고성능 프로세서-메모리 혼합 구조의 설계 및 성능 분석

김 영 식† · 김 신 덕†† · 한 탁 돈†††

요 약

프로세서 메모리 혼합 구조는 해마다 증가하는 프로세서와 메모리간의 성능 격차를 해결하는 대안으로 연구가 활발히 진행되고 있다. 본 논문에서는 프로세서 메모리 혼합 구조의 여러 가지 설계 대안들을 고찰하였다. 이를 위해서 DRAM 접근 시간의 분석적 모델을 제안하고 성능 향상점 및 성능 병목점을 찾았다. 제안한 분석적 모델에 의하여 DRAM 페이지 적중률을 증대하여 성능을 향상시키는 구조로써 새로운 온칩 DRAM 구조인 프리차지 연기 뱅크 아키텍처를 제안하였다. 또한 제안한 뱅크 아키텍처에 효율적으로 적용할 수 있는 뱅크 인터리빙 방법을 제시하였다. 제안한 구조는 기존의 일반적 DRAM 구조 및 계층적 다중-뱅크 구조보다 우수함을 시뮬레이션을 통하여 증명하였다. 시뮬레이션은 SimpleScalar 툴을 개조하여 사용하였고, SPEC95 벤치마크에 대해서, 캐쉬 메모리의 크기, 뱅크 개수, 프리차지 연기 시간 등의 변화에 대한 성능을 분석하였다.

Design and Performance Analysis of High Performance Processor-Memory Integrated Architectures

Young-Sik Kim† · Shin-Dug Kim†† · Tack-Don Han†††

ABSTRACT

The widening performance gap between processor and memory causes an emergence of the promising architecture, processor-memory (P-M) integration. In this paper, various design issues for P-M integration are studied. First, an analytical model of the DRAM access time is constructed considering both the bank conflict ratio and the DRAM page hit ratio. Then the points of both the performance improvement and the performance bottle neck are found by the proposed model as designing on-chip DRAM architectures. This paper proposes the new architecture, called *the delayed precharge bank architecture*, to improve the performance of memory system as increasing the DRAM page hit ratio. This paper also adapts an efficient bank interleaving mechanism to the proposed architecture. This architecture is verified to be better than the hierarchical multi-bank architecture as well as the conventional bank architecture by execution driven simulation. Eight SPEC95 benchmarks are used for simulation as changing parameters for the cache architecture, the number of DRAM banks, and the delayed time quantum.

* 이 논문은 1997년 학술진흥재단의 신진연구인력 연구장려금 지원 연구비에 의하여 연구되었음.
이 논문은 삼성전자 지원 연구비에 의하여 연구되었음.

† 준 회 원 : 연세대학교 대학원 컴퓨터과학과
†† 정 회 원 : 연세대학교 컴퓨터과학과 교수
††† 종신회원 : 연세대학교 컴퓨터과학과 교수

논문접수 : 1998년 5월 27일, 심사완료 : 1998년 7월 16일

1. 서 론

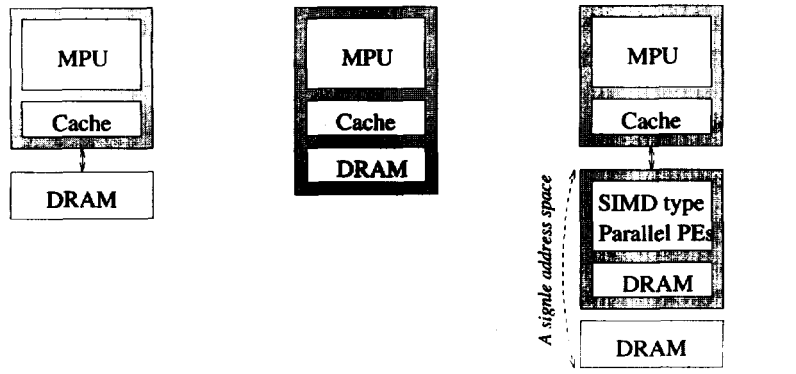
프로세서와 메모리 설계 기술은 속도와 집적도 중심으로 서로 다르게 발전하여서, 프로세서와 메모리간의 속도 면에서 성능 격차는 해마다 증가하고 있다. 그런데 메모리 집적 설계 기술은 많은 응용 분야의 컴퓨터 시스템에서 요구하는 메모리 용량보다 큰 용량을 단일 칩으로 집적할 수 있는 수준으로 발전하리라 예상된다. 따라서 해마다 증가하는 프로세서 메모리 성능 격차를 극복하는 해결책으로 온칩 DRAM을 가지는 프로세서-메모리 (p-m) 혼합 구조 연구가 활발히 진행되고 있다[1]-[14].

컴퓨터 그래픽을 위해서 DRAM과 소량의 로직을 3-D DRAM [3] 칩으로 집적되었다. 낮은 메모리 대역폭(bandwidth) 문제를 극복하기 위해서, 단일 프로세서와 메모리가 단일 칩으로 집적하려는 연구로써 미국 버클리대학의 IRAM 프로젝트 [1], 미쯔비시의 M32R/D [4], 선 S3mp 프로젝트 [12] 등이 있고, 다중 프로세서들로는 일본 규슈대학의 PPRAM 프로젝트 [2,3, 14] 미국 스탠포드대학의 Hydra 프로젝트 [10,11] 등이 있다. 또한 C-RAM [5], PIM [6], MPA [7,8,9] 같이 SIMD 프로세서들과 자신의 지역 메모리를 단일 칩으로 집적하려는 연구들도 있었다.

p-m 혼합 구조는 높은 내부 메모리 대역폭과 낮은 메모리 지연(latency)을 제공하고 오프칩 버스를 구동하지 않아서 저전력의 장점을 가진다. 또한 일반 DRAM은 세대에 따라서 2의 승수로 크기가 증가하지만 온칩 DRAM은 응용 문제에 따라서 크기와 형태를 마음대로 조절할 수 있다. 그리고 칩의 개수가 줄어서 보드 크기도 감소 할 수 있다. 이와 같은 장점에 의하여 향후 p-m 혼합 구조는 저전력을 가져야 하고 메모리 확장의 요구가 적고 작은 보드 크기를 필요 하는 휴대용 정보통신 기기부터 우선적으로 채택되리라 예상된다.

그러나 이와 같은 장점을 가지기 위해서는 해결해야할 많은 문제점을 가진다. 우선 DRAM 공정 기술은 집적도를 중심으로 발전하였고, 프로세서 공정 기술은 속도 중심으로 발전하였기 때문에 p-m 혼합 구조는 집적도와 속도를 모두 만족하는 새로운 공정 기술을 개발하여야 한다. 두 번째는 높은 내부 메모리 대역폭과 낮은 메모리 지연을 얻기 위해서는 기존의 DRAM 구조만으로는 어렵기 때문에 구조를 변경해야하는 데 그에 따라 전력과 칩 크기의 오버헤드(overhead)를 겪게된다. 그리고 DRAM과 함께 집적된 로직에서 발생하는 열에 의해 메모리 셀의 데이타 보유 시간이 감소하고 리프레시(refresh) 비율이 증가하므로 리프레시 구조도 바뀌어야 한다. 또한 p-m 혼합칩의 상업적 마켓과 테스팅 문제도 해결해야한다.

그림 1은 (a) 일반적으로 프로세서와 주메모리 칩이 분리된 시스템을 나타내고 (b) 와 (c)는 p-m 혼합 구조의 개념을 나타낸다. 특히 그림 1 (c)의 DRAM 기반 코프로세서는 p-m 혼합 구조의 여러 가지 해결해야할 문제점 때문에 장기적 해결책이 아닌 단기적 해결책으로써 연구가 진행되고 있는 구조이다 [3,5,6,7,8,9]. 본



(a) Conventional system (b) P-M integrated system (c) DRAM based coprocessor

(그림 1) p-m 혼합 구조의 설계 대안
(Fig. 1) Design alternatives for P-m integrated architectuses

연구진은 DRAM 기반 코프로세서 연구로써 MPA (memory based processor array) [7,8,9] 구조를 연구하고 있다. MPA 시스템은 데이타 병렬 응용에 적합한 구조로써, 호스트 프로세서가 가지는 일반 DRAM 메모리와 단일 주소 공간을 가지기 때문에, 호스트 프로세서와의 데이타 통신을 간단한 메모리 읽기/쓰기로 해결할 수 있다.

그런데, 본 논문에서는 DRAM 기반 코프로세서가 아닌 장기적 해결책으로써 그림 1 (b)의 구조를 목표로, 높은 메모리 대역폭과 낮은 메모리 지연을 얻기 위한 온칩 DRAM 아키텍처를 설계하는 데 있어서 고려해야할 설계 대안(design alternative) 및 구현 이슈(issue)를 시스템적으로 분석하여 응용 시스템에 적합한 새로운 온칩 DRAM 아키텍처를 설계하고 검증한다.

이어지는 절에서는 온칩 DRAM의 설계 이슈를 고찰하고 성능 척도로써 DRAM 접근 시간에 대한 분석적 모델을 개발하였다. 이 모델을 기반으로 온칩 DRAM의 성능이 어느 정도 개선될 수 있는 지를 예측한다. 3절에서는 2절에서 제시한 설계 이슈 및 분석적 모델에 의해서 온칩 DRAM의 성능을 개선할 수 있는 여러 가지 설계 대안들을 살펴보고 보다 우수한 구조를 제안한다. 4절에서는 3절에서 살펴본 여러 가지 설계 대안 및 제안한 구조의 시뮬레이션을 통한 성능 검증을 수행한다. 5절에서는 이 논문의 결론을 맺는다.

2. 시스템 모델

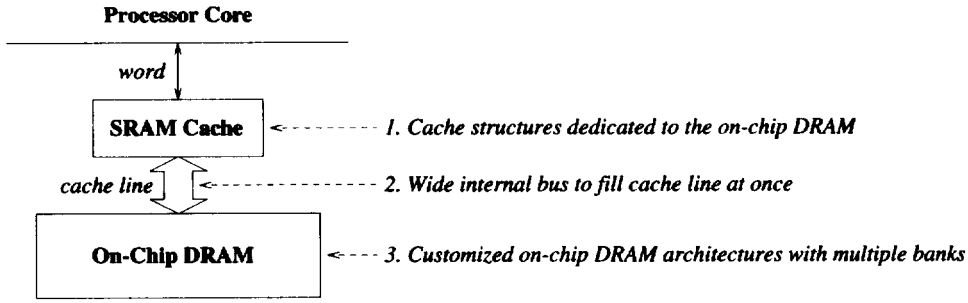
본 논문에서는 온칩 DRAM을 가지는 프로세서 아키텍처에서 성능 증대 효과를 얻을 수 있는 계층적 메모리 설계 대안에 초점을 맞춘다. 메모리 시스템 설계의 목적은 단위 시간 동안 접근할 수 있는 메모리 양 즉 메모리 대역폭을 증가시키는 것이다. 이를 위해서는 메모리 대역폭을 프로세서 대역폭과 버스 대역폭에 맞춰주어야 한다.

2.1 설계 이슈

그림 2는 기본적인 시스템 모델 및 설계 이슈를 표시하는 데, 프로세서 코어와 온칩 L1 캐쉬, 온칩 DRAM 주메모리를 가지는 시스템 모델을 가정한다. 그러나 일반적인 DRAM 공정은 효율적인 프로세서를 구성하는 데 적합하지 않다. 왜냐하면 16Mb DRAM 이전까지는 1 메탈 층만을 사용하였기 때문이다. 적어도 256Mb 세대인 0.25um 2-3 메탈 층을 가지는 공정이라야 100-200MHz CPU 로직을 구성할 수 있다. 또한 p-m 혼합 구조의 가격 대 성능비가 우수하기 위해서는 CPU 코어는 DRAM 다이 크기의 10-15%정도이어야 한다. 0.25um 공정 256Mb DRAM 다이의 10% 정도는 30mm^{2}이다. 이 정도의 다이 크기는 작은 크기의 슈퍼스칼라(superscalar) 프로세서 (예를 들어 MIPS R4xxx)정도이다.

목표 시스템 모델은 2-3 메탈 이상의 CMOS 공정을 가지는 256Mb DRAM 이상으로 하고 CPU 코어는 이러한 DRAM 공정에 맞추어 전체 DRAM 다이 크기의 10-15%를 차지하는 작은 크기의 슈퍼스칼라 프로세서를 목표로 한다.

가정한 시스템 모델에서 온칩 DRAM을 가지게 되면 메모리 시스템은 크게 다음 그림 2에서와 같이 구조가 바뀔 수 있어서 성능 개선을 얻을 수 있다. 우선 그림 (1)에서처럼 온칩 DRAM에 적합하도록 변형된 캐쉬 설계를 채택할 수 있으며 이의 예로써 SUN 컬럼 버퍼(column buffer) 캐쉬 [12]와 일본 규슈대학의 가



(그림 2) DRAM-프로세서 혼합 구조에 대한 설계 이슈
(Fig. 2) Design issues for DRAM-processor integration

변적 라인 크기(VLS: Variable Line-Sized) 캐쉬 [14]가 있다. 또한 (2) 메모리 버스 넓이의 핀(pin) 제약이 없어지므로 써 캐쉬 라인 넓이까지 증대되어 메모리 전송 시간을 획기적으로 줄일 수 있고, 나아가서 온칩 메모리 버스의 클럭 속도의 제약도 완화될 수 있다. 그리고 (3) 온칩 DRAM 뱅크 아키텍처를 계층적 다중-뱅크 DRAM [10,11]과 같이 설계하여 성능을 올릴 수 있다.

이와 같이 온칩 DRAM을 가지는 컴퓨터 시스템에서의 성능 증대를 이룰 수 있는 설계 대안들을 고찰하고 새로운 구조 제안 및 검증을 수행한다.

2.2 분석적 모델

본 절에서는 각 설계 이슈들에 대한 설계 방법들의 성능을 비교하기 위한 척도로써 DRAM 접근 지연 $T_{access_latency}$를 정의한다.

우선 DRAM 접근 지연의 분석적 모델을 구성하기 위해서 필요한 변수들을 정의한다.

변수

- F_{CPU} [MHz]: CPU 클럭 주파수
- F_{BUS} [MHz]: 메모리 버스 클럭 주파수
- B [bytes/seconds]: 메모리 버스 대역폭
- L [bytes]: 캐쉬 라인 크기
- W [bytes]: 메모리 버스 넓이
- i: 인덱스 $(0 \le i \le n-1)$
- Q_i: i번째 DRAM 뱅크의 서비스를 기다리는 DRAM 요구 큐에 있는 요구들의 개수.
- α_i: DRAM 요구가 i번째 DRAM 뱅크에서 서비스 받을 확률.
- β_i: DRAM 요구가 i번째 DRAM 뱅크에서 서비스 받고 있는 이전 요구와 충돌을 일으킬 확률, 즉 뱅크 충돌률.
- γ_i: 요구하는 데이타를 i번째 DRAM 뱅크의 감지 증폭기(sense amplifier)가 이미 적재하고 있을 확률, 즉 DRAM 페이지(page) 적중률.

Q_i는 $(1-\beta_i)\times 0+\beta_i\times(Q_i+1)$으로 나타낼 수 있으므로 $Q_i=\beta_i/(1-\beta_i)$이다.

캐쉬 메모리를 가지는 컴퓨터 시스템 아키텍처에서 $T_{access\_latency}$[cpu cycles]는 캐쉬 실패 발생 시의 실패 비용을 1 캐쉬 라인 채우기 시간으로 정의하면 다음과 같이 구성할 수 있다.

$$T_{access\_latency} = T_{DRAM\_access} + T_{transfer} + T_{overhead} \quad (1)$$

$T_{transfer}$ [cpu cycles]는 1 캐쉬 라인의 데이타를 DRAM 뱅크와 L1 캐쉬 사이에서 메모리 버스를 통해 전송하는 시간을 의미하며 다음과 같은 식으로 표현할 수 있다.

$$T_{transfer} = \frac{L}{B} \times F_{CPU} = \frac{L \times F_{CPU}}{W \times F_{BUS}} \quad (2)$$

F_{BUS}는 DDR (Double Data Rate) SDRAM 기술을 사용한다면 200MHz이상을 지원할 수 있고, W는 온칩 DRAM에서는 핀 제약이 사라지므로 캐쉬 라인 크기와 같은 크기인 $W=L$로 설계한다. 이와 같이 설계하면 기존의 오프칩 DRAM 구조를 가지는 컴퓨터 시스템에서와 달리 $T_{access_latency}$에서 $T_{transfer}$가 차지하는 비중은 매우 작아지고 T_{DRAM_access}의 비중은 매우 커진다[13].

$T_{overhead}$[cpu cycles]는 프로세서가 메모리 버스를 중재하는 시간, t_{arb}, 하고 프로세서에서 DRAM으로가는 요구와 DRAM에서 L1 캐쉬로 가는 데이타를 버퍼링하는 시간, t_{buf}, 으로 구성되어 다음과 같이 나타낸다.

$$T_{overhead} = t_{arb} + t_{buf} \quad (3)$$

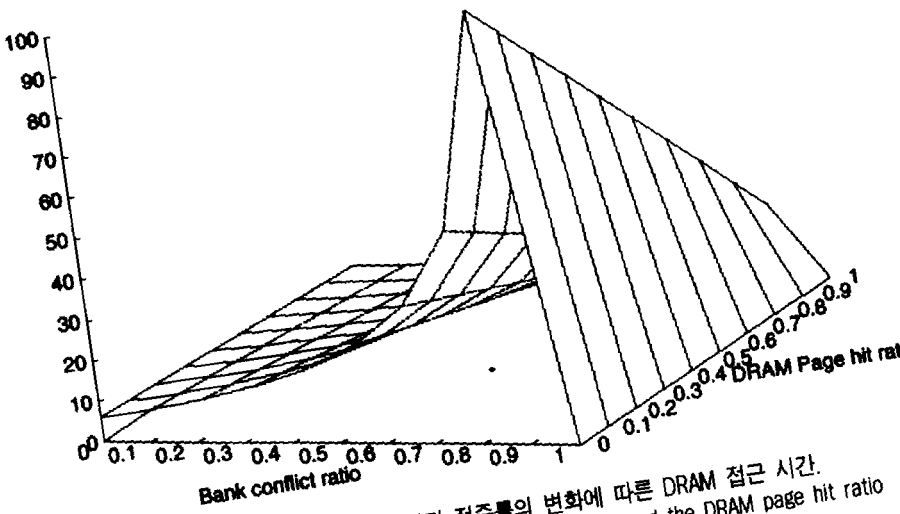
그런데 DRAM 뱅크의 감지 증폭기와 캐쉬 라인을 직접 연결하는 EDRAM 16]이나 컬럼 버퍼 캐쉬 [12]와 같은 경우에는, 캐쉬와 DRAM 사이의 메모리 버스를 각 DRAM 뱅크가 공유하는 것이 아니다. 따라서, 각 DRAM 뱅크와 연결된 버퍼 캐쉬들은 그 DRAM 뱅크만을 캐싱하므로 t_{arb}은 필요없게되어서 $T_{overhead}$는 메모리 버스를 공유하는 구조보다는 작게 된다. 그러나 이와 같은 캐쉬 구조는 뱅크하나에 집중적인 접근이 발생하면 캐쉬 접근 성공 비율이 떨어지게 된다.

T_{DRAM_access} [cpu cycles]는 접근 요구가 DRAM 뱅크에 보내 진 후부터 데이타가 DRAM 뱅크에서 출력되기 시작하는 시간을 의미하며 다음과 같이 구성할 수 있다.

$$T_{DRAM\_access} = \sum_{i=0}^{L} \alpha_i [(1-\beta_i) t_{RAC} + (\beta_i + \varepsilon_i)(t_{RP} + t_{RAC})] = \sum_{i=0}^{L} \alpha_i [(1-\beta_i) t_{RAC} + (\beta_i + \frac{\beta_i}{1-\beta_i})(t_{RP} + t_{RAC})] \quad (4)$$

오프칩 DRAM을 가지는 일반적인 컴퓨터 시스템에서는 메모리 버스 넓이와 지연 제약에 의하여 메모리 접근 요구가 많이 발생하지 않아서 뱅크 충돌률 (β_i) 은 크지 않으며 따라서 성능 제약 요소가 아니다. 그러나, 온칩 DRAM을 가지는 컴퓨터 시스템에서는 늘어난 내부 메모리 대역폭을 활용하기 위해 메모리 접근 요구를 많이 발생할 수 있으며 그때는 β_i가 클 수 있으므로 성능 제약 요소가 된다. 또한 넌 블럭킹(non blocking) 메모리 시스템, 다중 이슈 슈퍼스칼라 프로세서, 다중프로세서 등과 온칩 DRAM으로 구성된 고성능 컴퓨터 시스템의 경우에는 동시 다발의 메모리 요구를 발생하게 되고 따라서 뱅크 개수가 작은 DRAM 구조를 가지면 β_i는 커질 수밖에 없다 [11]. 따라서 온칩 DRAM 구조를 가지는 컴퓨터 시스템에서는 β_i를 감소시키는 새로운 DRAM 구조를 채택할 수 있다[10,11].

t_{RAC}은 DRAM의 물리적 특성에 의해서 결정되는 요소로써 RAS 신호가 액티브한 후 데이터가 출력되기까지의 시간으로 정의되며 본 논문에서는 30ns로 가정한다. t_{RAC}은 RAS 접근 시간으로 부른다. t_{RP}는 DRAM의 프리차지(precharge) 시간으로써 DRAM 뱅크는 프리차지 되어 있어야지 DRAM 셀의 데이타를 감지할 수 있다. 따라서 데이타를 접근한 후에는 DRAM 뱅크를 프리차지 하는 t_{RP} 시간이 필요하다. t_{RP}+t_{RAC}은 RAS 싸이클 시간으로 부르며 본 논문에서는 60ns로 가정한다. 만약 DRAM 뱅크가 서비스를 하지 않고 있었다면 DRAM 뱅크는 프리차지 되어 있으므로 t_{RP}는 숨길 수 있다. DRAM 페이지 적중률(γ_i) 은 일반 오프칩 DRAM의 버스트 모드 접근의 경우에 감지 증폭기를 프리차지하지 않아서 닫지 않고 열어놓아서 연속적인 컬럼 주소에 의한 데이타 선택을 수행할 수 있는 확률을 의미한다. 그러나 온칩 DRAM 구조에서는 내부 메모리 버스 넓이가 캐쉬 라인 크기까지 커질 수 있으므로 오프칩 DRAM과 달리 버스트 모드 접근은 필요하지 않는다. 즉 감지 증폭기의 내용 중에서 캐쉬 라인에 해당하는 데이타를 한번에 전송할



(그림 3) 뱅크 충돌률과 DRAM 페이지 적중률의 변화에 따른 DRAM 접근 시간.
(Fig. 3) DRAM access time for changes of the bank conflict ratio and the DRAM page hit ratio

수 있으므로, 감지 증폭기의 데이타를 한번 접근한 후 곧바로 프리차지 한다면 γ_i는 0이 된다. 그런데 온칩 DRAM의 감지 증폭기가 프리차지 되기 전에 이어지는 요구가 원하는 데이타가 감지 증폭기에 포함되어 있다면 DRAM 접근은 CAS 지연 (t_{CAC})에 해결될 수 있다. t_{CAC}은 데이타가 이미 감지된 후에 CAS 신호의 로(low) 액티브부터 데이타가 출력되기까지의 시간으로써 본 논문에서는 10ns로 가정한다. 만약 온칩 DRAM 뱅크 구조와 메카니즘을 바꾸어서 γ_i가 양의 값을 가지면 T_{DRAM_access}는 다음 식과 같이 감소할 수 있다.

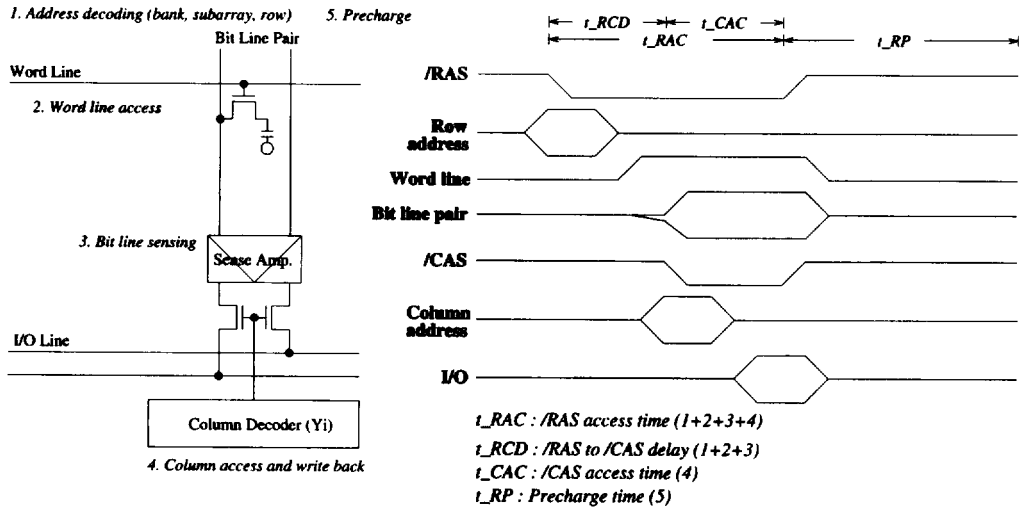
$$T_{DRAM\_access} = \sum_{i=0}^{n-1} \alpha_i [(1-\beta_i)\gamma_i t_{CAC} + (1-\gamma_i) t_{RAC} + (\beta_{i+} \frac{\beta_i}{1-\beta_i})\gamma_i t_{CAC} + (1-\gamma_i)(t_{RP} + t_{RAC})] \quad (5)$$

그림 3은 뱅크 충돌률과 DRAM 페이지 적중률의 변화에 따른 DRAM 접근 시간의 변화를 나타낸다. 이때 α_i는 모든 뱅크 i에 대해서 동일한 값을 갖는다고 가정한다. 그림 3에서 보듯이 뱅크 충돌률이 증가할 수록 DRAM 접근 시간은 기하급수적으로 증가한다. DRAM 뱅크의 개수를 늘이면 뱅크 충돌률은 감소할 수 있지만 대신 전력과 면적의 오버헤드를 가지므로 트레이드오프가 필요하다. DRAM 페이지 적중률은 온칩 DRAM이 일반 DRAM의 구조와 프로토콜(protocol)을 가지면 0이지만 구조를 바꾸어 DRAM 페이지 적중률이 증가하면 DRAM 접근 시간은 상당히 많이 감소한다. 또한 DRAM 페이지 적중률이 증가되면 상대적으로 뱅크 충돌 비율도 감소하기 때문에 전체적인 성능 향상을 이룰 수 있다. 본 논문에서는 이와 같이 구성한 분석적 모델에서 보듯이 온칩 DRAM의 성능을 향상시키기 위해서 뱅크 충돌률을 감소시키고 DRAM 페이지 적중률을 증가시킬 수 있는 DRAM 구조를 제안하고 검증한다.

3. 설계 대안

3.1 DRAM 접근 순서

T_{DRAM_access}의 근본적인 시간 요소는 그림 4에서와 같이 t_{RAC} (=t_{RCD}+t_{CAC})와 t_{RP} 시간으로 나뉜다. t_{RAC}과 t_{RP}는 DRAM의 물리적 특성에 의하여 많이 줄일 수 없는 요소이다. 최근에 높은 대역폭을 제공하는 새로운 DRAM들은 물리적 제약을 받는 RAS 지연을 줄이는 것이 아니라 접근 비율을 늘이는 것이다. 예를 들어 FPM DRAM, EDO DRAM, SDRAM 등과 같이 감지 증폭기의 데이타를 활용하는 페이지 모드나 버스트 모드를 사용하여 연속적인 데이타 접근 비율을 늘이는 것이다. 더 나아가 컬럼 주소 파이프라인(pipeline), 선인출(prefetch), 다중-뱅크 아키텍처를 사용하여 전체 $T_{access_latency}$를 줄이는 것이다. 그러나 온칩 DRAM을



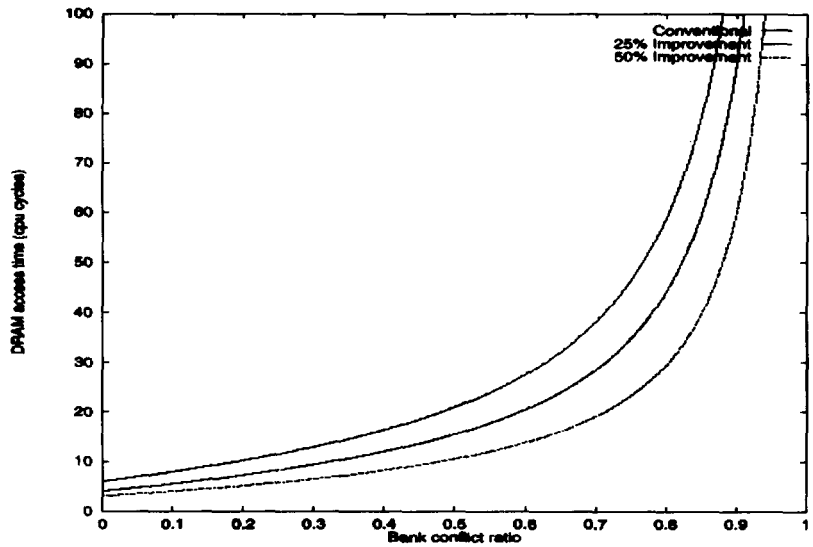
(그림 4) DRAM 접근 순서
(Fig. 4) DRAM access seguence

가지는 컴퓨터 시스템에서는 1 캐쉬 라인을 접근하는데 페이지 모드나 버스트 모드는 필요 없어진다. 왜냐하면 메모리 버스 넓이가 캐쉬 라인 크기까지 증가할 수 있기 때문이다. 따라서 새로운 온칩 DRAM 구조와 메카니즘이 필요하다.

DRAM에서 데이타를 읽는 과정은 그림 4에서 보는 것과 같이 크게 5가지로 나뉜다. (1) 주소 해독(decode), (2) 워드(word) 라인 접근, (3) 비트(bit) 라인 감지, (4) 컬럼 접근과 재쓰기(write back) (5) 프리차지. DRAM 쓰기 동작은 읽기 동작과 유사하면서 보다 간단하므로 본 논문에서는 DRAM 읽기 동작만 다룬다. 그림 4는 일반 DRAM의 접근 시간 순서도를 나타낸다. 로(row) 접근 시간(t_{RCD})은 위의 접근 스텝 중에서 (1), (2), (3) 으로 이루어 지고, 컬럼 접근 시간(t_{CAC})은 (4) 로 이루어진다. 마지막으로 t_{RP}은 (5) 프리차지 시간을 나타낸다. RAS 접근 시간 (t_{RAC})은 로 접근 시간과 컬럼 접근 시간으로 이루어 지고, RAS 싸이클 시간은 RAS 접근 시간과 프리차지로 이루어진다.

DRAM 접근 순서의 각각은 DRAM 특성에 해당하고 주로 VLSI의 발전과 함께 발전하였다. 본 논문은 상위 단계의 마이크로 아키텍쳐에 집중하므로, DRAM 접근 스텝 각각에 대해서는 개략적인 조사를 수행한다.

(1) 주소 해독은 주소를 래치(latch)한 후 워드 라인을 결정하는 시간으로써 핀 제약에 의하여 일반 DRAM에서는 보통 로와 컬럼 주소를 멀티플렉싱하는 것이 보통이다. 그런데 주소 멀티플렉싱을 안하면 상당한 시간을 줄일 수 있으나 [17,21], 이것은 핀 수의 증가에 따른 추가적인 패키지 면적이 필요하다. 이러한 주소 멀티플렉싱 문제는 온칩 DRAM과 같이 핀 제약이 없는 구조에서는 자연스럽게 해결된다. (2) 워드 라인 접근은 워드 라인을 액티브 전파(propagation)하는 시간이다. 즉 워드 라인에 연결된 게이트 개수에 따라 달라지는 RC 지연이므로 메모리 어레이를 서브 블럭으로 나누어 워드 라인 지연을 줄이는 방법이 많이 사용된다. 그러나 이것은 워드 라인을 작은 부분으로 나누기 위해 중간에 버퍼를 두어야하고 제어가 복잡해지므로 전력과 면적 면에서 오버헤드가 발생한다. 이러한 점은 메모리 집적도와 트레이드오프(trade-off)를 해야한다. 워드 라인 지연을 줄이는 방법으로 주 워드 라인과 서브 워드 라인으로 나누고, 일반적인 폴리실리콘(polysilicon) 라인을 쓰지 않고 메탈 라인으로 사용하는 방법이 있다 [18]. 또한 BiCMOS 라인 드라이버를 사용하여 워드 라인 지연을 줄이는 방법도 있다 [19,20]. (3) 비트 라인 감지는 액티브된 워드 라인의 각 DRAM 셀의 축전기(capacitor)에서 전하를 읽어내는 감지 시간이다. 감지 증폭기를 읽기/쓰기 증폭기



(그림 5) t_{RCD}, t_{CAC}, t_{RP}의 증가에 따른 성능 개선 정도 (DRAM 페이지 적중률 γ_1=0)

(Fig. 5) Performance improvement for decreasing t_{RCD}, t_{CAC}, and t_{RP} (DRAM page hit ratio γ_1 =0)

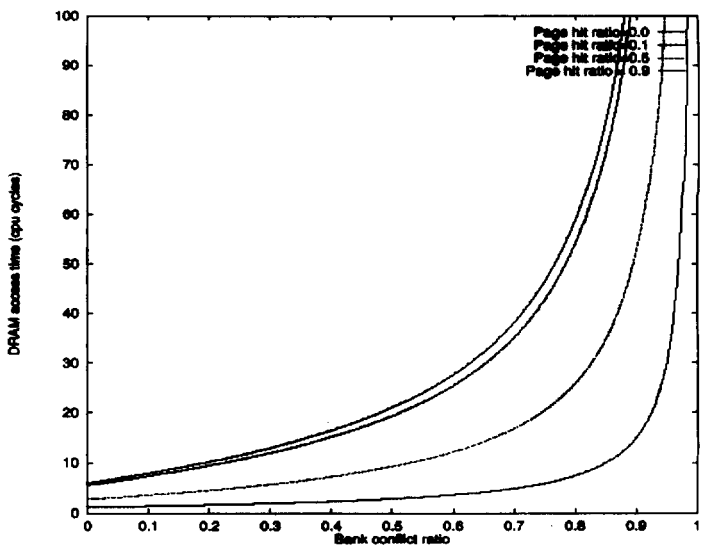
로 분리시키는 직접 감지 방법을 사용하여 성능을 높일 수 있다 [17]. 또한 감지 증폭기 드라이버를 높이는 오버드리븐(overdriven) 감지 증폭기 방법도 성능을 높일 수 있다 [18]. (4) 컬럼 접근 재쓰기는 감지 증폭기로 하나의 로의 데이타 중에서 원하는 컬럼의 데이터를 I/O 라인으로 읽어내고, 또한 DRAM 셀의 각 축전기에 본래의 전하를 재쓰기하는 과정이다. (5) 프리차지 시간은 DRAM의 각 라인들은 준비하는 과정으로, 각 라인들을 프리차지 시켜야만 DRAM이 제대로 동작한다. 이 시간은 오버헤드와 함께 전체 RAS 싸이클 시간의 절반정도를 차지할 정도로 비중이 크다. 다중-뱅크 DRAM 구조를 가지면 바로 t_{RP}시간을 숨길 수 있어서 전체 DRAM 성능을 높일 수 있다.

그림 5는 이전 절에서 구성한 분석적 모델을 통해서, VLSI의 발전에 따른 DRAM 접근 시간의 성능 개선(performance improvement) 정도를 예측해 본다. 즉 t_{RCD}, t_{CAC}, t_{RP}가 감소함에 따라 변하는 DRAM 접근 시간의 성능 증대 효과를 나타낸다. 논문 [11]에 따르면 온칩 다중프로세서에서 부동소수점(floating point) SPEC95 벤치마크의 경우에 DRAM 뱅크 충돌률은 50%를 상회하는 시뮬레이션 결과를 보여준다.

따라서 β_i=0.5, γ_i= 0.0이라고 가정하고, DRAM 접근 시간의 근본 요소들의 성능 개선 정도가 각각 25%와 50%라고 가정했을 때 DRAM 접근 시간은 26%와 50%의 성능 향상을 보인다. 즉 각 DRAM 접근 요소의 성능 향상은 DRAM 접근 시간의 성능 향상에 그대로 반영되리라고 예측할 수 있다. 그러나, 각 요소들의 성능 개선은 VLSI 특성에 따라 거의 포화되어 향후 큰 발전을 이루지 못할 것으로 예상되고, 또한 뱅크 충돌률이 0.8을 넘으면 DRAM 접근 시간은 급격히 증가하여 각 요소의 감소가 큰 성능 개선을 이루지 못함을 알 수 있다.

3.2 다중-뱅크 아키텍쳐

캐쉬 메모리의 성능을 좌우하는 근본 요소는 4가지로써 캐쉬 크기, 캐쉬 라인 크기, 대체 정책(replacement policy), 상관도(associativity)이다. 이에 비해 DRAM 주메모리의 성능을 좌우하는 요소는 DRAM 크기, 뱅크 개수, 뱅크 형태 (뱅크 넓이 = 감지 증폭기의 넓이, 서브어레이(subarray)의 개수), 뱅크 인터리빙 메카니즘으로 볼 수 있다. 이 절에서는 DRAM 성능 요소로써 뱅크 아키텍쳐와 뱅크 인터리빙 메카니즘을 고찰하고 새로운 구조를 제안한다. DRAM 뱅크는 여러 개의 서브어레이로 구성되고 각 서브어레이들은 공유 감지 증폭기를 사용한다고 가정한다.

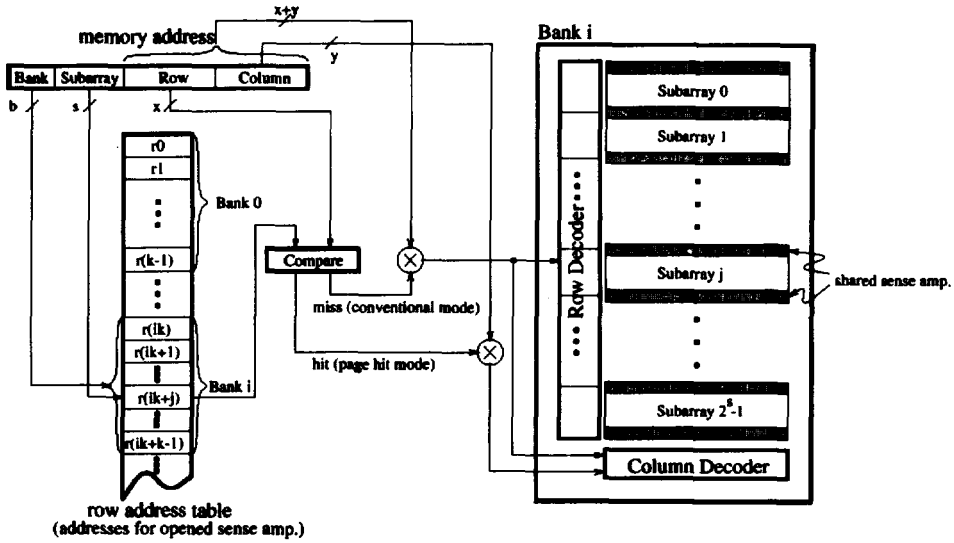


(그림 6) DRAM 페이지 적중률의 증가에 따른 성능 개선 정도.
(Fig. 6) Performance improvement for the increasing DRAM page hit ratio

일반적인 컴퓨터 시스템에서 $T_{transfer}$ 비중을 줄이기 위해서 제시되는 고전적인 방법은 메모리 버스 넓이를 늘리거나, 메모리 인터리빙을 사용하는 것이었다. 메모리 버스 넓이는 오프칩 DRAM의 경우에는 핀 제약에 의하여 제약을 받았으나, 온칩 DRAM의 경우에는 이러한 제약이 없어지므로 자연스럽게 캐쉬 라인 크기의 큰 메모리 버스 넓이를 가질 수 있다. 고전적인 메모리 인터리빙 방법은 벡터 슈퍼컴퓨터에서 사용하던 방법이다. 즉 메모리 블럭 접근에 대해서 인접한 메모리 워드를 서로 다른 메모리 뱅크에 저장한 후 파이프라인 접근을 통해서 전송 시간을 감소하는 것이다. 그러나, 최근의 DRAM 뱅크 구조는 캐쉬 라인에 해당하는 메모리 블럭을 여러 뱅크에 분산 저장하는 고전적인 인터리빙을 사용하지 않고 1 캐쉬 라인은 한 뱅크에 저장한다. 그리고 캐쉬 라인을 접근할 때는 단지 감지 증폭기를 캐쉬처럼 사용하는 페이지 모드와 버스트 모드를 통해서 DRAM 접근 비율을 증가시킨다. 이전 절에서 설명했듯이 온칩 DRAM 에서 메모리 버스 넓이가 캐쉬 라인 크기로 늘어나면 캐쉬 라인 접근하는 데 페이지 모드나 버스트 모드는 불필요해 진다. DRAM의 발전은 FPM DRAM, EDO DRAM, SDRAM, 컬럼 주소 파이프라인, 선인출 방법 등과 같은 점진적 설계 방식들이 있고, 최근에 고속, 협역(narrow width) 버스를 사용하는 프로토콜 기반 DRAM으로 Rambus DRAM, SyncLink DRAM 등과 같은 혁신적 설계 방식들이 있다.

다중-뱅크 DRAM 구조는 뱅크 충돌률 (β_i)을 감소하고 t_{RP} 시간을 숨겨서 DRAM 접근 시간을 감소시키는 역할을 한다. 그러나 뱅크의 개수를 늘이면 추가적인 컬럼 디코더와 I/O 라인 등에 따른 전력과 면적의 오버헤드를 겪게된다. 이전 절의 그림 3에서 보듯이 뱅크 충돌률을 감소시키면 DRAM 접근 시간의 성능은 상당히 개선된다. 이와 같은 이유에 의해서 뱅크 개수를 늘리지 않고 뱅크 충돌률을 감소시키는 구조가 제안되었다. 이 구조는 뱅크 내부의 서브어레이를 마치 반독립적 뱅크처럼 사용하는 계층적 다중-뱅크 DRAM [10,11] 이다. 이 구조는 다중프로세서 환경의 부동소수점 SPEC95 벤치마크에 대해서 4개의 주 뱅크로 구성된 256Mb 계층적 다중-뱅크 DRAM이 면적 면에서는 일반적인 4 뱅크 DRAM과 비슷하면서 DRAM 접근 시간은 일반적인 16 또는 32 뱅크 DRAM과 비슷한 정도를 보였다.

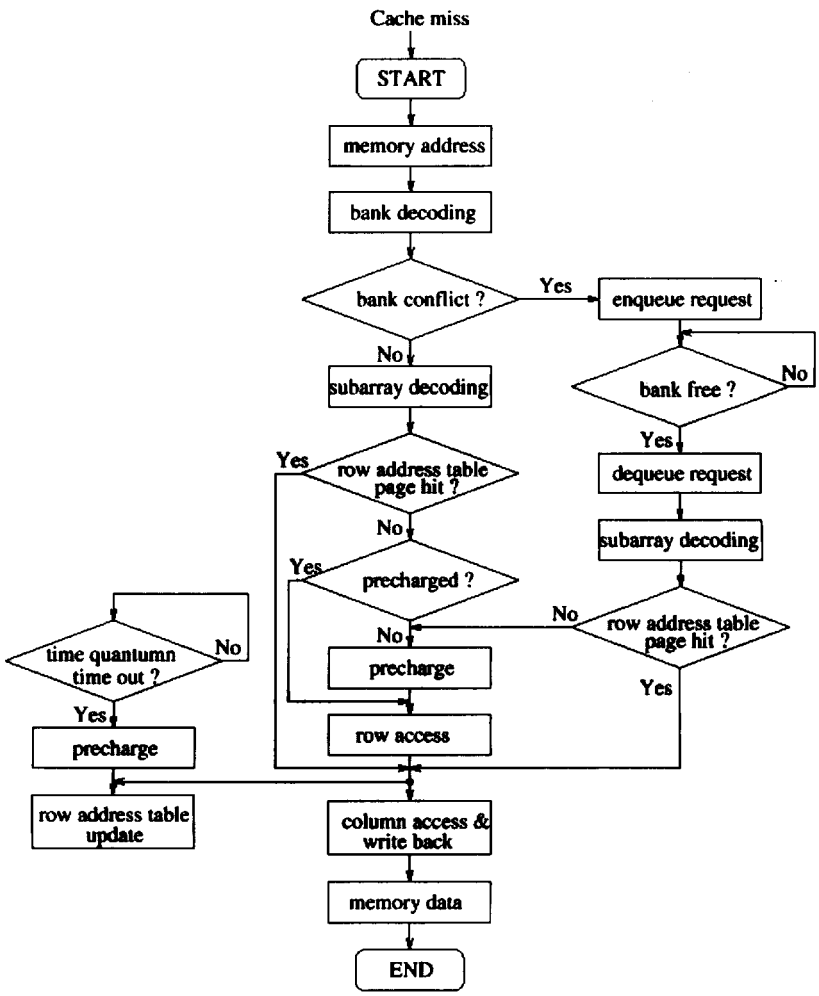
본 논문에서는 DRAM 접근 시간을 감소시키는 다른 설계 대안으로써 DRAM 페이지 적중률(γ_i)을 증가시키는 프리차지 연기(delayed precharge) 아키텍쳐를 제안한다. 그림 6은 이전 절에서 구성한 분석적 모



(그림 7) 제안하는 뱅크 아키텍쳐: 프리차지 연기 아키텍쳐.
(Fig. 7) Proposed bank architecture : Delayed precharge architecture

델을 통해서, DRAM 페이지 적중률의 증가에 따른 DRAM 접근 시간의 성능 중대 효과를 예측한다. 뱅크 충돌률이 0.5일 경우에는 DRAM 페이지 적중률의 증가에 따른 성능 향상 정도는 t_{RAC}의 감소에 따른 성능 개선 정도와 비슷하다. 그러나 뱅크 충돌률이 그 이상일 경우에는 DRAM 페이지 적중률의 증가에 따른 성능 향상은 t_{RAC}의 그것 보다 훨씬 우수함을 알 수 있다.

그림 7은 제안하는 프리차지 연기 뱅크 아키텍처를 보여준다. 프리차지 연기 구조는 온칩 DRAM 구조에서 캐쉬 라인 접근을 처리한 후 기존의 일반 DRAM처럼 곧바로 프리차지 하여 감지 증폭기를 닫는 것이 아니다. 미래의 캐쉬 접근 실패에 따른 DRAM 요구가 동일한 뱅크 로 접근 요구일 것에 대비해서 프리차지를 일정한 시간 정량(quantum) 동안 연기하여 이미 열려있는 감지 증폭기를 캐쉬처럼 활용하는 것이다. 이와 같이 하면 일정한 시간 정량 안에 도착한 동일한 뱅크 로 접근 요구는 DRAM 접근 시간을 t_{CAC}으로 처리할 수 있으므로 접근 시간이 상당히 감소한다. 그러나 시간 정량 안에 도착한 접근 요구가 동일한 로 주소를 가지지 않으면 아직 DRAM은 준비가 안된 상태 즉, 프리차지가 안된 상태이므로 t_{RAS}이라는 더 큰 지



(그림 8) 프리차지 연기 아키텍처에 대한 동작 모델.
(Fig. 8) Operational model for the delayed precharge architecture

연을 가지게 된다. 따라서 연기를 주는 일정한 시간 정량은 DRAM 페이지 적중률과 트레이드오프가 필요하다.

제안하는 구조는 기존의 일반 DRAM 구조에 프리차지를 일정시간 연기하는 시계 로직과 도착한 메모리 요구가 동일한 로 주소를 가져서 페이지 적중인지를 판별하는 비교 로직 정도의 작은 변경이 필요하다. 그리고 이전 메모리 요구들 중에서 감지 증폭기에 데이타가 아직 남아있는 즉 열려있는 감지 증폭기를 가지는 메모리 요구들의 로 주소들을 저장하기 위해서 로 주소 테이블(row address table)이 필요하다. 로 주소 테이블의 항목 개수는 뱅크 개수와 서브어레이 개수의 곱이다. 즉 메모리 주소 필드 중에서 뱅크 필드 넓이가 b이고 서브어레이 필드 넓이가 s라면 로 주소 테이블의 항목 개수는 2^{b+s}이다. 그리고, 로 주소 테이블에 저장된 로 주소의 총 크기는 $2^{b+s}x$비트이다. 예를 들어서 256Mbit DRAM일 경우에 뱅크가 4개이고 서브어레이가 32개이고, 각 서브어레이 당 256개의 로를 가진다면 로 주소 테이블의 총 크기는 $2^{2+5}\times 8$ = 1024 비트 = 128 바이트이다. 따라서 로 주소 테이블의 비용은 미미하다고 할 수 있다.

만약 현재의 메모리 요구가 로 주소 테이블의 값과 같다면 성공으로써 페이지 적중 모드로 동작하고 아니라면 일반적인 모드로 동작한다. 페이지 적중 모드에서는 이미 감지 증폭기까지 읽혀진 데이타를 컬럼 접근만 하면 된다. 그림 7의 메모리 요구는 뱅크 $i(0\leq i\leq 2^{b-1})$와 서브어레이 $j(0\leq j\leq k-1=2^{s-1})$의 접근 예를 보여준다. 그림 8은 프리차지 연기 아키텍처에서 DRAM 접근 요구가 처리되는 과정을 보여주는 동작 모델이다. 제안하는 프리차지 연기 아키텍처는 계층적 다중-뱅크 DRAM [10,11]에 첨가되어 동작할 수도 있다. 즉 각각의 서브어레이들은 반독립적 뱅크로써 동작하면서, 프리차지 연기를 수행하여 뱅크 충돌률은 낮추고, DRAM 페이지 적중률은 증가시킬 수 있다. 그림 8은 일반적인 뱅크 아키텍처를 기반으로 하는 프리차지 연기 아키텍처의 동작 모델만을 나타낸다.

또한 캐쉬 실패 비율이 높은 응용을 위해서는 요구 큐에 저장된 대기 요구들의 주소를 살펴보고 동일 로를 가지는 요구들끼리 클러스터링(clustering)하는 재순서(reordering)를 수행하여 DRAM 페이지 적중률을 더욱 더 증대시킬 수 있다. 그러나 캐쉬 성공 비율이 높은 CPU 중심의 응용일 경우에는 DRAM 접근 비율이 높지 않고, 따라서 요구 큐에 대기(pending)하는 요구 수가 적으므로 요구 재순서는 큰 의미를 가지지 않게 된다. 4절에서는 자세한 시뮬레이션을 통해서 제안한 프리차지 연기 구조가 계층적 다중-뱅크구조보다 우수함을 증명할 것이다.

4. 실험 연구

이 절에서는 3절에서 고찰한 여러 가지 설계 방식들에 대해서 시뮬레이션을 통하여 성능을 비교 검증한다.

4.1 시뮬레이션 환경

본 논문에서는 SimpleScalar 툴 [23]을 사용하여 여러 가지 설계 대안들을 시뮬레이션 하였다. SimpleScalar 툴은 실행 방식(execution driven), 비순서(out-of order), 투자방식(speculative) 프로세서 시뮬레이터이다. 그러나 SimpleScalar 툴은 DRAM 주메모리에 대한 자세한 기술을 하지 않았다. 다만 주메모리는 2GBytes의 가상 메모리를 가정하여 페이지 폴트는 발생하지 않는다. 본 논문에서는 SimpleScalar 툴에 DRAM 내부 구조를 기술하여 첨가하였다. 첨가한 코드는 일반적인 DRAM의 다중-뱅크구조와 페이지 폴트 처리 메카니즘을 구현하였다. 페이지 폴트 처리 메카니즘은 LRU 대체를 사용하였다. 또한 계층적 다중-뱅크와 제안하는 프리차지 연기 구조도 성능 비교를 위해서 코딩하였다. 다음의 항목들은 시뮬레이터에 첨가한 여러 가지 뱅크 아키텍처들의 모델들을 나열한다.

DRAM 뱅크 아키텍처

- B_1 : 일반적인 DRAM 뱅크 아키텍처.
- B_2 : 계층적 다중-뱅크 아키텍처.
- B_3 : 계층적 다중-뱅크 아키텍처에 요구 재순서 추가.
- B_4 : 일반적 뱅크 기반의 프리차지 연기 아키텍처.
- B_5 : 일반적 뱅크 기반의 프리차지 연기 아키텍처에 요구 재순서 추가.
- B_6 : 계층적 다중-뱅크 아키텍처 기반의 프리차지 연기 아키텍처

- B_7 : 계층적 다중-뱅크 아키텍쳐 기반의 프리차지 연기 아키텍쳐에 요구 재순서 추가.

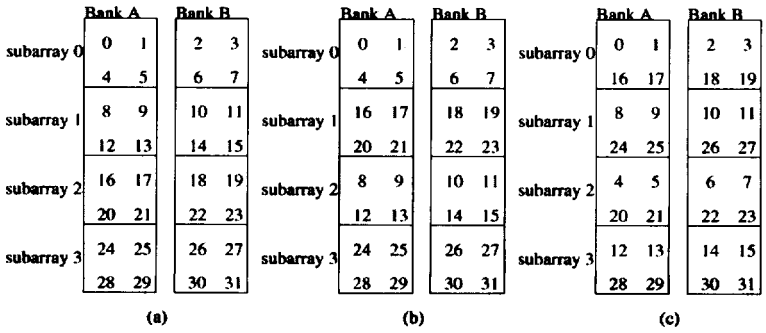
인터리빙 메카니즘은 일반적인 뱅크 구조에서는 크게 문제가 되지 않는다. 그러나, 계층적 다중-뱅크 DRAM이나 제안하는 프리차지 연기 뱅크 DRAM에서는 메카니즘에 따라서 많은 성능 차이를 보인다. 따라서 본 논문의 시뮬레이션에서는 일반적인 인터리빙 메카니즘과 계층적 다중-뱅크 DRAM에서 사용한 인터리빙 메카니즘과 더불어서 성능을 더욱 더 높일 수 있는 인터리빙 메카니즘을 시뮬레이션의 변수로 사용하였다. 다음은 이와 같이 설명한 인터리빙 메카니즘의 변수들이다.

인터리빙 메카니즘

- I_1 : 뱅크 로 인터리빙 (그림 9 (a)).
- I_2 : 계층적 다중-뱅크 아키텍쳐에서 사용된 뱅크 로 인터리빙에 셔플된 서브어레이 추가 (그림 9 (b)).
- I_3 : 뱅크 로와 서브어레이의 인터리빙에 셔플된 서브어레이 추가 (그림 9 (c)).

대상 벤치마크로는 SPEC95 벤치마크를 사용하였다. 정수 벤치마크로써 099.go, 126.gcc, 130.li를 사용하였고, 부동소수점 벤치마크로써 101.tomcatv, 102.swim, 103.su2cor, 107.mgrid, 110.applu를 대상으로 하였다.

표 1은 시뮬레이션에 사용된 아키텍쳐 모델을 나타낸다. 기본 아키텍쳐로써 CPU는 200MHz 4 이슈 비순서 SimpleScalar 프로세서를 가정하고 SRAM 캐쉬 와 DRAM 주메모리가 온칩화 되었다. 캐쉬 변수는 DRAM 접근 비율의 가장 큰 영향을 미치는 요소로써 캐쉬 크



(그림 9) 뱅크 인터리빙의 주소 매핑 방식. 가정: (1) 4개의 서브어레이를 가지는 2 뱅크 DRAM. (2) 각 서브어레이의 숫자는 선형 주소 공간 표현.
(Fig. 9) Address mapping methods in bank interleaving. Assume: First, two bank DRAM with each of four Sub-arrays. Second, the numbers in each Sub-array presents the linear address space

〈표 1〉 시뮬레이션 아키텍쳐 모델
〈Table 1〉 Architectural model of simulation

구 성	A	B	C	D	E	F
뱅크 아키텍쳐	B_1	B_2		B_4		B_6
인터리빙 아키텍쳐	I_1	I_2	I_3	I_2	I_3	I_3
프로세서	200MHz, 4-이슈 비순서 SimpleScalar					
SRAM 캐쉬	온칩 16KB 또는 32KB 2-웨이 I/D 캐쉬					
캐쉬 라인	32 바이트					
캐쉬 접근 시간	1 CPU 싸이클					
DRAM 주메모리	온칩 256Mbits 4 또는 8 뱅크					
메모리 버스 넓이	캐쉬 라인 크기					
DRAM 접근 시간	t_{RAC} = 6, t_{RP} = 6, t_{RAS} = 12, t_{CAC} = 2 CPU 싸이클					
프리차지 연기 시간	50 CPU 싸이클 또는 무한대					

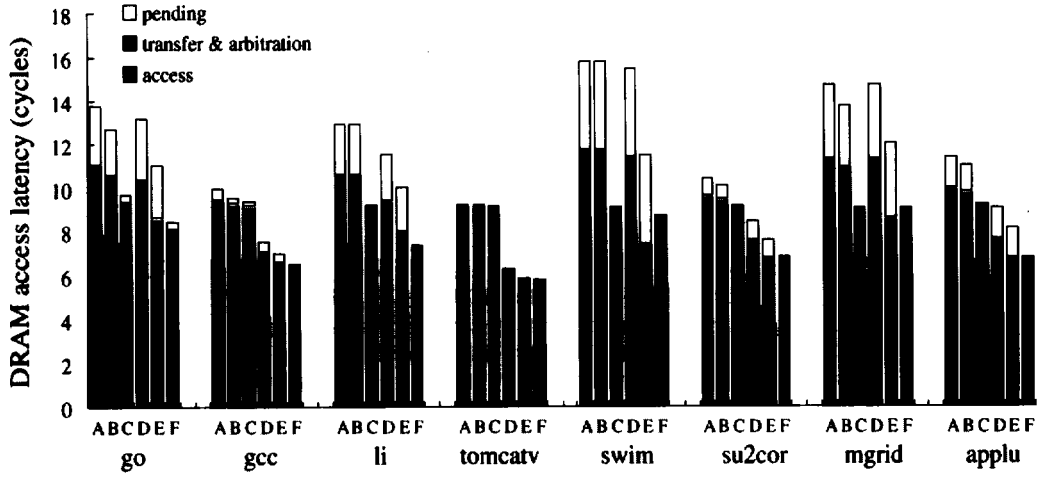
기를 16KB I/D 캐쉬에서 32KB I/D 캐쉬로 변화를 주었다. DRAM의 용량은 차세대 256Mbit 기술을 가정하고 DRAM의 아키텍쳐 변수로는 뱅크 개수의 변화와 프리차지 연기의 시간 정량 변화를 주었다.

4.2 기본 성능

· 그림 10은 표 1에서 정의한 여러 가지 모델들이 기본이 되는 아키텍쳐 변수에 대한 성능으로써 DRAM 접근 지연 시간을 나타낸다. 모두 6가지 DRAM 아키텍쳐 모델에 대하여 DRAM 접근 지연 시간을 요구 큐 대기 시간(pending), 전송과 중재 시간(transfer & arbitration), DRAM 접근 시간(access)으로 분류하여 나타내었다. 다만 모든 모델에 대하여 전송과 중재 시간은 3 CPU 싸이클로 고정하고 다른 시간 변수 환경에 따라 변화한다. 기본이 되는 변수는 16KB I/D 캐쉬, 4 뱅크 DRAM, 그리고 연기 시간량은 50 CPU 싸이클이다.

이러한 구성에서 SPEC 95 벤치마크 8가지를 시뮬레이션하면 모든 경우에 F 설계가 가장 낮은 DRAM 접근 지연 시간을 나타낸다. 그리고 제안하는 프리차지 연기 아키텍쳐를 가지는 D E 설계 또한 일반적인 모델인 A보다 좋음을 알 수 있다. 그러나 계층적 다중-뱅크 모델인 B와 C 중에서 인터리빙 메카니즘 I_2를 사용한 B 모델은 E보다 성능이 떨어졌다. 다만 I_3 인터리빙 메카니즘을 가지는 C 모델은 8가지 벤치마크중에서 4가지가 D, E 모델보다 우수하였다. 따라서 인터리빙 메카니즘이 DRAM 성능에 큰 영향을 미친다는 것을 알 수 있다.

표 2는 매 CPU 싸이클마다 발생하는 DRAM 접근 요구 개수 즉 DRAM 접근 비율을 의미한다. 특히 그림 10의 환경과 그림 13의 환경에서 A 모델의 경우에 DRAM 접근 비율을 의미한다. 다른 모델들의 경우도 매우 비슷한 DRAM 접근 비율을 나타낸다. 이와 같이 SPEC95 벤치마크 응용의 경우에는 DRAM 접근 비율이 상당히 작음을 알 수 있다. 따라서 DRAM 접근 지연의 개선이 전체 컴퓨터 시스템 아키텍쳐의 성능에 미치는 비중은 매우 작다. 그렇지만 다음 부절에서 보



(그림 10) DRAM 지연 시간. (4 뱅크 256Mbits DRAM, 16KB 명령어/데이타 캐쉬, 50 CPU 싸이클의 프리차지 연기 시간 정량)
(Fig. 10) DRAM access latency.(the 256Mbits DRAM with four banks, the 16KB instruction cache and 16KB data cache, the delayed time quantum of 50 CPU cycles for precharge)

〈표 2〉 DRAM 접근 비율 (DRAM 접근 / CPU 싸이클).
〈Table 2〉 DRAM access rate (DRAM access per CPU cycle)

SPEC95 벤치마크	go	gcc	li	tomcatv	swim	su2cor	mgird	applu
16KB I/D 캐쉬	0.038	0.071	0.021	0.087	0.001	0.027	0.01	0.007
32KB I/D 캐쉬	0.035	0.046	0.009	0.041	0.001	0.027	0.01	0.003

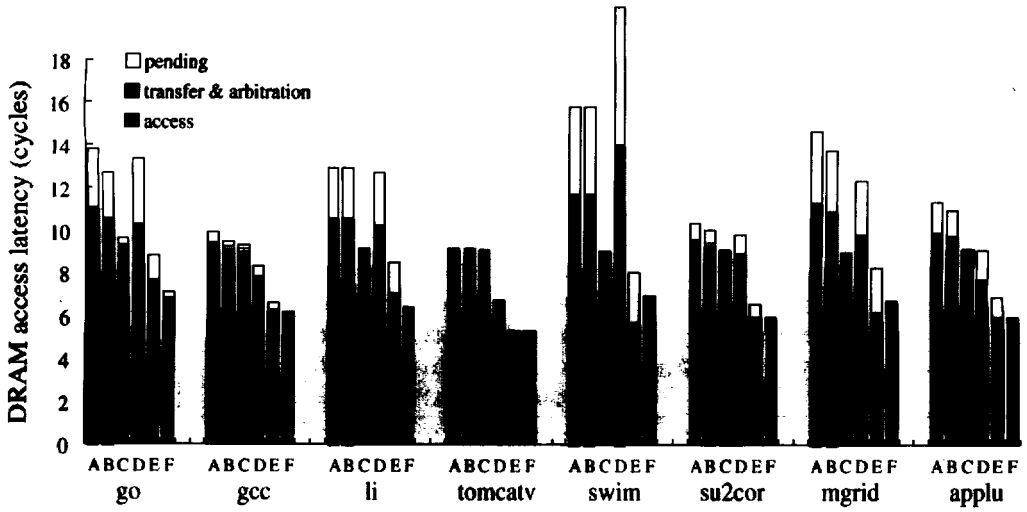
여주듯이 DRAM 접근 지연의 성능 개선이 상당하기 때문에 전체 시스템 성능이라할 수 있는 CPI의 성능 개선이 무시 못 할 수준임을 알 수 있다. 또한 SPEC95 벤치마크처럼 CPU 중심의 프로그램이 아닌 메모리 중심의 프로그램의 경우에는 DRAM 접근 비율이 훨씬 크므로 전체 컴퓨터 시스템 아키텍처의 성능 개선 비중이 커짐을 기대할 수 있다.

4.3 시스템 변수의 변화

그림 11은 그림 10과 달리 시간 정량을 무한대로 주고 시뮬레이션을 수행한 결과이다. D 모델의 경우에는 유한한 시간 정량을 주는 것보다 성능이 떨어졌지만 E와 F 모델은 성능이 월등히 나아졌다. 즉 모든 벤치마크에 대해서 F 모델의 성능이 제일 좋고 그 다음으로 E 모델의 성능이 좋았다.

이러한 결과는 시계 로직을 두고 일정 시간 동안 감지 증폭기를 재사용하고 일정시간이 지나면 프리차지하므로써 뱅크 서브어레이를 준비 상태로 만드는 것보다, 시계 로직 없이 무조건 감지 증폭기를 열어놓고 좋은 인터리빙 메카니즘에서는 높은 DRAM 페이지 적중률을 얻으므로 써 프리차지를 하지 않아서 발생하는 DRAM 페이지 실패 비용의 비율을 줄인다. 결과적으로 시계 로직을 없애므로 써 비용도 줄이고 성능도 높일 수 있다. 그리고 F 모델이 DRAM 접근 지연이 가장 작지만 계층적 다중-뱅크 아키텍처와 프리차지 연기 아키텍처를 모두 지원하므로 비용이 제일 많이 사용된다. 그러므로 비용을 고려하면 E 모델의 성능이 가장 우수함을 알 수 있다. 즉 E 모델은 A 모델에 비하여 DRAM 접근 지연이 32.61%~48.68%의 높은 성능 개선을 얻는다. 그런데 D, E 모델과 같이 계층적 다중-뱅크 아키텍처를 채택하지 않는 모델은 전체 DRAM 접근 지연의 감소 요인이 대기 시간의 감소가 아니라 DRAM 접근 시간의 감소임을 알 수 있다. 이것은 D, E, 모델이 DRAM 페이지 적중률을 증가시키고 그에 따라 DRAM 접근 시간을 감소시키는 구조이지, 뱅크 충돌률을 감소시키는 구조가 아니기 때문이다. 표 3에서 이와 같은 사실을 확인할 수 있다.

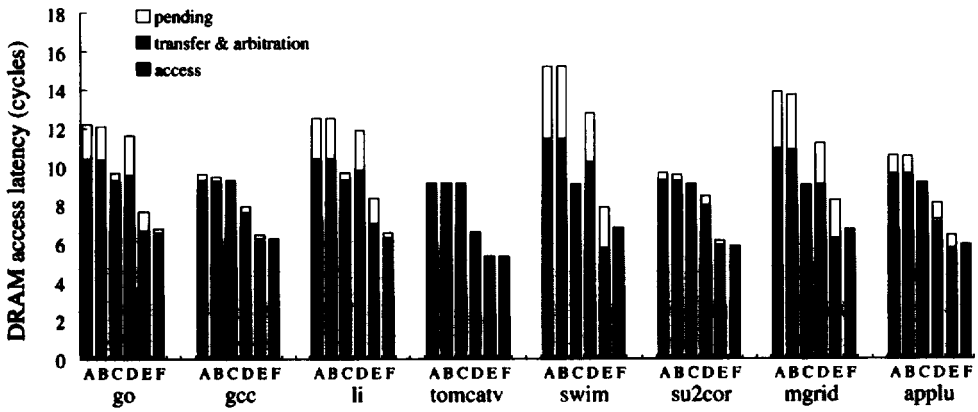
표 3은 그림 11의 환경에서 각 구성 모델들의 캐쉬 실패 비율, 뱅크 충돌률, DRAM 페이지 적중률, CPI를 나타낸다. 표 3에서 알 수 있듯이 제안하는 C, D, E, F 모델의 성능이 우수함을 알 수 있다. 캐쉬 실패 비율은 크게 달라지지 않지만 DRAM 접근 지연 개선에 영향을 받아서 CPI의 경우에는 E 모델이 A 모델보다 0.01%~25.16% 성능 개선을 보였다. 그러나 DRAM 접근 지연의 개선과 달리 D, E 모델은 뱅크



(그림 11) DRAM 지연 시간. (4 뱅크 256Mbits DRAM, 16KB 명령어/데이타 캐쉬, 프리차지의 무한대 연기 시간 정량)
(Fig. 11) DRAM access lotency (the 256Mbits DRAM with four banks, the 16KB instruction/data cache, the infinitive delayed time quantum for precharge)

〈표 3〉 그림 11에대한 성능 결과.
〈Table 3〉 Performance results for Figurce 11.

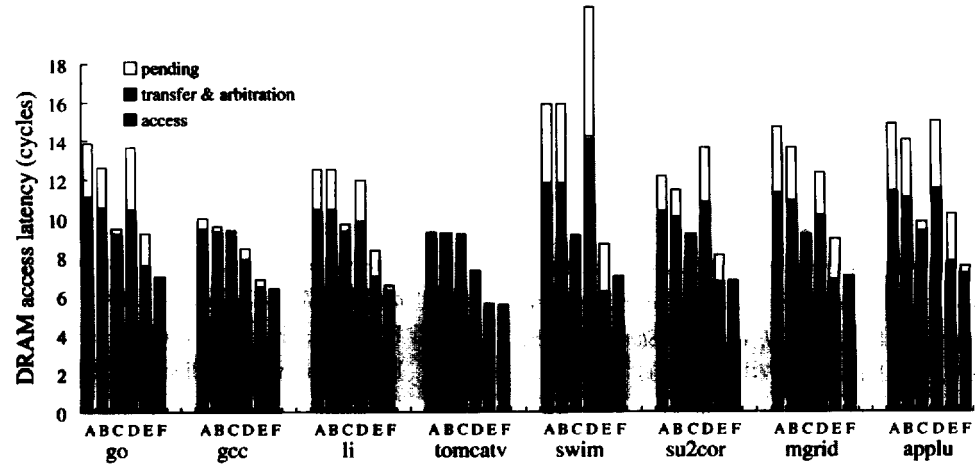
벤치마크	성능 척도	A	B	C	D	E	F
099.go	I 캐쉬 실패 비율	0.0036	0.0036	0.0036	0.0036	0.0036	0.0036
	D 캐쉬 실패 비율	0.0598	0.0599	0.0598	0.0599	0.0598	0.0598
	뱅크 충돌률	0.3466	0.2643	0.0515	0.3299	0.3169	0.0413
	DRAM 페이지 적중률	0.0000	0.0000	0.0000	0.4682	0.7246	0.5875
	CPI	0.6589	0.6544	0.6380	0.6391	0.6197	0.6079
126.gcc	I 캐쉬 실패 비율	0.0437	0.0439	0.0440	0.0452	0.0455	0.0448
	D 캐쉬 실패 비율	0.0253	0.0251	0.0250	0.0246	0.0242	0.0239
	뱅크 충돌률	0.0749	0.0466	0.0302	0.0594	0.0544	0.0130
	DRAM 페이지 적중률	0.0000	0.0000	0.0000	0.7083	0.8003	0.7939
	CPI	1.1397	1.1292	1.1234	1.0415	0.9600	0.9392
130.li	I 캐쉬 실패 비율	0.0052	0.0052	0.0052	0.0052	0.0052	0.0052
	D 캐쉬 실패 비율	0.0092	0.0092	0.0092	0.0092	0.0092	0.0092
	뱅크 충돌률	0.2634	0.2634	0.0113	0.2636	0.2591	0.0102
	DRAM 페이지 적중률	0.0000	0.0000	0.0000	0.4767	0.7336	0.6601
	CPI	0.7192	0.7192	0.6731	0.7136	0.6735	0.6479
101.tomcatv	I 캐쉬 실패 비율	0.0762	0.0762	0.0762	0.0761	0.0762	0.0762
	D 캐쉬 실패 비율	0.0007	0.0007	0.0007	0.0007	0.0007	0.0007
	뱅크 충돌률	0.0182	0.0168	0.0150	0.0049	0.0022	0.0001
	DRAM 페이지 적중률	0.0000	0.0000	0.0000	0.8275	0.9194	0.9184
	CPI	1.3365	1.3352	1.3365	1.1229	1.0002	0.9990
102.swim	I 캐쉬 실패 비율	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	D 캐쉬 실패 비율	0.0008	0.0008	0.0008	0.0008	0.0008	0.0008
	뱅크 충돌률	0.4552	0.4552	0.0074	0.4491	0.4484	0.0008
	DRAM 페이지 적중률	0.0000	0.0000	0.0000	0.0985	0.9026	0.5102
	CPI	0.7293	0.7293	0.7293	0.7292	0.7292	0.7292
103.su2cor	I 캐쉬 실패 비율	0.0136	0.0136	0.0135	0.0134	0.0130	0.0130
	D 캐쉬 실패 비율	0.0139	0.0140	0.0143	0.0146	0.0158	0.0159
	뱅크 충돌률	0.0998	0.0727	0.0092	0.1005	0.1032	0.0045
	DRAM 페이지 적중률	0.0000	0.0000	0.0000	0.6289	0.8387	0.7841
	CPI	0.7881	0.7843	0.7741	0.7609	0.7201	0.7139
107.mgrid	I 캐쉬 실패 비율	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
	D 캐쉬 실패 비율	0.0105	0.0104	0.0104	0.0105	0.0104	0.0104
	뱅크 충돌률	0.3809	0.3143	0.0010	0.3804	0.3797	0.0005
	DRAM 페이지 적중률	0.0000	0.0000	0.0000	0.5161	0.8557	0.5639
	CPI	0.6497	0.6443	0.6434	0.6479	0.6451	0.6411
110.applu	I 캐쉬 실패 비율	0.0027	0.0027	0.0027	0.0027	0.0027	0.0027
	D 캐쉬 실패 비율	0.0033	0.0033	0.0033	0.0033	0.0032	0.0033
	뱅크 충돌률	0.1567	0.1300	0.0099	0.1538	0.1533	0.0073
	DRAM 페이지 적중률	0.0000	0.0000	0.0000	0.7094	0.8531	0.7755
	CPI	0.6409	0.6398	0.6361	0.6317	0.6272	0.6248



(그림 12) DRAM 지연 시간. (8 뱅크 256Mbits DRAM, 16KB 명령어/데이타 캐쉬, 프리차지의 무한대 시간 정량)
(Fig. 12) DRAM access latency(the 256Mbits DRAM with eight banks, the 16KB instruction/data cache, the infinitive delayed time quantum for precharge)

충돌률은 크게 개선되지 않는 것을 알 수 있다. 다만 DRAM 페이지 적중률의 개선으로 DRAM 접근 지연이 크게 감소하고 그에 따라서 CPI도 개선된다. 그렇지만 연기 시간 정량이 무한대이기 때문에 DRAM 페이지 적중률이 상대적으로 E 모델보다 작은 D 모델은 프리차지 비용의 비중이 크므로 전체적인 성능 개선이 크지않는다. 그림 12는 DRAM 뱅크의 개수를 4개에서 8개로 늘렸을 때, 성능 변화를 살펴본다. 전체적으로 뱅크 개수를 늘이면 뱅크 충돌률이 감소하여 성능이 증대된다. 전체적인 성능 변화는 이전의 결과와 비슷하다.

그림 13은 캐쉬 크기를 16KB에서 32KB로 늘렸을 때의 DRAM 접근 지연 시간을 나타낸다. 캐쉬 크기가 증가하면 캐쉬 실패 비율이 감소하고 그에 따라서 표 2에서 보듯이 DRAM 접근 비율이 감소한다. 따라서 제안하는 DRAM 뱅크 아키텍처가 미치는 전체 성능 비중은 감소하게 된다. 그렇지만 DRAM 접근 지연의 성능 변화는 이전 결과와 비슷함을 알 수 있다.



(그림 13) DRAM 지연 시간. (4 뱅크 256Mbits DRAM, 32KB 명령어/데이타 캐쉬, 프리차지의 무한대 연기 정량).
(Fig. 13) DRAM access latency(the 256Mbits DRAM with four banks, 32KB instruction/data cache, the infinitive delayed time quantum for precharge)

5. 결론 및 향후 연구 방향

본 논문에서는 효율적인 p-m 구조를 제안하고 검증하였다. 우선 DRAM 구조를 시스템적으로 분석하고 분석적 모델을 구성하여 성능 향상점을 찾았다. 구성한 분석적 모델에 의하여 DRAM 페이지 적중을 활용하여 성능을 증대시키는 프리차지 연기 뱅크 아키텍쳐를 제안하였다. 제안한 구조의 검증은 SimpleScalar 툴을 개조하여 사용하였고 8가지 SPEC95 벤치마크 프로그램을 사용하였다. 제안한 구조는 일반적인 뱅크 구조뿐만 아니라 계층적 다중-뱅크 구조보다도 높은 성능을 제공한다. 시뮬레이션 결과 제안하는 구조는 일반적인 뱅크 구조보다 DRAM 접근 지연은 32.61%~48.68%의 높은 성능 향상을 얻었고, 그에 따라 CPI는 0.01%~5.16%의 성능 개선을 얻었다.

향후에는 SPEC 95 벤치마크 프로그램처럼 CPU 중심의 일이 아닌 대용량의 데이타를 사용하는 멀티미디어 프로그램처럼 메모리 중심의 프로그램의 성능 평가도 필요하다. 그리고 온칩 DRAM의 용량을 초과하는 데이타를 가지는 벤치마크 프로그램에 대해 페이지 폴트 환경을 고려하는 것이 필요하다. 또한 온칩 DRAM과 더불어서 다양한 캐쉬 구조의 고찰도 요구된다.

참 고 문 헌

[1] D. Patterson, et. al, "Intelligent RAM(IRAM): Chips that remember and compute," *Dig. Tech. Papers IEEE Int'l Solid-State Circuit Conf.*, pp. 224-225, 1997.

[2] K. Murakami, et. al, "Parallel processing RAM chip with 256Mb DRAM and quad processors," *Dig. Tech. Papers IEEE Int'l Solid-State Circuit Conf.*, pp.228-229, 1997.

[3] K. Inoue, H. Nakamura, and H. Kawai, "A 10Mb frame buffer memory with Z-compare and A-blend units," *IEEE J. of Solid-State Circuits*, Vol.30, No.12, pp.1563-1568, Dec. 1995.

[4] T. Shimizu, et. al, "A multimedia 32b RISC microprocessor with 16Mb DRAM," *ISSCC96*, pp. 216-217.

[5] D. Elliott, M. Snelgrove, and M. Stumm, "Computational RAM: a memory-SIMD hybrid and its application to DSP," *IEEE 1992 Custom Integrated Circuit Conference*, pp.30.6.1-30.6.4, 1992.

[6] M. Gokhale, B. Holmes, and K. Iobst, "Processing in memory: the terasys massively parallel PIM array," *IEEE Computer*, Vol.28, No.4, pp. 23-31, Apr. 1995.

[7] Y. Kim, T.D. Han, S.D. Kim, and S.B. Yang, "An effective memory-processor integrated architecture for computer vision," *26th Int'l Conf. Parallel Processing*, pp.266-269, Aug. 1997.

[8] J.M. Kim, Y. Kim, S.D. Kim, T.D. Han, and S.B. Yang, "An adaptive parallel computer vision system," Int'l J. Pattern Recognition and AI., Vol.12, No.3, pp.311-333, May 1998.

[9] Y. Kim, M.J. Noh, T.D. Han, S.D. Kim, and S.B. Yang, "Mapping of neural networks onto the memory-processor integrated architecture," to be accepted in *Neural Networks*.

[10] T. Yamauchi, L. Hammond, and K. Olukotun, "A single chip multiprocessor integrated with DRAM," *Workshop on Mixing Logic and DRAM with ISCA97*, 1997.

[11] T. Yamauchi, L. Hammond, and K. Olukotun, "The hierarchical multi-bank DRAM: A high-performance architecture for memory integrated with processors," *19th Conf. on Advanced Research in VLSI*, 1997.

[12] A. Saulsbury, F. Pong, and A. Nowatzyk, "Missing the memory wall: the case for processor-memory integration," *Int'l Symposium on Comp. Arch.*, pp.90-101, 1996.

[13] H. Miyajima, K. Inoue, K. Kai, and K. Murakami, "On-chip memorypath architecture for parallel processing," *Workshop on Mixing Logic and DRAM with ISCA97*, 1997.

[14] K. Inoue, K. Kai, and K. Murakami, "High bandwidth, variable line-size cache architecture for merged DRAM/logic LSIs," Technical Report, PPRAM-TR-29, Kyushu Univ., Feb. 1998.

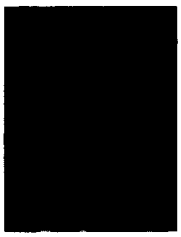
[15] Mitsubishi M5M4V4169 Cache DRAM Datasheet.

[16] Ramtron Specialty Memory Products Data Book, Oct. 1994.

[17] T. Nagai et al., "A 17-ns 4-Mb CMOS DRAM," IEEE Journal of Solid-State Circuits, Nov. 1991.
[18] A. Tanabe et al., "A 30-ns 64-Mb DRAM with Built-in Self-Test and Self-Repair Function," *IEEE Journal of Solid-State Circuits*, Nov., 1992.
[19] T. Kawahara et al., "A Circuit Technology for Sub-10-ns ECL 4-Mb BiCMOS DRAM's," *IEEE Journal of Solid-State Circuits*, Nov. 1991.
[20] G. Kitsukawa et al., "A 23-ns 1-Mb BiCMOS DRAM," *IEEE Journal of Solid-State Circuits*, Oct. 1990.
[21] T. Sugibayashi et al., "A 30-ns 256Mb DRAM with a Multidivided Array Structure," *IEEE Journal of Solid-State Circuits*, Nov. 1993.
[22] D.A. Patterson and J.L. Hennessy, *Computer Architecture: A Quantitive Approach*, Morgan Kaufmann Publishers, Inc, San Francisco, CA, 1996.
[23] D. Burger and T.M. Austin, "The SimpleScalar tool set, version 2.0," Technical Report 1342, Computer Science Department, University of Wisconsin, Madison, June 1997.

김 영 식

1993년 연세대학교 전산과학과 졸업(학사)
1995년 연세대학교 대학원 전산과학과 졸업(이학석사)
1995년~현재 연세대학교 대학원 컴퓨터과학과 박사과정
관심분야 : 병렬처리, 컴퓨터구조, 프로세서-메모리 집적구조



김 신 덕

1982년 연세대학교 공과대학 전자공학과(학사)
1987년 University of Oklahoma 전기공학(석사)
1991년 Purdue University 전기공학(박사)
1993년 2월~1995년 2월 광운대학교 컴퓨터공학과 조교수
1995년 3월~현재 연세대학교 공과대학 컴퓨터과학과 부교수
관심분야 : 병렬처리 시스템, 컴퓨터 구조, Heterogeneous Computing

한 탁 돈

1978년 연세대학교 공과대학 전자공학과(학사)
1983년 Wayne State University 컴퓨터공학(석사)
1987년 University of Massachusetts 컴퓨터공학(박사)
1987년~1989년 Cleveland 주립대학 조교수
1989년~현재 연세대학교 공과대학 컴퓨터과학과 교수
관심분야 : 병렬처리 컴퓨터 구조 및 알고리즘, 오류보정 시스템, VLSI 설계, HCI