할 때 다른 보드의 간섭이 전혀 없다고 가정한 결과이다. 그러나 실제의 상황에서는 다른 프로세서 보드의 버스 사용과 입출력 콘트롤러의 DMA(Direct Memory Access) 동작에 의한 메인 메모리를 사용할 때 지연이 발생한다 따라서 프로세서 개수가 추가될 때 각 프로세서당 성능은 감소하게 되는데, 이 값은 지연 시간에 비례하고 지연시간은 프로세서의 버스 이용률에 따라 결정된다. 즉, 큰 버스 이용률은 상호 간섭이 증가하여 메모리를 사용할 때 마다

많은 시간을 기다리게 되어 프로세서의 성능의 저하를 초래한다. 본절에서는 이와 같이 HW. 1의 목표인 6개의 프로세서를 버스 상에 꽂았을 때 각 프로세서의 성능을 계산하기 위해 각 프로세서(캐쉬 메모리)의 버스 이용률에 대해 언급한다.

버스 이용률

$$= \frac{\text{단위 인스트럭션 수행중 버스 사용 시간}}{\text{단위 인스트럭션의 수행 시간}}$$

..... (공식 4)

$$= \frac{rpi \times mr \times bm + rpi \times wf \times bw}{\text{ait} + (rpi \times (AAT - 133))} \quad \text{.... (공식 5)}$$

% ait : average instruction execution time

% wf : write fraction

% bm : miss bus cost

% bw : write bus cost

(공식 4)는 한 프로세서의 버스 이용률을 나타내고 있다. 단위 시간 (단위 인스트럭션이 수행 시간)에 대한 버스 사용시간으로, 버스를 사용하는 경우는 캐쉬 메모리에서 Miss가 발생했을 때와 프로세서의 Writc 주기 이다. 단위 인스트럭션 수행중 평균 메모리 사용회수는 [4]에 의해 1.201 번이고 버스의 사용 시간은 앞에서 언급한 바와 같다. (공식 5)는 (공식 4)를 MC68020의 버스 동작 특성과 16 MHz의 Clock을 사용하는 것으로 가정하여 구체화한 것으로 분자의 첫째항이 Miss로 인한 버스 사용 부분, 둘째항이 Write에 의한 것이다.

분모는 단위 인스트럭션의 평균 수행 시간으로, AAT에 133 nsec를 뺀 이유는 ait안에 기본 2 clock이 포함되어 있기 때문이다. 표 5는 여러 캐쉬 형태에 대한 버스 이용률을 계산한 것으로 각 항의 앞에 값은 Wirte buffer를 사용한 경우이다. Write buffer를 사용함으로써 프로세서의 성능을 높이는 대신 시간당 버스를 많이 사용하는 것을 알 수 있다. 또한 캐쉬의 크기를 두배씩 증가시켜도 버스이용률이 현저히 감소하지 않는 것을 알 수 있는데, 이는 프로세서의 Write 주기에 해당되는 버스 사용은 Write-thru 캐쉬 메모리를 사용하기 때문에 줄일 수 없기 때문이다.

표 5의 모든 캐쉬 형태는 모두 6을 곱해도 100

표 5. 여러가지 캐쉬/블럭 크기에 대한 버스 이용률(percent)

BS/CS	8 Kbytes	16 Kbytes	32 Kbytes
4 Bytes	16.6/15.2	14.0/12.8	11.8/10.7
16 Bytes	13.8/12.6	11.7/10.8	9.9/ 9.0

을 넘지 않기 때문에, 각 프로세서의 버스 이용 시점을 적절히 제어하면 위의 어떠한 캐쉬 형태를 취하더라도 성능의 저하없이 6개의 프로세서를 VME bus에 꽂을 수 있다. 그러나 실제 시스템이 운용될 때 각 프로세서의 버스의 사용은 독립적으로 발생하기 때문에 간섭이 일어난다. 이와 같은 간섭의 정도를 알아봄으로써 프로세서가 증가할 때 각 프로세서의 성능의 저하를 예측할 수 있다.

버스 이용률을 일정한 확률 분포로 가정하고 두개 이상의 프로세서가 동시에 버스를 요청할 확률로 간섭을 계산한다. 예를 들면 세개의 프로세서 (A, B, C)를 사용한다고 가정할때, 동시에 버스를 요청할 확률은 프로세서가 동시에 버스를 사용할 경우의 (AB, BC, CA) 확률과 ${}_3C_2 \times BU^2$, 세개의 프로세서가 동시에 프로세서를 사용할 경우 (ABC) 확률 ${}_3C_3 \times BU^3$의 합이 된다. N개의 프로세서를 사용할 경우는 (공식 6)과 같다.

두 개 이상의 프로세서가 동시에 버스를 사용할 확률

$$= {}_nC_2 \times BU^2 + \cdots + {}_nC_n \times BU^n \cdots\cdots \text{(공식 6)}$$

한 프로세서가 버스를 사용할 때 간섭받을 확률

$$= {}_{n-1}C \times BU^2 + \cdots + {}_{n-1}C_{n-1} \times BU^{n-1}$$

..... (공식 7)

하나의 프로세서가 버스를 사용할 때 버스의 간섭에 의한 지연시간을 계산하기 위해서 (공식 6)을 한 프로세서를 중심으로 바꾼 것이 (공식 7)이 된다. (공식 6)을 표 5의 각 형태에 적용할때 두번째 항부터는 1% 미만의 값을 갖기 때문에 무시하도록 하고 계산한다. 6개의 프로세서를 꽂을 때 한 프로세서가 다른 한 프로세서와 동시에 버스를 사용하게 될 확률은 ${}_5C_1 \times BU^2$이 된다. 두개가 동시에 버스를 요구할 때 하나는 진행되고 하나가 기다리게 되므로 평균 지연

시간은 단위 인스트럭션의 수행시간의 $5 \times BU^2/2$배로 계산할 수 있다. 이와 같은 간섭은 프로세서의 성능의 저하를 초래하게 되는데 각 캐쉬 형태의 버스 간섭을 고려하여 계산한 전체의 성능은 표6과 같다. 표6의 아래 항은 한개의 프로세서당 성능으로써 많은 버스 이용률을 나타내는 캐쉬 메모리를 사용하는 프로세서는 많은 성능의 저하가 일어남을 알 수 있다.

표6. 여러가지 캐쉬/블럭 크기에 대한 시스템의 성능(n=6)

BS/CS	8 Kbytes	16 Kbytes	32 Kbytes
4 Bytes	9.63/8.99 1.60/1.49	10.8/10.0 1.8/1.66	11.8/10.8 1.96/1.8
16 Bytes	11.2/10.3 1.86/ 1.7	12.1/11.0 2.01/1.83	12.7/11.6 2.10/1.93

4. 고찰

앞 장에서는 HW.1에서 구현할 캐쉬의 형태별 사양, 설계 변수의 산출, 끝으로 HW.1이 낼 수 있는 성능을 알아보았다. 이 과정에서 두 개 이상의 프로세서에 의해 공유되는 변수에 의한 캐쉬의 간섭을 무시했으며, 입출력 프로세서의 버스 사용에 의한 지연시간을 무시했기 때문에 버스 이용률과 AAT가 약간 늘어날 수 있음을 감안해야한다. 또한 응용 프로그램의 특성 (여러개의 프로세서에 의한 공유변수의 정도, 입출력 수행의 빈도 등)에 따라 캐쉬 성능의 변화를 예상할수 있다. 3.2에서는 캐쉬 메모리가 버스를 사용하는 시점을 버스 이용률에 따른 균등한 확률 분포로 가정하였으나, 실제 버스의 사용은 응용 프로그램과 Operating System (OS)의 상태에 따라 가정한 것과 차이를 보이는 경우가 발생할 수 있다. 버스 사용의 집중 현상은 메인메모리의 지연 시간을 증가시켜서 시스템의 성능 저하를 초래하게 된다.

프로세서의 Context switching이 발생하면 현재 수행중인 일을 중단하고 보다 우선순위가 높은 새로운 일을 수행하게 되는데, 이 순간 캐쉬 메모리의 내용은 전에 수행하던 일에 대한 내용이 있을 뿐 새로이 시작하는 일에 대한 것은 거의 없게 된다. 이와 같은 캐쉬 메모리의 상태에서 일을 시작하는 상태를 Cold start라 하고, 어느 정도 Miss가 발생되어 캐쉬 메모리의 내용이 지금 수행중인 일과 관련된 것을 갖고 동작하는 상태를 Warm start라고 한다. Cold start 상태와 Warm start를 비교하면, 후자의 경우는 비교적 버스의 사용이 잘 분포되지만 전자는 많은 Miss가 발생되고 따라서 버스의 사용이 집중되게 된다. 따라서 두개 이상의 프로세서가 동시에 Context switching이 발생하면 버스의 지연시간이 증가하기 때문에 시스템의 성능이 일시적으로 떨어지게 된다. 그러므로 캐쉬 메모리의 성능을 극대화하기 위해서 OS는 두개 이상의 프로세서가 동시에 Context switching이 발생하지 않도록 하는 것이 바람직하다. 이와 같은 시도는 상용화되고 있는 TCMCS에서 찾아볼 수 있는데 인터럽트의 하드웨어 수준의 분배[6]과 O.S의 Process scheduler의 조정[7] 등이 그것이다.

끝으로 본 결과를 요약하면, VME bus, 68020-16MHz 그리고 8 Kbyte의 Write-thru 캐쉬 메모리를 사용하여 약 10 MIPS (MC68020의 MIPS)의 성능을 얻을 수 있다는 것이다. Write-thru 캐쉬 메모리의 성능 향상을 극대화하기 위한 방법으로 캐쉬 메모리 자체의 하드웨어를 보강하는 방법 (Dual Tag Memory)과 메인 메모리의 Write 주기를 짧게 하는 방법이 있으며, 그밖에 OS와 인터럽트의 구조 등과의 조화를 이루므로써 성능의 향상을 올릴 수 있다. 현재 Copyback 캐쉬 메모리의 특성과 Consistency protocols에 대한 연구를 계속 진행 중이다.

참고문헌

1. 한셈틀 하드웨어 팀, 한셈틀 하드웨어 사양,TD 87-HW 1, 1987.1
2. A. J. Smith,"Cache Memory,"Computing Surveys, Vol.14, NO.3, Sept.1982
3. James Archibald and Jean-Loup. Baer, "Cache Coherence Protocols : Evaluation Using a Multiprocessor Simulation Model," ACM Tr. on Computer Systems, Vol. 4, No. 4, Nov.1986
4. Doug Mac Gregor and Jon Rubinstein,"A Per-

formance Analysis of MC68020-based Systems," IEEE MICRO, Dec. 1985
5. J. K. Annot and M. D. Janssens, Multi Processor UNIX, Delft University of Technology Department of Electrical Engineering, July 1985
6. 박 병관, Interrupt System in Tightly Coupled Multiprocessor Computer System, TD87-HW-3, 1987.3
7. Bob Beck, Bob Kasten, "VLSI Assist in Building a Multiprocessor UNIX System, Proc. of USENIX, Summer, 1985
8. Motorola, MC68020 32-Bit Microprocessor User's Manual, 2nd ed, Pretice-Hall, 1985
9. Motorola, VME bus Specification Manual, Rev. 3, Feb, 1985
10. DEC, VAX Hardware Handbook, 1982
11. Robert Olson, "Parallel Processing in a Message Based Operating System," IEEE Software, July, 1985
12. Texas Instruments, Supplement to MOS Memory Data Book, 1984

電子用語 略語表

CPU : Central Processing Unit
CRU : Channel Request Unit
ECU : External Control Unit
EPU : Execution Processing Unit
GIU : G Bus Interface Unit
MAU : Memory Control and Address Translation Unit
MIU : Memory Interface Unit
MMU : Main Memory Unit
MPU : Micro Processor Unit
PLA : Programmable Logic Array
SIM : Single Inline Memory Module
ASTTL : Advanced Schottky TTL
TTL : Transistor Transistor Logic
CAS : Column Address Strobe
DIP : Dual Inline Package
DMA : Direct Memory Access
ECL : Emitter Coupled Logic
MAB : Memory Array Bus
MIPS : Million Instruction Per Second
PGA : Pin Grid Array
PLCC : Plastic Leaded Chip Carrier
RAS : Row Address Strobe
RFI : Radio Frequency Interference
SMB : System Main Bus
SOIC : Small Out-line Integrated Circuit package
SOJ : Small Out-line J-lead package
AM : Amplitude Modulation
BPF : Band Pass Filter
C/N : Carrier to Noise Ratio
LPF : Low Pass Filter
S/N : Signal to Noise Ratio
ASIC : Application Specific Integrated Circuit
ASTL : Advanced Schottky Transistor Logic
PLD : Programmable Logic Device
RISC : Reduced Instruction Set Computer