독립적인 미디어 캐시를 채용한 프로세서의 성능향상

주 영 관*, 전 중 남, 김 석 일
충북대학교 컴퓨터과학과
juyg*@john.chungbuk.ac.kr

{joongnam, ksi}@cbucc.chungbuk.ac.kr

Performance Improvement of A Processor with Independent Media Cache

Youngkwan Ju*, Joongnam Jeon, Sukil Kim
Dept. of Computer Science, Chungbuk National University

요 약

본 논문에서는 미디어 프로세서에서 메모리 참조시 평균 메모리 참조 지연시간을 줄이기 위하여 지역성이 높으나 재사용성이 떨어지는 미디어 데이터를 지역성과 재사용성이 높은 일반 데이터로부터 분리하여 별도의 캐시에 저장하도록 하는 캐시 구조를 제안하였다. 또한, 미디어 데이터의 선인출 기법을 캐시 운영 전략으로 채택하도록 하여 평균 메모리 지연시간을 단축하였다. EPIC, JPEG 벤치마크에 대한 실험 결과, 미디어 데이터를 일반 데이터 캐시와 구분한 이중캐시 구조가 하나의 캐시에 모든 데이터를 저장하는 단일캐시구조에 비하여 캐시미스횟수가 감소하였음을 확인할 수 있었다.

1. 서 론

멀티미디어 응용 프로그램은 미디어 데이터를 스트리밍 패턴으로 참조하는 특성이 있다. 따라서 미디어 데이터는 재사용성이 떨어지는 대신 지역성이 좋은 특징이 있다[1,2]. 또한, 근래에 들어와 멀티미디어 응용 프로그램이 실시간 비디오 정보를 처리하게 되면서 처리하는 데이터의 용량이 매우 방대하다.

이들 미디어 데이터를 기존의 캐시에 저장하는 경우에 미디어 데이터로 인하여 재사용성이 높은 일반 데이터가 미디어 데이터의 반복된 로딩에 의하여 캐시로부터 제거하게 되는 현상이 발생하게 된다. 이러한 현상은 결국 일반 데이터의 캐시 미스율을 증가하도록 만들어 궁극적으로는 캐시의 성능을 떨어트리게 된다[13].

미디어 데이터에 의한 캐시의 성능 저하 현상을 방지하기 위하여 메모리와 캐시 사이에 버퍼를 두는 방법도 제안되었다[9]. 즉, 미디어 데이터를 참조하는 경우에 참조한 메모리 블록을 캐시로 적재하지 않고 버퍼에 저장하고 실제로 필요한 부분만 캐시로 옮기는 것이다. 따라서 이 방법은 사용되지 않는 미디어 데이터가 불필요하게 캐시를 점유하지 않도록 하는 장점이 있다. 그러나 메모리 블록이 버퍼의 크기를 넘는 경우에 버퍼는 캐시에서와 같은 버퍼 블록의 교체 전략을 하드웨어로 구현해야 하는 단점이 있다. 또한, 버퍼를 채용하는 것은 버퍼의 운용 방식이 캐시와 동일하므로 결국 버퍼를 사용하는 방식도 캐시의 크기를 늘리는 것과 효과가 같다.

본 논문에서는 미디어 데이터와 일반 데이터가 참조특성이 다른 점에 착안한 이중 캐시 구조를 제안하였다. 즉, 미디어 데이터는 미디어 데이터 캐시에만 저장하며, 일반 데이터는 일반 데이터 캐시에만 저장하도록 하여 각 데이터의 참조 특성을 살리도록 하여 평균메모리 참조시간을 줄이도록 하였다.

본 논문의 구성은 다음과 같다. 제 2장에서는 미디어 프로세서와 관련한 메모리 구조와 선인출 기법을 고찰하였다. 제 3장에서는 두개의 캐시로 구성된 본 논문에서 제안하는 이중캐시 구조를 설명하고, 이중캐시 구조를 위한 캐시 운영 전략을 토의하였다. 제 4장에서는 여러 가지 벤치마크에 대한 성능 실험을 수행하였다. 실험에서는 ATOM[7]과 DineroIII[8]를 이용하여 캐시 구조에 따른 벤치마크의 캐시미스횟수를 측정하고 이를 비교하였다. 마지막으로 제 5장에서는 본 논문의 결론을 도출하였다.

2. 관련연구

2.1 미디어 프로세서 구조

초기의 멀티미디어 프로세서 구조는 응용 프로그램의 데이터 참조 특성을 고려하지 않고 연산 속도를 빠르게 하는 방법에 초점이 맞추어져 있었다. 즉, 미디어 연산처리기를 파이프라인으로 구성하고 클록 속도를 높였으며, 빠른 메모리와 캐시를 사용하도록 하였다. 또한 하나의 명령어로 여러 개의 데이터를 처리하도록 하는 서브워드 병렬성[10]와 여러 개의 연산처리기로 프로세서를 구성하는 방법이 제안되었다. 뿐만 아니라 프로세서 칩 내에 장착하는 온 칩(on-chip) 캐시의 크기가 제한적이므로 프로세서 외부에 별도의 캐시를 두기도 한다.

그러나 멀티미디어 응용 프로그램은 스트리밍 형태로 데이터를 참조하는 특성이 있어 캐시를 계층화하거나 캐시의 크기를 증가시키더라도 각 캐시 계층에서 충분히 활용되지 못하고 주기적인 블록 교체를 유발하므로 캐시 계층 구조의 효과가 낮은 특징이 있다.[13].

대표적인 초기 미디어 프로세서인 TriMedia TM-1000에서는 미디어 데이터의 캐시 오염을 방지하기 위하여 참조된 미디어 데이터 블록을 캐시에 저장하지 않는다[6]. 즉, 미디어 데이터와 일반 데이터를 컴파일 시간에 구분하여 서로 다른 메모리 영역에 적재하도록 하고 미디어 데이터 영역에 적재된 데이터를 참조할 때에는 캐시를 우회하여 연산처리기로 제공되도록 하고 일반 데이터의 경우에만 캐시를 경유하여 연산처리기에 제공되도록 하였다. 따라서 TM-1000에서는 미디어 데이터가 캐시를 오염시키는 현상을 방지할 수 있다. 그러나 이 구조는 참조하는 미디어 데이터가 캐시를 사용하지 않으므로 평균 메모리지연을 낮추

본 연구는 한국과학재단 목적기초연구(R05-2002-000-01470-0)지원으로 수행되었음.

려면 메모리 대역폭을 높여야만 하므로 메모리구성비용이 높아진다. 본 논문에서는 TM-1000의 이런한 문제점을 해결하기 위하여 메모리 대역폭을 높이는 대신 메모리와 연산처리기간에 미디어 캐시를 두는 구조를 제안하였다.

2.1 미디어 데이터 선인출 기법

스트리밍 데이터를 구성하는 자료 구조는 배열 형태가 일반적이다. 또한, 이들 배열의 처리는 프로그램의 반복 구조로 표현되므로 스트리밍 데이터 참조는 규칙성이 존재한다[1, 12]. 따라서 이러한 스트리밍 데이터의 규칙성을 활용하면 프로그램의 실행 중에도 필요한 스트리밍 데이터를 선인출 할 수 있어서 캐시의 효과를 높일 수 있다.

가장 간단한 선인출 기법은 어떤 메모리 블록을 참조할 때에 이 블록과 이웃한 몇 개의 블록을 선인출하는 방법이다[14]. 이 방법은 선인출할 블록들의 주소가 실제로 참조하려는 블록의 주소와 이웃하고 있으므로 선인출 주소를 결정하는 메모리 제어기의 구성이 매우 간단한 장점이 있으나 응용 프로그램이 메모리 블록들을 순서대로 참조하는 경우에만 효과적이다. 예를 들어 몇 개의 블록을 건너뛰면서 참조하는 패턴을 보이는 응용 프로그램의 경우에는 선인출된 메모리 블록들이 모두 캐시에 저장되나 더 이상 프로그램이 사용하지 않는 문제, 즉 캐시 오염(cache pollution)을 유발하는 문제가 발생한다. 이러한 현상은 동시에 몇 개의 메모리 블록을 선인출하는 경우에 보다 심각한 문제를 유발하므로 한 개의 메모리 블록을 선인출하는 것이 일반적이다.

Chen[11]이 제안한 선인출 기법은 프로그램 내의 특정한 주소의 명령어가 반복해서 참조하는 메모리 주소를 기억하였다가 이 주소가 일정한 차이를 두고 변화하는 경우에 이를 예측하여 메모리 블록을 캐시로 적재하는 기법이다. 일반적으로 미디어 데이터는 루프에 의하여 반복적으로 참조되는 프로그램 특징을 지니고 있으므로 미디어 데이터를 참조하는 명령어 마다 명령어의 적재 주소를 기억해두고 이 명령어가 데이터를 참조할 때마다 먼저 번에 명령어가 실행될 때에 참조하였던 데이터 주소와 비교하여 두 데이터 주소의 차이, 즉 거리(stride)를 계산하고, 이 차이를 현재 참조하려는 데이터의 주소에 더하여 앞으로 참조할 것으로 예상되는 데이터의 선인출 주소를 계산하는 것이다. 따라서 이 기법은 일정한 패턴으로 메모리를 참조하는 스트리밍 데이터의 처리 시에 매우 유용하다.

Moon[13]은 Chen의 기법을 2차원으로 확장하여 배열을 서브블록으로 나누어 블록 단위로 계산하는 경우에도 선인출 예측 에러를 줄이는 기법을 제안하였다. 즉 Chen의 기법에서 열과 행이 모두 바뀌는 경우 발생하는 선인출 예측 에러를 열과 행이 모두 바뀌는 경우에는 Stride의 변화를 예측하여 선인출 조소를 예측하도록 한다.

본 논문에서는 단일캐시구조와 이중캐시구조에서 여러 가지 선인출 기법을 적용하였을 때 캐시미스의 횟수를 측정하여 캐시 구조와 선인출 기법에 따른 성능을 비교하였다.

3. 이중 캐시 구조

하나의 캐시에 멀티미디어 데이터와 일반 데이터를 함께 적재하는 단일 캐시 구조에서, 멀티미디어 데이터를 연속해서 캐시에 적재할 경우 캐시 블록의 교체(replacement)가 발생한다. 이 때, 이전에 사용된 멀티미디어 데이터 블록은 더 이상 재사용될 가능성이 없음에도 불구하고 최근에 사용된 데이터 블록이므로 재사용의 가능성이 충분한 일반 데이터 블록을 교체하게 될 가능성이 높다.

따라서 하나의 캐시 모듈에 멀티미디어 데이터와 일반 데이터를 함께 적재하는 기존의 구조에서는 캐시 오염문제가 대두되므로 만족할 만한 성능을 얻을 수 없다. 이러한 문제를 해결하기 위해서는 캐시 내에 멀티미디어 데이터와 일반 데이터를 저장하는 독립된 캐시를 두고 각 데이터별로 캐시를 운영하는 전략이 필요하다. 그림 1은 본 논문에서 고려하는 캐시 구조이다.

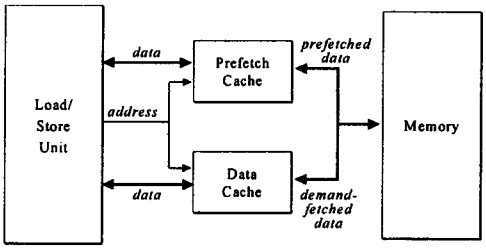


그림 1 이중 캐시 구조

그림1에서 프리페치 캐시(Prefetch Cache: PC)는 미디어 데이터를 저장하는 캐시로 메모리의 특정 영역에 저장된 미디어 데이터의 참조 시에 이 블록을 저장한다. 데이터 캐시(Data Cache: DC)는 일반 데이터를 저장하는 캐시로 PC와 마찬가지로 메모리의 특정영역(본 논문에서는 일반 데이터 영역)에 저장된 데이터를 저장하는 캐시이다. Load/Store Unit는 load 또는 store 명령어를 실행할 때에 참조할 메모리 블록의 주소를 캐시 제어기로 전달한다. 캐시 제어기는 메모리 블록의 주소를 분석하여 미디어 데이터가 저장된 영역인지를 확인하여 만일 미디어 데이터 영역의 데이터를 참조하는 경우에는 선인출 명령어를 발생하도록 한다. 일반 데이터 영역일 경우에는 기존의 캐시와 마찬가지로 DC에 저장하게 된다.

또한, PC와 DC의 운영전략은 다음과 같다. 즉, DC의 경우에는 DC가 기존의 캐시와 같이 일반 데이터를 저장하므로 LRU를 사용하는 것으로 간주하였다. PC의 경우에는 미디어 데이터가 지역성은 높은 대신, 재사용성이 낮은 특성을 고려하여 FIFO 방식으로 운영하도록 하였다. 즉, PC에서는 먼저 적재된 캐시 블록이 먼저 캐시에서 제거된다.

4. 실험 및 분석

4.1 실험 환경

본 논문에서 제안한 이중 캐시 구조와 단일 캐시 구조의 성능을 비교를 위하여 여러 가지 미디어 벤치마크를 대상으로 실험을 수행하였다. 우선 메모리 참조 트레이스를 얻기 위해서는 ATOM 시뮬레이터를 이용하였다. 즉, 벤치마크 프로그램의 메모리 참조 트레이스를 생성하였다. 생성된 메모리 참조 트레이스는 Load 또는 Store 명령어의 실행에 필요한 물리적 번지들이다.

또한, 제안한 캐시 구조와 전략의 효과를 확인하기 위하여 메모리 참조 트레이스로부터 캐시의 사용 과정을 모의하는 캐시 시뮬레이터인 CASIM을 구현하였다. 본 논문에서 구성한 캐시 시뮬레이터는 위스콘신 대학에서 개발한 트레이스 구동 방식의 시뮬레이터인 Dinero III[8]를 바탕으로 구현하였다. 즉, 캐시 시뮬레이터는 ATOM형 실행코드로부터 생성된 메모리 참조 명령어 트레이스를 입력받아 load/store 명령어별 메모리 참조 회수와 캐시 미스율 등의 결과를 산출한다. 이 과정에서 각 벤치마크 별로 캐시 크기, 캐시 블록 크기, 연관도, 블록 교체 정책 등을 매개변

수로 지정하여 다양한 캐시 구조에 대한 모의실험을 수행할 수 있다. 본 논문에서 사용한 벤치마크는 표 1과 같다.

표 1 실험에 사용된 벤치마크 프로그램

	description	program module	data source	input size	image/frame size
JPEG	흑백 및 칼라 이미지를 위한 표준 압축 방식	cjpeg	image files of .ppm format	100K bytes	172×189
		djpeg	image files of .jpg format		
EPIC	실험용 이미지 압축 도구	epic	image files of .raw format	60K bytes	256×256
		unepic	image files of .E format		

4.2 실험결과 및 성능 분석

본 논문에서는 선인출을 활용하지 않는 단일 캐시 구조(ORG), 미스 발생시 한 블록을 선인출하는 기법(OBL), 규칙 선인출 기법을 채용한 구조(RPT), 블록 선인출 기법을 채용한 기법(BRPT) 별로 단일 캐시를 사용하는 경우와 미디어 데이터와 일반 데이터용 캐시를 각각 사용하는 이중캐시에 대하여 캐시 미스 횟수를 조사하였다. 여기서 이중캐시 구조의 PC는 4Kb와 8Kb 이 두 가지 경우를 실험하였으며 DC는 2Kb에서 256Kb까지 변화시키면서 캐시미스를 측정하였다. 또한 이중캐시구조를 구성하는 캐시의 매핑 방식은 직접사상(direct mapping)이라고 가정하였으며, 캐시블록의 크기는 32바이트라고 가정하였다. 실험에서 선인출된 메모리블록을 참조할 경우에 메모리블록이 캐시에 적재가 완료된 상태일 경우에는 캐시 적중, 적재중인 경우에는 캐시 미스를 구분하였다.

그림 2, 3에서 이중캐시구조에서의 실험결과(obl_d, rpt_d, erpt_d)가 단일 캐시구조에서의 실험결과(obl_s, rpt_s, erpt_s)에 비하여 캐시미스의 횟수가 줄어든 것을 보여준다. 여기서 org는 단일 캐시 구조에서 아무런 선인출 장치를 사용하지 않았을 때를 의미한다. obl_s와 obl_d는 각각 OBL기법을 단일캐시구조와 이중캐시구조에 적용했을 경우이다. rpt_s와 rpt_d는 각각 Chen의 RPT기법을 단일캐시구조와 이중캐시구조에 적용한 경우이다. erpt_s와 erpt_d는 각각 Moon이 제안한 BRPT기법을 단일 캐시구조와 이중캐시구조에 적용한 결과이다. 그림2과 3에서 공히 BRPT방법을 이중캐시구조에 적용한 캐시미스발생횟수가 가장 작음을 알 수 있다. 그러나 CJPEG의 경우는 PC가 4Kb일 때 DC가 4Kb~8Kb인 범위에서 OBL기법이 BRPT에 비하여 성능이 우수한 것을 보여 준다. 이는 CJPEG의 경우에 이웃한 메모리 블록이 연속적으로 사용되는 특성 때문인 것으로 판단된다. 이러한 현상은 PC가 8Kb일 때 DC가 2Kb~8Kb인 경우에도 발생하는 것을 알 수 있다. 그러나 DC가 8Kb이상인 경우에는 이중캐시구조에서 BRPT기법을 적용한 경우가 가장 성능이 우수함을 알 수 있다.

특히, 모든 실험에서 이중캐시구조의 경우가 단일캐시구조인 경우에 비하여 우수 하였으며 DC의 크기가 클수록 그 현상이 확실히 나타나는 것을 확인할 수 있었다.

5. 결론

본 논문에서는 멀티미디어 응용 프로그램의 수행 성능에 큰 영향을 미치는 메모리 참조 지연 시간을 줄이기 위하여 독립적인 미디어 캐시를 사용하는 캐시 구조를 제안하였다. 제안한 구조는 미디어 데이터와 일반 데이터 별로 데이터가 메모리의 특정영역에 저장되어 있다고 가정하고 특정영역에 존재하는 데이터를 참조할 경우에는 각각 지정된 캐시를 사용한다. 또한, 미디어 데이터의 경우에는 지역성이 좋은 점과 메모리 참조 패턴이 균일한 점을 활용하여 선인출 기법을 적용하였다. 실험 결과, 미디어 데이터를 저장하는 캐시를 독립적으로 운영하고 선인출 기법을 채용하는 캐시 구조가 같은 용량의 단일 캐시를 사용하는 경우보다 캐시 미스의 횟수가 크게 줄어 든 것을 알 수 있었다.

6. 참고문헌

[1] M. E. Wolf and M. S. Lam, "A Data Locality Opitmizing Algorithm," *Proceedings SIGPLAN'91 Conf. Programming Language Design and Implementation*, pp.30-44. June, 1991

[2] S. Carr, K. S. McKinley and C. W. Tseng, "Compiler Optimization for Improving Data Locality," *Proceedings 6th Int. Conf. Architectural Support for Programming Languages and Operating Systems*, pp. 252-262, October, 1994.

[3] S. P. VanderWiel, D. J. Lilja, "When Caches Aren't Enough: Data Prefetching Techniques," *IEEE Computer*, pp.23-20, July, 1997.

[4] J. R. Goodman, *Cache Consistency and Sequential Consistency*, Technical Report TR-1006, University of Wisconsin-Madison, February, 1991.

[5] M. Tremblay and J. M. O'Conner, "UltraSparc I : A Four-Issue Processor Supporting Multimedia," *IEEE Micro*, Vol. 16, No. 2, pp.42-50, April, 1996.

[6] N. Mitchell, "Philips TriMedia: A Digital Media Convergence Platform," *Proceedings Wescon'97*, pp.56-60, 1997.

[7] A. Srivastava and A. Eustace, "ATOM: A System for Building Customized Program Analysis Tools," *Proceedings ACM SIGPLAN 94*, 196-205, 1994.

[8] M. D. Hill, *Dinero III Cache Simulator*, Technical Report, Computer Sciences Department, University of Wisconsin-Madison.

[9] N. P. Jouppi, "Improving Direct-mapped Cache Performance by the Addition of a Small Fully associative Cache and Prefetch Buffers," *Proceedings 17th Int. Symp. Computer Atchitecture*, pp.364-373, May 1990.

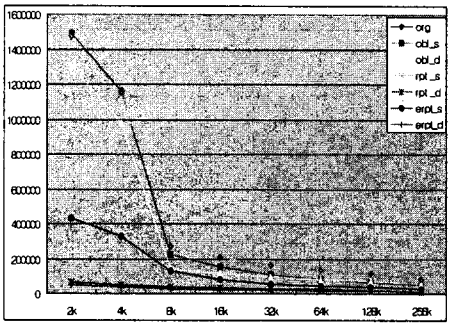
[10] 문현주, 전중남, 김석일, "복수 캐시로 구성한 미디어 프로세서의 설계," *정보과학회 논문지*, 제29권, 제9호, pp.573-581, 2002년 10월.

[11] D. A. Koufaty, X. Chen, D. K. Poulsen and J. Torrellas, "Data Forwarding in Scalable Shared-Memory Multiprocessors," *Proceedings 9th Int. Conf. Supercomputing*, pp.255-264, July, 1995.

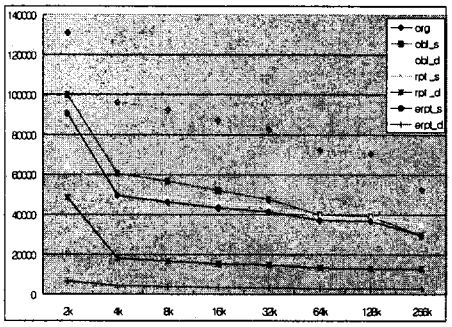
[12] C. K. Luk, *Optimizing the Cache Performance of Non-Numeric Applications*, Ph. D. Thesis, University of Toronto, 2000.

[13] Hyunju Moon, *A Cache Managing Strategy for Fast Media Data Access*, Ph. D. Thesis, Dept. of Computer Science, Chungbuk National University , Feb 2003.

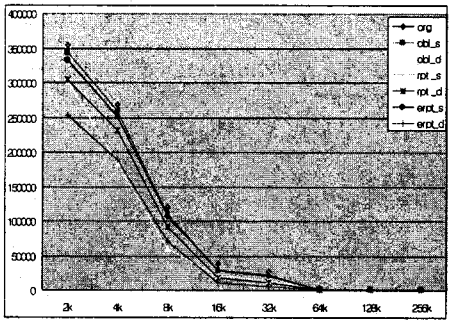
[14] Alan Jay Smith, "Cache Memories", *ACM Computing Surveys*, Vol14,pp.473-530, September, 1982.



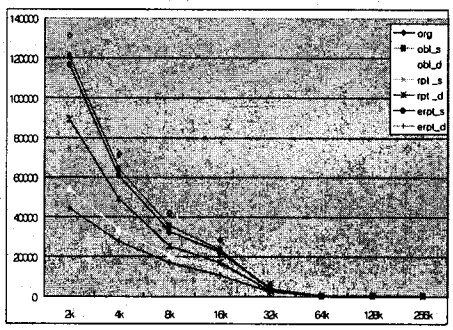
(a)EPIC



(b)UNEPIC

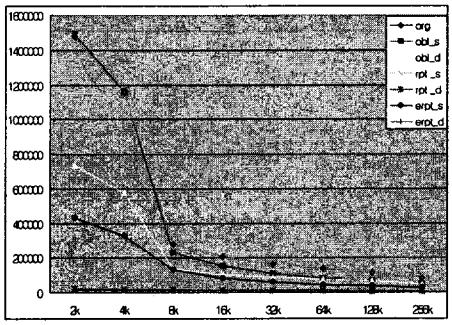


(c)CJPEG

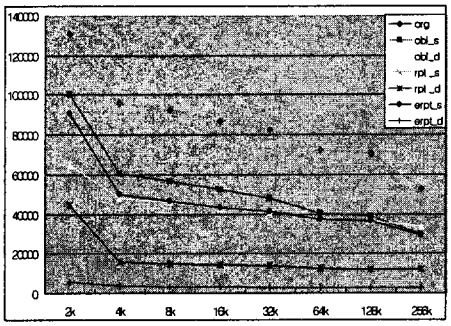


(d)DJPEG

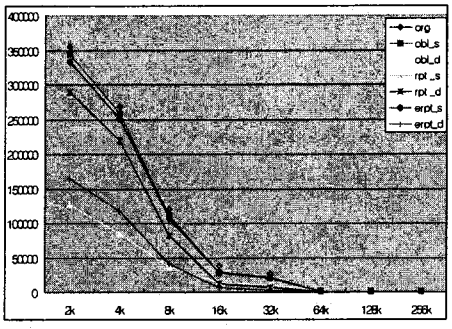
그림 2 캐시크기변화에 따른 캐시미스(PC=4KB)



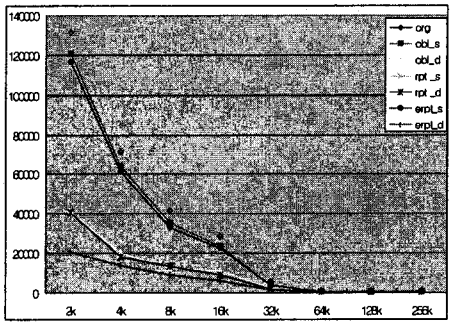
(a)EPIC



(b)UNEPIC



(c)CJPEG



(d)DJPEG

그림 3 캐시크기변화에 따른 캐시미스(PC=8KB)