복수 캐시로 구성한 미디어 프로세서의 설계
(Design of A Media Processor Equipped with Dual Cache)

문 현 주 † 전 중 남 †† 김 석 일 †††
(Hyunju Moon) (Joongnam Jeon) (Sukil Kim)

요 약 본 논문에서는 미디어 프로세서에서 메모리 지연으로 인한 성능 저하를 방지하기 위하여 멀티미디어 데이타 캐시와 일반 데이타 캐시로 구성된 이중 캐시 구조의 미디어 프로세서를 제안하였다. 제안된 프로세서에서는 응용 프로그램에서 서브워드 명령으로 표현되는 멀티미디어 데이타를 멀티미디어 캐시에 적재하고 나머지 데이타를 일반 데이타 캐시에 적재하도록 하였다. 또한 멀티미디어 데이타 캐시는 멀티미디어 데이타의 높은 지역성을 활용하도록 2개의 데이타 블록을 동시에 인출하는 선인출 기법을 적용하였다. MPEG과 JPEG 벤치마크에 대한 실험 결과, 제안한 프로세서의 캐시구조가 단일 캐시 구조에 비하여 성능이 우수하였다.

키워드 : 멀티미디어 데이타, 이중 캐시 구조, 미디어 프로세서, 서브워드 병렬성, 블록 선인출

Abstract In this paper, we propose a mediaprocessor of dual-cache architecture which is composed of the multimedia data cache and the general-purpose data cache to prevent performance degradation caused by memory delay. In the proposed processor architecture, multimedia data that are written in subword instructions are loaded in the multimedia data cache and the remaining data are loaded in the general-purpose data cache. Also, we use multi-block prefetching scheme that fetches two consecutive data blocks into a cache at a time to exploit the locality of multimedia data. Experimental results on MPEG and JPEG benchmark programs show that the proposed processor architecture results in better performance than the processor equipped with single data cache.

Key words : multimedia data, dual-cache, mediaprocessor, subword parallelism, multi-block prefetching

1. 서 론

반도체 공정 기술이 발전을 거듭함에 따라 다량의 트랜지스터를 효과적으로 사용하면서 최고의 성능을 보장하는 차세대 마이크로 프로세서의 구조 개발에 관한 연구가 활발히 시도되고 있다[1]. 특히 컴퓨터가 처리하는 정보는 기존의 문자나 숫자를 처리하던 방식으로부터 2차원 영상 정보를 처리하는 방식으로 발전하게 되었으며, 최근에는 정지 영상, 동영상, 음성, 3차원 그래픽 및 애니메이션 등의 복잡한 멀티미디어 정보를 다루게 되었다. 이러한 정보를 빠르게 처리하기 위하여 마이크로프로세서 구조는 명령어 수준에서 멀티미디어 데이타 처리를 지원하는 방법을 모색해 왔으며 주요 마이크로프로세서 구조들은 멀티미디어 지원을 위한 확장 명령어 세트를 개발하여 별도의 하드웨어 없이 프로세서 내에서 멀티미디어 데이타의 처리 속도를 향상시키는 전략을 취하고 있다[2-4].

그러나 실시간 처리를 요하는 멀티미디어 응용 프로그램의 처리나 복합 스트림에 대한 동시 처리, 방대한 크기의 미디어 데이타 세트에 대한 캐시 최적화 등의 기술적인 문제를 해결하기에 범용 프로세서의 성능에는 한계가 있으므로 멀티미디어 데이타를 보다 전문적으로 처리할 수 있는 전용 미디어 프로세서에 관한 연구가 수행되고 있다[5-9]. 이들 연구의 대부분은 슈퍼스칼라(superscalar)나 VLIW와 같이 명령어 수준 병렬성(ILP: Instruction

· 본 연구는 한국과학재단 목적기초연구(R05-2002-000-01470-0)지원으로 수행되었음
† 학생회원 : 충북대학교 대학원 전자계산학과
hjmoon@john.chungbuk.ac.kr
†† 종신회원 : 충북대학교 전기전자및컴퓨터공학부 교수
joongnam@cbucc.chungbuk.ac.kr
††† 종신회원 : 충북대학교 전기전자및컴퓨터공학부 교수
ksi@cbucc.chungbuk.ac.kr
논문접수 : 2002년 3월 15일
심사완료 : 2002년 9월 6일

Level Parallelism)을 활용하는 구조를 바탕으로 설계함으로써 빠른 처리속도에 대한 요구를 해결하고 있다. 또한 데이타의 워킹 세트(working set)가 매우 크며 개개의 데이타 크기가 작고 데이타의 지역성이 높은 멀티미디어 데이타의 특성을 반영하여 하나의 명령어로 독립적인 여러 개의 데이타를 동시에 처리하도록 하는 서브워드 병렬성을 활용하고 있다[7-10].

이와 같은 다양한 연구의 결과로 미디어 프로세서의 처리 성능이 급격히 향상됨에 따라 응용 프로그램의 수행성능이 상대적으로 메모리 참조시간에 의하여 결정되는 현상이 두드러지게 나타나고 있다. 특히, 대규모의 멀티미디어 데이타를 처리해야 하는 응용 프로그램의 경우에는 메모리 참조 빈도가 범용 프로그램에 비하여 매우 높다[12]. 뿐 만 아니라, 서브워드 병렬성을 사용하는 미디어 프로세서 구조에서는 여러 개의 데이타가 동시에 처리되므로 보다 빠른 데이타의 공급을 필요로 한다.

본 논문에서는 멀티미디어 데이타의 참조 특성을 고찰하고, 이를 바탕으로 빠른 데이타 공급을 지원하는 새로운 캐시 구조를 채용하는 미디어 프로세서를 제안한다. 멀티미디어 응용 프로그램에서 사용되는 데이타는 이미지, 영상, 사운드 등의 멀티미디어 데이타와 이를 처리하기 위하여 사용되는 수치나 문자 등의 일반 데이타로 분류할 수 있다. 멀티미디어 데이타는 규모가 크며 메모리에 인접하여 저장된 데이타들이 연속적으로 사용될 가능성이 높다는 점 등 일반 데이타와는 메모리 참조 패턴이 서로 다르다. 그러나 기존의 미디어 프로세서 구조들은 캐시 구조에 멀티미디어 데이타의 메모리 참조 특성을 충분히 활용하지 못하고 있다.

본 논문에서는 미디어 프로세서의 데이타 캐시를 멀티미디어 데이타 캐시와 일반 데이타 캐시의 두 모듈로 구성하고 각 캐시에 멀티미디어 데이타와 일반 데이타를 각각 적재하도록 하였다. 또한 각 캐시에 서로 다른 인출 기법을 적용함으로써 메모리 참조로 인한 지연을 줄이도록 하였다. 응용 프로그램에 사용되는 데이타가 멀티미디어 데이타인지 여부를 판단하기 위해서는 서브워드 명령어로 처리되는 데이타를 멀티미디어 데이타로 간주하고 이들을 멀티미디어 데이타 캐시로 적재하도록 하였다. 일반 데이타 캐시에 데이타 적재시에는 기존의 캐시 구조에서 일반적으로 사용하고 있듯이 하나의 데이타 블록을 인출하도록 하고, 멀티미디어 데이타 캐시에 데이타 적재시에는 현재 요구되는 데이타 블록과 함께 메모리상에 인접하여 저장되어 있는 여러 개의 메모리 블록을 함께 인출하는 데이타 선인출 기법을 적용하도록 함으로써 멀티미디어 데이타의 높은 지역성을 활용하도록 하였다. 또한 한 번에 인출할 데이타 블록의 수와 미디어 프로세서를 구성하기 위한 연산처리기의 수는 실험을 통하여 결정하였다.

본 논문의 구성은 다음과 같다. 제 2장에서는 미디어 프로세서 구조에 관한 기존 캐시 구조의 연구를 살펴본다. 제 3장에서는 멀티미디어 데이타에 관한 특성을 고찰하고 그 결과를 바탕으로 이중 데이타 캐시와 이를 채용하는 미디어 프로세서를 설계한다. 제 4장에서는 실험으로 통하여 제안한 프로세서 구조에서의 캐시 성능을 분석한다. 마지막으로 제 5장에서는 본 논문의 결론을 도출하였다.

2. 미디어 프로세서의 캐시 구조

기존의 미디어 프로세서 구조에 관한 연구들은 프로세서로의 빠른 데이타 공급을 위하여 다양한 메모리 구조 및 데이타 접근 기법을 제시하였다. 멀티미디어 데이타의 낮은 재사용성을 해결하기 위한 메모리 구조 연구에서는 메모리 영역을 지정하는 특수한 레지스터를 두고, 이 레지스터가 가리키는 주소 이상의 메모리에 대해서는 캐시를 경유하지 않고 직접 레지스터로 읽어 오도록 하는 기법이 연구된 바 있다. 이러한 기법을 채용한 프로세서 구조로는 Phillips사가 설계한 TriMedia 프로세서[7]가 있다. TriMedia 프로세서는 재사용성이 적은 멀티미디어 데이타의 경우에 캐시를 거치지 않도록 하여 전체적인 데이타 캐시의 교체 회수를 감소시킴으로써 캐시의 성능저하를 방지하고 있다. 이를 위하여 컴파일러는 캐시를 경유할 주소번지를 미리 결정하고, *store* 명령의 수행시 데이타 저장 위치를 미리 알려주어야 한다. 또한 구조적으로는 메모리 파이프라인 및 레지스터 제어를 위한 회로가 복잡해지는 문제를 내포하고 있다.

캐시를 여러 단계로 나누어 대용량의 캐시를 채용하는 프로세서 구조에 관한 연구도 수행된 바 있다. 이들 연구에서는 프로세서 외부에 SDRAM으로 대용량의 외부 캐시를 구성하고 내부 캐시에서 미스가 발생한 경우 외부 캐시의 내용을 참조하도록 하였다. 이 과정에서 빠른 참조를 위하여 내부 캐시와 외부 캐시간의 참조과정을 파이프라인으로 구성하였다. 이러한 기법은 Sun Micro System사에서 설계한 UltraSPARC 시리즈[8-9]에 적용되었다. MicroUnity 시스템[11]의 경우에는 SDRAM을 칩 내부에 구성하여 별도의 외부 캐시가 필요하지 않다.

대용량 레지스터 파일을 채용하여 캐시 구조의 효과를 얻는 연구[14], 메모리와 캐시간의 데이타 이동을 프로세서의 연산과 동시에 수행하는 구조[5] 등도 연구되

고 있다. 그러나 이들의 경우에는 하드웨어 구성비용 및 복잡도로 인한 문제점이 있다. 따라서 메모리 참조시간을 줄일 수 있는 새로운 구조 및 기법에 관한 지속적인 연구가 필요하다.

3. 이중 캐시 구조

3.1 멀티미디어 데이타의 특성

멀티미디어 응용 프로그램에서 사용되는 데이타를 멀티미디어 데이타와 일반 데이타로 구분할 때, 메모리 참조의 대부분을 차지하는 멀티미디어 데이타는 일반 데이타와 메모리 참조 특성이 매우 다르다. 멀티미디어 데이타에 대한 메모리 참조의 첫 번째 특징은 대량의 데이타가 연속적으로 참조되는 것이다. 멀티미디어 응용 프로그램의 많은 부분이 대규모의 멀티미디어 데이타 각각에 대하여 단순한 연산을 적용하는 구조로 표현된다. 그 결과, 멀티미디어 데이타는 최근에 접근된 데이타가 캐시 블록의 교체 이전에 다시 접근될 가능성, 즉 재사용성이 매우 낮다. 하나의 캐시에 멀티미디어 데이타와 일반 데이타를 함께 적재하고 데이타 블록 교체를 위하여 LRU(Least Recently Used) 정책을 사용하는 캐시 구조에서, 큰 규모의 멀티미디어 데이타를 연속해서 캐시에 적재할 경우 캐시 블록의 교체(replacement)가 발생한다. 이 때, 이전에 사용된 멀티미디어 데이타 블록은 더 이상 재사용될 가능성이 없음에도 불구하고 최근에 사용된 데이타 블록이므로 재사용의 가능성이 충분한 일반 데이타 블록을 교체하게 될 가능성이 높다. 만일, 멀티미디어 데이타 블록을 적재하기 위하여 이미 사용된 멀티미디어 데이타 블록을 교체한다면 일반 데이타 블록을 재사용할 때 발생하는 캐시 미스를 방지할 수 있다. 그러나 하나의 캐시 모듈에 멀티미디어 데이타와 일반 데이타를 함께 적재하는 기존의 구조에서 블록 교체를 위하여 멀티미디어 데이타 블록과 일반 데이타 블록을 구분하는 것은 구조적으로 비용이 매우 크다.

멀티미디어 데이타에 대한 메모리 참조의 두 번째 특징은 기억장치에 인접하여 저장된 데이타들이 연속적으로 접근될 가능성, 즉 지역성(locality)이 매우 높다는 점이다. 또한 참조 패턴이 일정한 간격이나 간단한 함수로 표현될 수 있는 가능성이 높다. 만일 가까운 장래에 사용될 데이타를 미리 예측할 수 있다면, 캐시 미스시 현재 요구되는 데이타 블록을 메모리로부터 인출할 뿐만 아니라 이후에 사용될 것으로 예측된 데이타 블록을 미리 인출함으로써 이후에 발생할 캐시미스 가능성을 감소시킬 수 있다. 이러한 데이타 블록 선인출(prefetching) 기법[15-17]은 멀티미디어 데이타의 높은 지역성 및 단순한 참조패턴과 부합하여 메모리 참조시 큰 효과를 발휘할 수 있다. 그러나 일반 데이타는 멀티미디어 데이타와 비교하여 지역성이 낮으며 데이타 참조 패턴 또한 예측이 어려우므로, 하나의 캐시 모듈에 멀티미디어 데이타와 일반 데이타를 함께 적재하는 기존의 구조에서 멀티미디어 데이타와 일반 데이타에 동일한 예측 기법을 적용하여 데이타 블록을 선인출 할 경우 만족 할만한 성능 향상을 얻을 수 없다. 뿐 만 아니라, 일반 데이타에 대한 선인출은 더 많은 데이타 블록의 교체를 수반하므로, 재사용 될 데이타 블록의 교체로 인한 부작용을 유발할 우려가 있다.

이러한 문제를 해결하기 위해서는 캐시 내에 멀티미디어 데이타와 일반 데이타를 저장하는 영역을 구분하거나 별도의 캐시 모듈로 구성할 필요가 있다. 따라서 본 논문에서는 멀티미디어 데이타와 일반 데이타별로 캐시를 구성하는 구조를 연구하였다. 그림 1은 본 논문에서 설계한 캐시 구조이다.

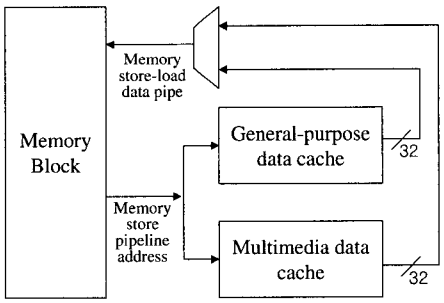


그림 1 이중 캐시 구조

3.2 멀티미디어 데이타 캐시

범용 고성능 프로세서에서 한 번에 처리되는 데이타의 크기가 32비트 또는 64비트 단위인데 반해 멀티미디어 데이타의 크기는 주로 8비트 또는 16비트인 점을 감안하여 서브워드 병렬성을 활용하는 미디어 프로세서 구조에서는 하나의 데이타 워드를 일정한 크기로 구분하여 여러 개의 멀티미디어 데이타에 대한 처리를 동시에 수행하도록 하고 있다. 이러한 프로세서 구조에서는 기존의 명령어 세트에 새로운 서브워드 명령어 세트를 추가하고, 컴파일러가 목적 코드를 생성하는 과정에서 멀티미디어 데이타에 대한 연산을 추출하여 서브워드 명령어로 표현하도록 한다.

본 논문에서는 메모리로부터 캐시로 데이타를 적재할 때, 멀티미디어 데이타와 일반 데이타를 구분하기 위하여 서브워드 명령어로 표현된 데이타를 멀티미디어 데이

타로 분류하도록 하였다. 이를 위하여, 기존의 서브워드 명령어 세트에 서브워드 *load/store*를 위한 명령어 그룹을 추가하고 컴파일시 이들 명령어를 이용하여 멀티미디어 데이타에 대한 메모리 참조를 나타내도록 하였다. 따라서 일반 *load/store* 명령에 의한 메모리 참조시에는 요구되는 데이타를 일반 데이타 캐시로 적재하고, 서브워드 *load/store* 명령에 의한 메모리 참조시에는 데이타를 멀티미디어 데이타 캐시로 적재하도록 하였다.

또한 일반 데이타 캐시를 위한 데이타 인출 기법은 일반적인 경우와 마찬가지로 블록 단위의 인출을 적용하도록 하였다. 멀티미디어 데이타 캐시의 경우에는 데이타 블록 선인출 기법이 가능한 구조를 채택하였다. 기존에 연구된 데이타 블록 선인출 기법은 반복문에서 이전에 인출된 데이타 블록의 메모리 주소를 참조하여 다음에 선인출할 데이타 블록의 메모리 주소를 결정하는 기법[15], 이전에 인출된 데이타의 메모리 주소를 참조하되 분기예측표(branch prediction table)를 이용하여 사용될 것으로 예상되는 데이타 블록을 사용 전에 인출하는 방법[16], 메모리 참조패턴을 예측하지 않고 연속적으로 저장된 몇 개의 데이타 블록을 선인출 하는 기법[17] 등이 있다. 이들 중, 가장 효과가 좋은 기법은 분기예측표를 이용하는 방법이나 이 방법은 분기예측표 및 요구되는 하드웨어로 인하여 구조적 복잡도가 증가한다.
멀티미디어 데이타의 경우, 대부분의 메모리 참조가 순차적이거나 단순한 패턴을 반복하므로 낮은 선인출 비용을 요구하는 단순한 형태의 기법을 사용하는 것이 보다 효과적이다. 따라서 하나의 데이타 블록을 인출할 경우에 연속된 몇 개의 데이타 블록을 함께 인출하는 선인출 기법을 사용하는 것으로 간주하였다.

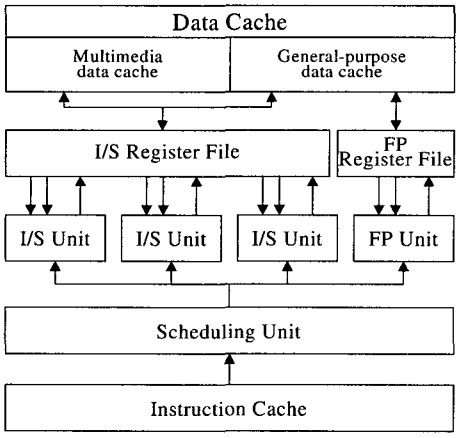


그림 2 DCM 프로세서 구조

그림 2는 본 논문에서 설계한 프로세서 구조이다. 제안하는 미디어 프로세서는 VLIW 프로세서 구조를 기반으로 서브워드 병렬성을 지원하는 구조로서 본 논문에서는 DCM(Dual-Cache Mediaprocessor) 프로세서로 칭하기로 한다. DCM 프로세서에서 연산처리기는 여러 개의 정수처리기와 실수처리기로 구성되며 메모리 참조 명령에 대한 처리는 정수처리기가 담당한다. DCM 프로세서가 지원하는 서브워드 연산은 표 1과 같다. 표 1에서 대부분의 명령어는 정수 연산과 같은 연산을 수행한다. 따라서 정수 처리기와 서브워드 처리기를 통합하여 I/S(Integer/Subword) 처리기로 구성하였다. 단, 각 서브워드간에 캐리가 전달되지 않는 점이 일반 정수연산과 다르므로 이를 위하여 각 서브워드의 마지막 비트에서 캐리가 발생할 경우 이를 무시하는 제어기를 추가하였다. I/S 처리기는 32비트 처리기로서 8비트 및 16비트 서브워드를 처리할 수 있도록 하였다. 각 연산처리기 그룹은 레지스터를 공유하며 실수 처리기 및 I/S 처리기간의 데이타 교환은 일반 데이타 캐시를 통하여 수행된다. 연산처리기의 수는 제 4장의 실험을 통하여 결정하기로 한다.

표 1 서브워드 명령어

유형	명령어	어셈블리 코드
사칙연산	덧 셈	s_add8sw, s_add16sw, s_add8ss, s_add16ss, s_add8su, s_add16su
	뺄 셈	s_sub8sw, s_sub16sw, s_sub8ss, s_sub16ss, s_sub8su, s_sub16su
	곱 셈	s_mul16sl, s_mul16sh, s_Mul16UL, s_muL16uh
비 교	비 교	s_gt8s, s_gt16s, s_gt8u, s_gt16u, s_ge8s, s_ge16s, s_ge8u, s_ge16u, s_eq8, s_eq16
쉬프트	쉬프트	s_shL8s, s_shl16s, s_shl8u, s_shL16u s_shr8s, s_shr16s, s_shR8u, s_shr16u
크기변경	압 축	s_pack16S, s_pack32s, s_pack16U, s_pack32u
	확 장	s_unpack8s, s_unpack16s, s_unpack8u, s_unpack16u
기타	곱 합	s_muladds, s_mulladdu
	평 균	s_avg8s, s_avg16s, s_avg8u, s_avg16u
메모리 참조	읽기	s_lb, s_lh, s_lw
	쓰기	s_sb, s_sh, s_sw

DCM 프로세서는 VLIW 프로세서 구조와 달리 명령어 캐시에 명령어를 저장할 때, NOP으로만 구성된 긴 명령어(LNOP: Long NOP)를 제거하고 실행시간에 스케쥴링 유니트를 통하여 이를 복원하는 구조이다. 명령어 캐시 및 각 데이타 캐시는 세트 연관 사상(set-associative) 방식에 의하여 운영되며 LRU(Least Recently Used) 교환 정책과 write-back 쓰기 정책을 채택하고 캐시 블록의 크기는 32바이트로 하였다. 각 캐시의 크기도 제 4장의 실험을 통하여 결정하였다.

4. 실험 및 고찰

4.1 실험 환경

제안된 캐시 구조의 성능을 분석하기 위하여 본 논문에서는 DCM 프로세서 시뮬레이션 시스템을 구현하였다. 구현한 시스템은 그림 3과 같이 DCM 프로세서용 목적코드를 생성하는 병렬화 어셈블러(parallelizing assembler)와 DCM 프로세서용 목적코드에 대한 수행을 모의하는 시뮬레이터로 구성된다.

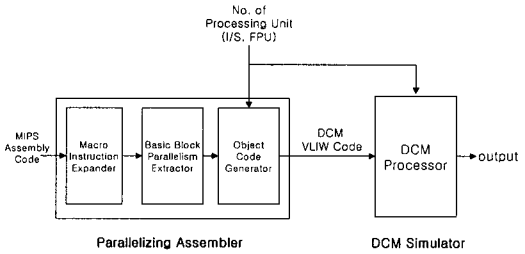


그림 3 시뮬레이션 시스템 구성도

병렬화 어셈블러는 C 언어로 작성된 멀티미디어 응용 프로그램을 MIPS 프로세서용 컴파일러로 컴파일 하여 얻은 MIPS 어셈블리 코드를 입력으로 받아 기본 블록을 추출하고 기본 블록 내에서 병렬성 있는 명령어를 추출하여 긴명령어(Very Long Instruction Word)로 구성된 목적코드를 생성한다. 이 과정에서 MIPS 프로세서용 컴파일러는 서브워드 명령을 지원하지 않으므로, 소스 프로그램에서 서브워드 명령어로 변환될 수 있는 부분에 컴파일러 디렉티브를 삽입하여 컴파일 한 후 생성된 어셈블리 코드에서 이 부분을 찾아 기존의 정수 명령어 대신 서브워드 명령어로 변환하였다. 또한 VLIW형 긴명령어를 구성하는 단위 명령어의 수는 DCM 프로세서를 구성하는 연산처리기의 수에 의해 결정되므로 정수/서브워드 처리기 및 실수처리기의 수를 병렬화 어셈블러 프로그램의 입력으로 받아 처리하도록 하고, 목적코드의 헤더 부분에 이 정보를 삽입하여 시뮬레이터에서 사용하도록 하였다.

DCM 프로세서 시뮬레이터는 병렬화 어셈블러에 의하여 생성된 DCM 프로세서용 목적 코드를 입력으로 받아 들여 헤더에 포함된 프로세서 정보를 분석하여 DCM 프로세서를 구성하는 정수/서브워드 처리기와 실수처리기의 개수에 따라 연산처리기를 구성한다. 또한 목적코드 길이에 따라 모의 실험에 필요한 가상의 주소 공간을 생성하고 VLIW 형식의 긴명령어를 적재하여 매 사이클 당 하나의 긴명령어를 인출 및 분석하고 연산처리기에 할당한다. 긴명령어가 실행처리기에 할당되는 경우에는 각각의 명령어별로 정의된 실행 사이클 동안 연산처리기를 점유한다. 본 논문에서는 정수명령어와 서브워드 명령어의 실행은 한 사이클이 소요되며 실수명령어는 명령어별로 실행 사이클이 다른 것으로 가정하였다. 각 연산처리기의 연산 결과는 레지스터 또는 메모리에 저장하고 실행 사이클 수, 스케쥴링 시 삽입된 LNOP 수, 캐시적중 및 미스 횟수 등 여러 가지 파라메터 값을 정보파일에 저장하도록 하였다.

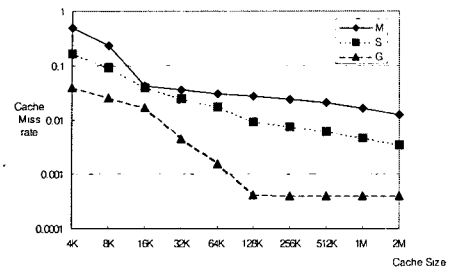
실험에 사용된 벤치마크 프로그램은 멀티미디어 데이타 처리에서 벤치마크 프로그램으로 가장 널리 쓰이는 MPEG과 JPEG의 인코더와 디코더를 사용하였다. 실험은 두 가지로 수행하였다. 첫째, 이중 캐시의 성능 분석을 위한 실험에서는 캐시를 두 개의 모듈로 분할한 경우와 분할하지 않은 경우에 대하여 각각 캐시미스율을 측정하여 비교하였다. 또한 멀티미디어 데이타 캐시에서 데이타 선인출의 효과를 분석하기 위하여, 캐시 미스시 메모리로부터 선인출 할 데이타 블록의 수를 변화시키면서 캐시미스율의 변화를 측정하였다. 이러한 실험 결과를 바탕으로 이중 캐시 구조의 각 캐시 크기를 결정하고 선인출 기법을 적용할 블록의 수를 결정하였다.

두 번째 실험에서는 DCM 프로세서를 구성할 연산처리기의 수를 결정하기 위하여 MPEG과 JPEG을 구성하는 핵심 모듈인 이산 코사인 변환 모듈(FDCT), 역이산 코사인 변환 모듈(IDCT) 및 이동 추정 모듈(ME: motion estimation) 프로그램에 대하여 연산처리기의 수를 증가시키면서 수행시간의 변화를 측정하였다.

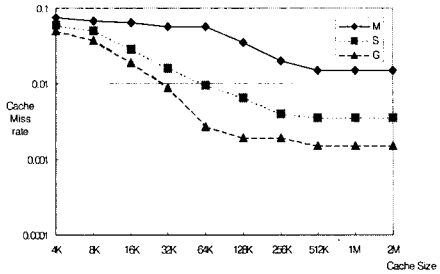
4.2 실험 및 분석

그림 4는 MPEG과 JPEG의 각 프로그램 모듈을 수행할 때 멀티미디어 데이타와 일반 데이타를 하나의 캐시에 함께 적재한 경우와 이중 캐시를 이용하여 이들을 분할하여 저장한 경우의 캐시미스율을 비교한 것이다. 그림에서 멀티미디어 데이타와 일반 데이타를 함께 저장하는 캐시의 미스율을 S로 표시하였으며 이중 캐시의 각 모듈에 대해서는 멀티미디어 캐시의 미스율을 M으로, 일반 데이타 캐시의 미스율을 G로 표시하였다. 이중 캐시를 사용한 경우, MPEG과 JPEG에 의하여 처리되는 이미지 데이타를 서브워드 명령어로 처리하도록 하고 이들을 멀티미디어 데이타로 분류하여 멀티미디어 데이타 캐시에 적재하였다. 각 캐시의 크기는 4K~2M 바이트까지 변화시키면서 실험한 것이다.

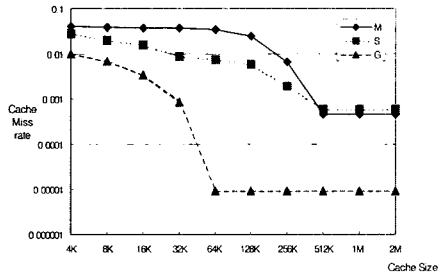
그 결과 이중 캐시 구조를 사용할 경우, 멀티미디어 캐시의 미스율(M)이 일반 데이타 캐시의 미스율(G)에 비해 현저하게 높은 것을 알 수 있다. 이것은 멀티미디



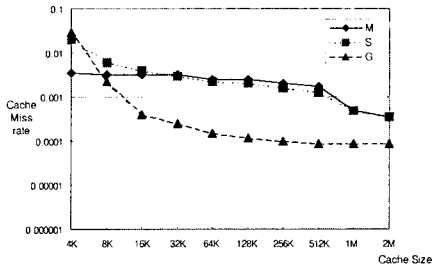
(a) JPEG 인코더



(b) JPEG 디코더



(c) MPEG 인코더



(d) MPEG 디코더

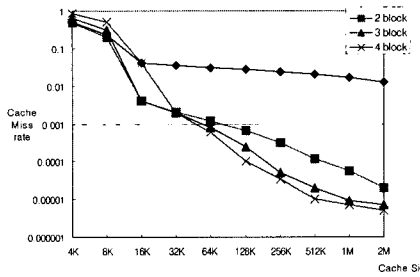
그림 4 데이타 유형별 캐시미스율 비교

어 데이타의 재사용성이 매우 낮음을 나타낸다. 또한 이중 캐시 구조의 각 캐시 모듈과 단일캐시의 미스율(S)을 비교해 보면, 단일 캐시의 미스율이 일반 데이타 캐시의 미스율 보다는 높고 멀티미디어 캐시의 미스율 보다는 낮은 것을 알 수 있다. 즉, 캐시를 분할하지 않는 경우, 일반 데이타와 함께 적재되는 멀티미디어 데이타로 인하여 전체적인 캐시 미스율이 높아진 것으로 판단된다.

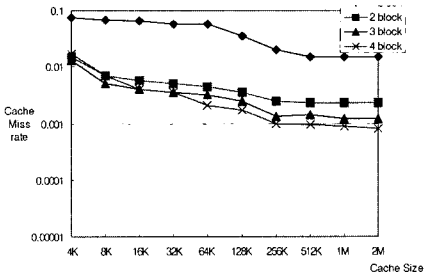
또한 각 그래프는 캐시의 크기를 증가시킬수록 캐시 미스율이 감소하는 추세를 나타내고 있다. 특히, 일정 크기 이상에서는 캐시미스율의 변화가 거의 나타나지 않는 것을 알 수 있다. 일반 데이타 캐시의 경우에는 캐시 크기가 64~128K 이상이면 캐시미스율의 감소가 현저히 둔화된다. 또한 멀티미디어 데이타 캐시의 경우에도 캐시 크기가 512K 이상이면 캐시미스율의 변화가 거의 없는 것을 알 수 있다. 이는 프로세서 구조의 설계시 캐시 크기를 무조건 크게 하기보다는 비용대 효율 측면을 고려한다면 각 캐시를 적절한 크기로 구성할 필요가 있음을 시사하는 것으로 볼 수 있다.

그림 5는 멀티미디어 데이타 캐시에 블록 선인출 기법을 적용한 경우의 캐시미스율을 측정한 것이다. 그림에서 1 block으로 표시된 결과는 캐시 미스가 발생한 데이타가 포함된 캐시 블록만을 읽어 오는 경우, 즉 선인출을 수행하지 않는 경우이며, 2~4 block으로 표시된 결과는 각각 캐시 미스 발생시 연속된 2~4개의 블록을 함께 읽어오는 경우를 나타낸다.

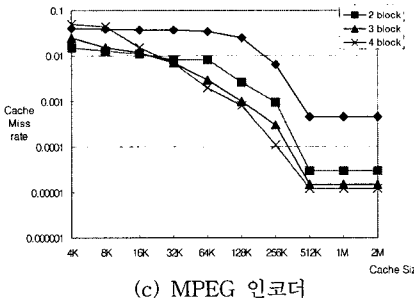
선인출 되는 블록의 수에 따른 성능 변화는 캐시의 크기에 따라 상반된 결과를 나타내고 있다. 그림 5(a)에서 JPEG 인코더의 경우, 캐시 크기가 32K 이상에서는 선인출 하는 데이타 블록의 수를 증가시킬수록 캐시미스율이 저하되고 있다. 그러나 캐시 크기가 32K 이하인 경우에는 선인출 하는 데이타 블록의 수를 증가시킬수록 오히려 캐시미스율이 증가하는 경향을 나타내고 있다. 그 이유는 캐시미스율에 영향을 주는 두 가지 원인으로부터 해석될 수 있다. 첫째는 캐시의 적중이고 둘째는 캐시 교체이다. 즉, 캐시의 크기가 큰 경우에는 선인출시 많은 수의 블록을 읽어옴으로써 이들이 가까운 장래에 사용될 때 캐시에서 적중될 가능성이 높아진다. 그리고 많은 수의 블록을 선인출 하여도 캐시의 크기가 충분하므로 기존에 캐시에 적재되어 있던 데이타들은 아무런 영향을 받지 않는다. 반면, 캐시의 크기가 작은 경우에는 선인출시 많은 수의 블록을 읽어옴으로써 미리 캐시에 적재되어 있던 데이타들이 교체될 가능성이 높아진다. 이 과정에서 교체된 데이타가 프로그램에 의하여 다시 사용되는 경우에는 캐시미스가 발생하게 되므로 선인출 하는 블록을 증가시킴으로써 오히려 캐시 미스를 유발하게 된다.



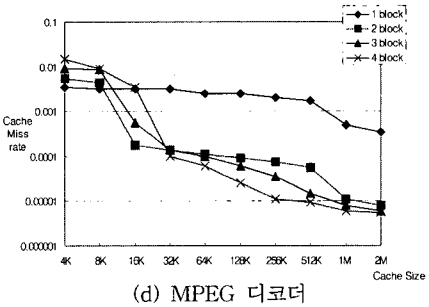
(a) JPEG 인코더



(b) JPEG 디코더

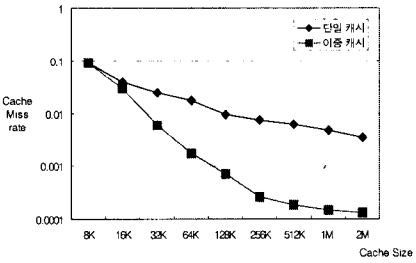


(c) MPEG 인코더

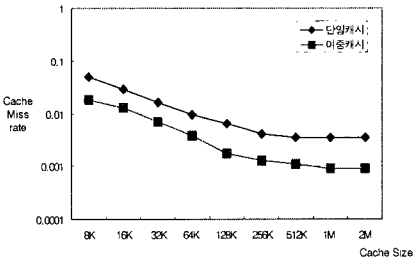


(d) MPEG 디코더

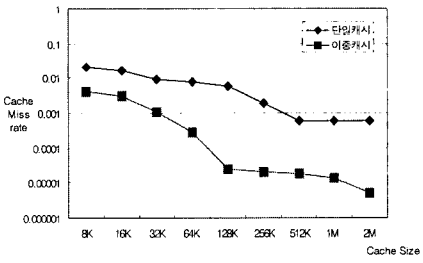
그림 5 선인출하는 블록 수에 따른 성능변화



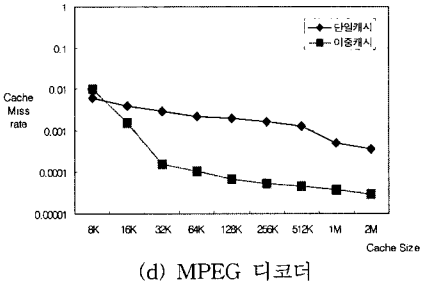
(a) JPEG 인코더



(b) JPEG 디코더



(c) MPEG 인코더



(d) MPEG 디코더

그림 6 캐시 구조별 성능 비교

따라서 선인출하는 데이타 블록의 크기를 무조건 크게 하는 것보다는 캐시의 크기를 고려하여 효과적인 블록의 크기를 결정할 필요성이 있음을 알 수 있다. 이러한 현상은 JPEG 디코더 및 MPEG의 각 프로그램 모듈에서 동일하게 나타나고 있다.

그림 4와 그림 5의 실험 결과에 기초하여 DCM 프로세서에 대한 각 캐시의 크기와 선인출할 데이타 블록의 크기를 결정하였다. 일반 데이타 캐시의 크기는 그림 4

의 실험 결과에 기초하여 64K로 결정하였으며, 멀티미디어 데이타 캐시의 크기는 그림 5의 각 결과에서 공통적으로 캐시미스율 변화폭이 둔화되는 지점인 16K로 결정하였다. 또한 멀티미디어 데이타 캐시에서 선인출 하는 블록의 수는 그림 5에서 블록 수를 증가시킴에 따른 미스율 감소가 가장 크게 나타난 2 block으로 결정하였다.

그림 6은 멀티미디어 데이타 캐시의 사용시 2 block을 선인출 하도록 한 경우에 이중캐시의 전체적인 캐시 미스율을 단일 캐시의 경우와 비교하고 있다. 이 실험에서는 동일한 크기의 이중 캐시와 단일 캐시를 비교하고 있으며, 이중 캐시는 멀티미디어 데이타 캐시와 일반 데이타 캐시의 크기를 동일하게 구성하여 실험한 결과를 나타내고 있다. 그림 6의 결과에서 모든 경우에 이중 캐시구조를 사용함으로써 캐시 미스율이 대폭 감소한 것을 알 수 있다. 그림 6(a)와 6(d)의 경우에는 캐시의 크기가 증가할수록 이중 캐시로 인한 성능향상이 더욱 증가하였으며, 그림 6(b)와 6(c)의 경우에는 캐시 크기에 관계없이 성능향상이 일정하게 나타났다. 그림 6(d)의 결과에서 캐시의 크기가 8K인 경우의 결과는 그림 4에서 보는 바와 같이 MPEG 디코더의 예외적인 데이타 참조패턴에 의한 것으로 판단된다.

그림 7은 DCM 프로세서의 연산처리기 수를 결정하기 위하여 정수/서브워드(I/S) 연산처리기의 수를 1~4개로 증가시키면서 수행한 경우의 수행시간에 대한 속도향상을 나타내고 있다. 그림 7에서 각 프로그램 모듈에 대하여 연산처리기가 3 이상인 경우에는 속도향상이 둔화되므로 DCM 프로세서를 구성할 I/S 연산처리기의 수를 3개로 결정하고, 나머지 1개의 연산처리기를 실수처리기로 구성하였다.

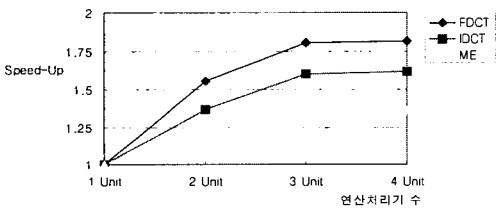


그림 7 연산처리기 수에 따른 속도향상

5. 결 론

본 논문에서는 멀티미디어 응용 프로그램의 수행 성능에 큰 영향을 미치는 메모리 참조 지연을 줄이기 위한 방안으로 일반 데이타와 멀티미디어 데이타를 분할 적재할 수 있는 데이타 캐시 구조를 제안하였다. 제안된 캐시 구조는 일반 데이타 캐시와 멀티미디어 데이타 캐시의 두 모듈로 구성하였으며, 서브워드 명령어로 표현된 데이타를 멀티미디어 데이타로 분류하여 멀티미디어 데이타 캐시에 적재하도록 하였다. 또한 멀티미디어 데이타 참조의 높은 지역성을 활용하기 위하여 멀티미디어 데이타 캐시의 데이타 적재시 연속된 두 개의 메모리 블록을 함께 적재하도록 하는 선인출 기법을 적용하였다.

제안된 캐시 구조의 성능을 평가를 위하여 시뮬레이션 시스템을 구성하고 MPEG 및 JPEG 벤치마크를 대상으로 한 실험 결과, 데이타 캐시를 두 개의 모듈로 구분함으로써 응용 프로그램 수행에 대한 캐시미스율이 현저히 감소됨을 알 수 있었다. 특히 멀티미디어 데이타의 캐시에 선인출 기법을 적용하여 멀티미디어 데이타의 높은 지역성을 활용함으로써 캐시 미스율을 낮출 수 있었다.

향후에는 멀티미디어 응용 프로그램의 데이타 참조 특성을 보다 세밀하게 분석하고 이를 반영하여 데이타를 더욱 효과적으로 선인출 할 수 있는 기법에 관하여 연구하고자 한다.

참 고 문 헌

[1] R. B. Lee, "Realtime MPEG video via software decompression on a PA-RISC processor," *COMPCON '95,* pp.186-192, 1995.

[2] A. Peleg and U. Weiser, "MMX technology Extension to the Intel architecture," *IEEE Micro,* Vol. 16, No. 4, pp.42-50, August 1996.

[3] W. A. Samaras, N. Cherukuri and S. Venkataraman, "The IA-64 Itanium processor cartridge," *IEEE Micro,* Vol. 21, No. 1, pp.82-89, January/February 2001.

[4] T. M. Conte, P. K. Dubey, M. D. Jennings, R. B. Lee, A. Peleg, S. Rathnam, M. Schlansker, P. Song and A. Wolfe, "Challenges to combining general-purpose and multimedia processors," *IEEE Computer,* Vol. 30, No. 12, pp.33-37, December 1997.

[5] C. Basoglu, W. Lee and J. S. O'Donnell, "The MAP1000A VLIW mediaprocessor," *IEEE Micro,* Vol. 20, No. 2, pp.48-59, March/April 2000.

[6] G. Frantz, "Digital signal processor trends," *IEEE Micro,* Vol. 20, No. 6, pp.52-59, November/December 2000.

[7] N. Mitchell, "Philips TriMedia: a digital media convergence platform," *WESCON '97,* pp.56-60, 1997.

[8] M. Tremblay and J. M. O'Connor, "UltraSparc I : A four-issue processor supporting multimedia," *IEEE Micro,* Vol. 16, No. 2, pp.42-50, April 1996.

[9] T. Horel and G. Lauterbach, "UltraSPARC-III : Designing third-generation 64-bit performance," *IEEE Micro,* Vol. 19, No. 3, pp.73-85, May/June 1999.
[10] R. B. Lee, "Subword Parallelism with MAX-2," *IEEE Micro,* Vol. 16, No. 4, pp.51-59, August 1996.
[11] C. Hansen, "Architecture of broadband media-processor," *IEEE COMPCON '96,* pp.25-29, February, 1996.
[12] H. Govindarajalu, A. Rengachari and A. Omondi "DSTRIDE: Data-cache miss-address-based stride prefetching scheme for multimedia processors," *Proc. 6th Australasian Computer Systems Architecture Conference,* pp.62-70, 2001.
[13] C. Hansen, "Architecture of broadband media-processor," *IEEE COMPCON '96,* pp.25-29, February, 1996.
[14] P. Kalapathy, "Hardware-software interactions on Mpact," *IEEE Micro,* Vol. 17, No. 2, pp.20-26, March/April 1997.
[15] J. L. Baer and T. F. Chen, "An effective on-chip preloading scheme to reduce data access penalty," *Proc. Supercomputing '91,* pp.176-186, 1991.
[16] T. F. Chen and J. L. Baer, "Effective hardware-based data prefetching for high-performance processors," *IEEE Trans. Computers,* Vol. 44, No. 5, pp.609-623, May, 1995.
[17] A. J. Smith, "Cache memories," *ACM Computing Surveys,* Vol. 14, pp.473-530, September, 1982.

문 현 주
1995년 충북대학교 컴퓨터과학과 학사취득. 1997년 충북대학교 전자계산학과 석사취득. 1999년 충북대학교 전자계산학과 박사과정 수료. 관심분야는 멀티미디어 프로세서, 병렬처리 컴퓨터 구조, 병렬처리 알고리즘 등

전 중 남
1981년 연세대학교 전자공학과 학사 취득. 1985년 연세대학교 전자공학과 석사취득. 1990년 연세대학교 전자공학과 박사취득. 1990년 ~ 현재 충북대학교 컴퓨터과학과 교수로 재직 중. 1999년 ~ 현재 충북대학교 컴퓨터과학과 학과장으로 재직 중. 관심분야는 컴퓨터 구조, 병렬처리 알고리즘, 임베디드 시스템 등

김 석 일
1975년 서울대학교 전기공학과 학사 취득. 1975년 ~ 1990년 국방과학연구소 선임연구원으로 근무. 1985년 ~ 1989년 미국 North Carolina State University 에서 공학박사 취득. 1990년 ~ 현재 충북대학교 전기전자컴퓨터공학부 교수로 재직 중. 2002년 ~ 현재 충북대학교 전지전자컴퓨터공학부 학부장으로 재직 중. 관심분야는 병렬처리 컴퓨터 구조, 슈퍼컴퓨팅, 병렬처리 언어, 이기종 분산처리, 시각장애인 사용자 인터페이스 등임