컴퓨터구조 프로젝트 보고서

Project 4

Cache Design

수업 명: 컴퓨터구조

담당 교수: 이성원 교수님

학과: 컴퓨터정보공학부

학번: 2023202070

이름: 최현진

제출일: 2025.05.

1. Introduction

2. Assignment

2-1. Observation for Program Behaviors

MIPS 기반 파이프라인 구조의 Bubble sort, Random access 두 프로그램의 캐시 구조는 다음과 같다.

|  |  |  |
| --- | --- | --- |
| IM (L1 I-cache) | Direct-mapped | 32 block, 4 word |
| DM (L1 D-cache) | 2-way Set-associative | 16 set \* 2 ways |
| MM (L2) unified | 공유 Memory (128bit) | 4 word per line |

IM은 명령어가 대부분 순차적으로 접근되므로, 충돌 가능성이 낮아 Direct-Mapped 구조로도 충분한 성능을 낸다. DM는 불규칙한 접근이 많아, 같은 set index를 가진 다른 주소들의 충돌을 줄이기 위해 2-Way Set-Associative 구조를 사용한다. MM은 캐시 miss 시 데이터를 공급하는 공통 저장소로, block 단위(4-word)로 응답하는 역할을 한다.

시뮬레이션에 사용된 필드 정보는 다음과 같다.

<IM>

|  |  |
| --- | --- |
| i\_cpu\_rd | read enable |
| o\_cpu\_instr | fetch된 명령어 |
| w\_hit | cache hit |
| w\_miss | cache miss |
| o\_cpu\_wait | IM miss 시 대기 |
| o\_l2\_rd | L2에 읽기 요청 신호 |
| o\_l2\_addr[31:] | L2에 접근할 주소 |

시뮬레이션 예상:

w\_hit = 1 → 바로 o\_cpu\_instr valid

w\_miss = 1 → o\_cpu\_wait 활성, L2 대기 발생

<DM>

|  |  |
| --- | --- |
| i\_cpu\_rd, i\_cpu\_wr | read, write enable |
| o\_l2\_rd, o\_l2\_wr | L2에 읽기/쓰기 요청 신호 |
| w\_d0\_hit, w\_d1\_hit | 2-way cache hit |
| w\_miss | cache miss |
| o\_cpu\_wait | DM miss 시 대기 |
| o\_cpu\_instr | fetch된 명령어 |
| o\_l2\_addr | L2에 접근할 주소 |
| i\_cpu\_data[31:0] | SW용 저장 데이터 |
| o\_cpu\_data | LW된 데이터 |

시뮬레이션 예상:

w\_d0\_hit or w\_d1\_hit = 1 → hit

w\_miss = 1 → L2 접근 필요

<MM>

|  |  |
| --- | --- |
| i\_DM\_Read, i\_DM\_Write | DM이 L2에 요청할 때 신호 |
| i\_DM\_addr[31:0] | 주소 |
| i\_DM\_data[127:0] | 쓰기 데이터 |
| o\_DM\_data[127:0] | 읽은 데이터 |
| o\_DM\_ready | 메모리 응답 완료 |
| i\_IM\_Read, i\_IM\_addr, o\_IM\_data | IM용 L2 인터페이스 |

시뮬레이션 예상:

i\_DM\_Read = 1 → DM cache miss 발생 시

o\_DM\_ready = 1 → L2 응답 완료, 캐시 블록 채움

2-1-1. Bubble Sort

◼ How I/D L1 hit operates, how I/D L1 miss operates

◼ When and how main memory (L2) operates

<IM miss / MM operates>

텍스트, 스크린샷, 번호, 평행이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

시뮬레이션 시작 시, IM은 0x00부터 0x0C까지 저장된 블록 단위로 L2(MM)에 접근하여 4개의 명령어를 가져온다. 위 결과 화면은 PC 0x00~0x0C 실행 후, 다음 명령어(PC = 0x10)를 실행하기 위해 IM에서 miss된 상황이다. 이에 따라 IM은 0x10~0x1C 범위의 명령어를 다시 L2로부터 가져오기를 시작하며, MM에 접근하는 동안 IM과 DM은 wait 중이다.

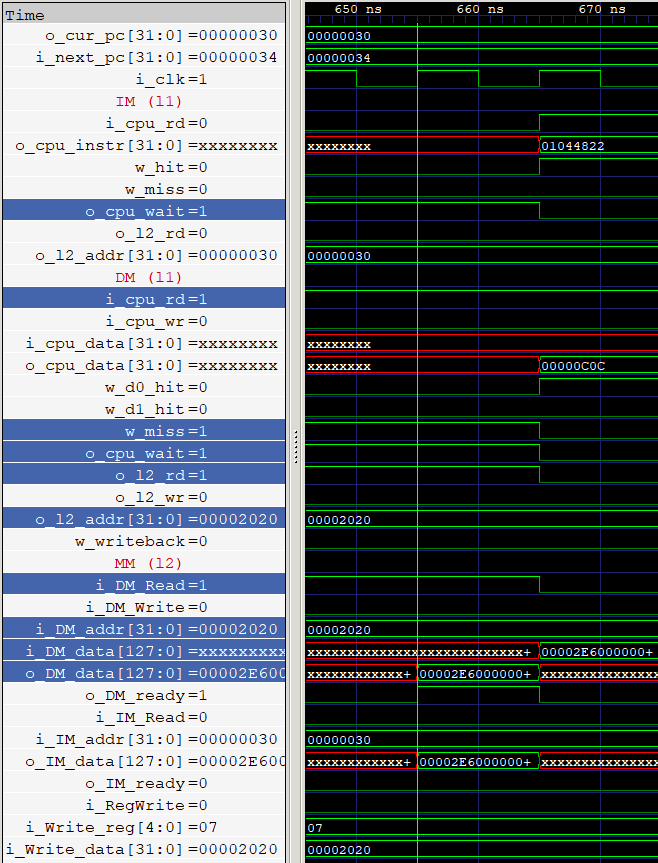
<IM hit operates>

텍스트, 스크린샷, 번호, 소프트웨어이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

명령어가 IM cahce에 있을 때 IM은 hit이고 명령어가 바로 fetch된다. IM에서 바로 응답되어 wait이 0이다. 이 때, DM은 wait 중이고, DM의 rd나 wr 신호는 0이며, MM의 Read나 Write 신호도 0이다.

<DM miss / D0 hit / MM operates>



버블 소트 내 lw $7, 0($6) 명령어의 실행 중에, 해당 주소에 대한 DM miss가 발생하였고 MM로부터 읽기를 시도한다. o\_DM\_ready = 1가 되어 데이터가 준비된다. 이후 다음 클럭에서 바로 D0 hit(w\_d0\_hit=1)가 발생하여 DM에서 MM에서 읽어온 00000C0C을 확인했다.

텍스트, 스크린샷, 번호, 평행이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

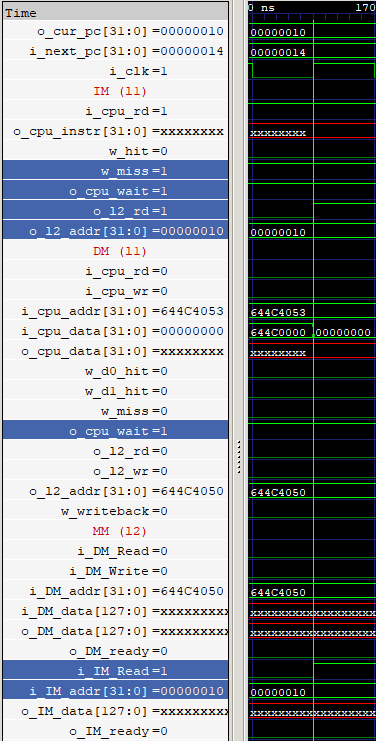
이후 lw 명령어의 동작대로 $7에 로드된 데이터 00000C0C가 저장된다.

2-1-2. Random Access

◼ How I/D L1 hit operates, how I/D L1 miss operates

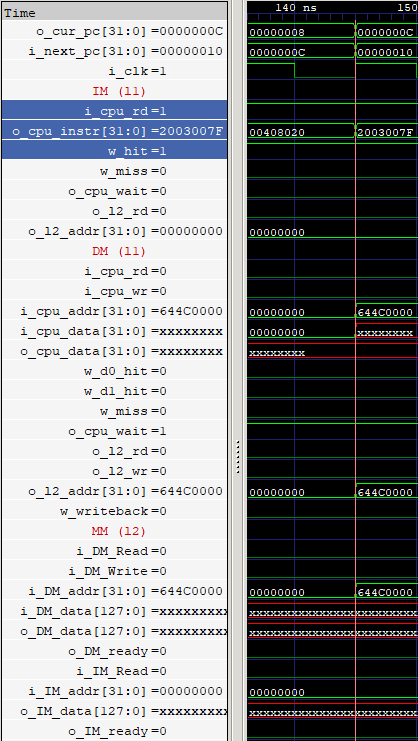
◼ When and how main memory (L2) operates

<IM miss / MM operates>



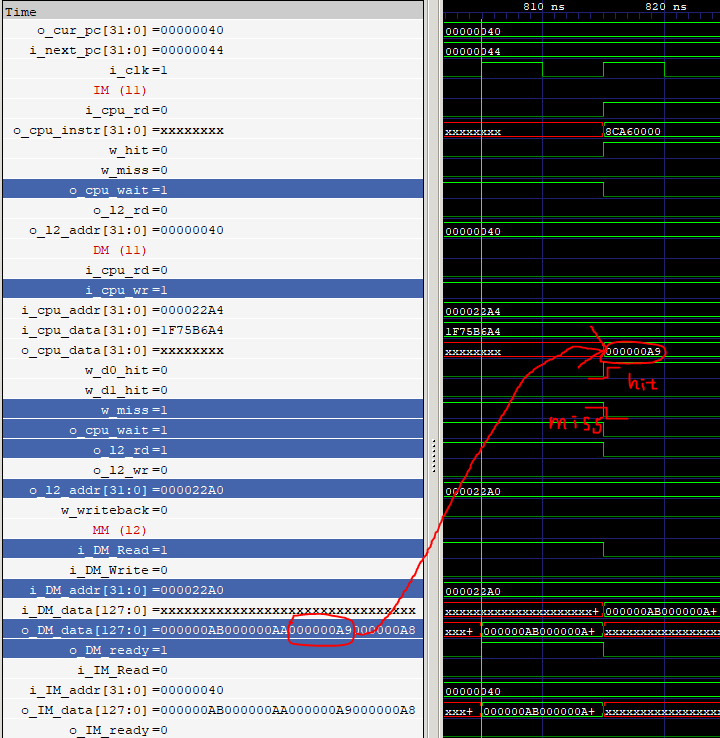
시뮬레이션 시작 시, IM은 0x00부터 0x0C까지 저장된 블록 단위로 L2(MM)에 접근하여 4개의 명령어를 가져온다. 위 결과 화면은 PC 0x00~0x0C 실행 후, 다음 명령어(PC = 0x10)를 실행하기 위해 IM에서 miss된 상황이다. 이에 따라 IM은 0x10~0x1C 범위의 명령어를 다시 L2로부터 가져오기를 시작하며, MM에 접근하는 동안 IM과 DM은 wait 중이다.

<IM hit operates>



명령어가 IM cache에 있을 때 IM은 hit이고 명령어가 바로 fetch된다. IM에서 바로 응답되어 wait이 0이다. 이 때, DM은 wait 중이고, DM의 rd나 wr 신호는 0이며, MM의 Read나 Write 신호도 0이다.

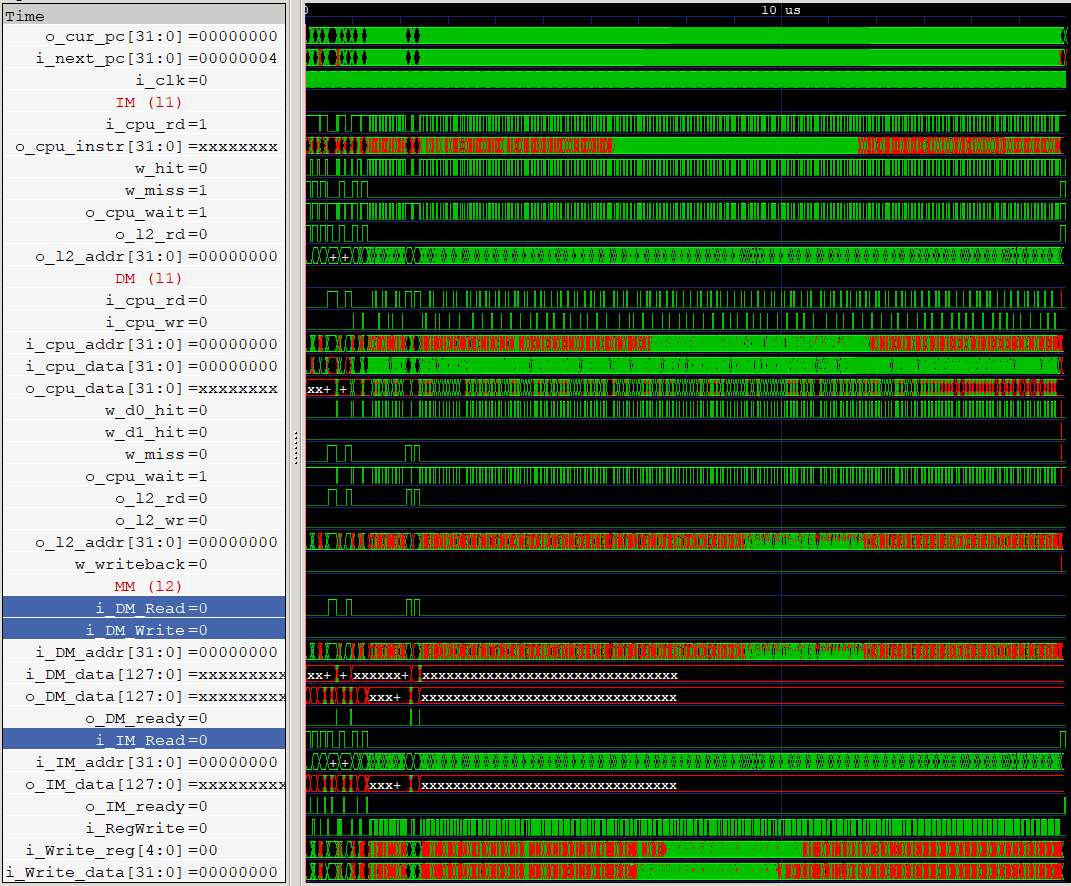
<DM miss / D0 hit / MM operates>



랜덤 액세스 프로그램의 lw 명령어 실행 중, 해당 주소가 DM에 존재하지 않아 DM miss가 발생하고, MM에 읽기 요청(i\_DM\_Read = 1)을 보낸다. 이후 MM으로부터 응답 신호인 o\_DM\_ready = 1이 되어 데이터가 준비된다. 다음 클럭에서 바로 해당 데이터 블록이 D0 way에 hit(w\_d0\_hit = 1)되며, 로드된 값 0x000000A9가 정상적으로 캐시로 전달되어 읽은 것을 확인했다.

2-1-3. 두 프로그램 비교 분석

◼ The differences between Bubble sort and Random access in terms of cache behaviors



텍스트, 스크린샷, 폰트, 블랙이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Bubble Sort 실행 시, Instruction Memory(IM)과 Data Memory(DM)의 L2 접근은 대부분 프로그램의 초반부에 집중되어 있음을 확인하였다. 총 실행 사이클 수는 1584 cycle로 측정되었다.

IM의 경우, 프로그램이 반복문 기반의 구조를 가지므로, 초기 반복문에 필요한 명령어들을 4개씩L2에서 가져와 캐시에 저장한 후, 이후에는 캐시에 저장된 명령어들을 반복적으로 재사용하게 된다. 이로 인해 초기 몇 차례 miss 이후 대부분 hit이 발생하며, 소트가 완료되고 프로그램 종료 직전 마지막 명령어를 가져올 때 후반부에 한 번의 miss가 추가로 발생한다.

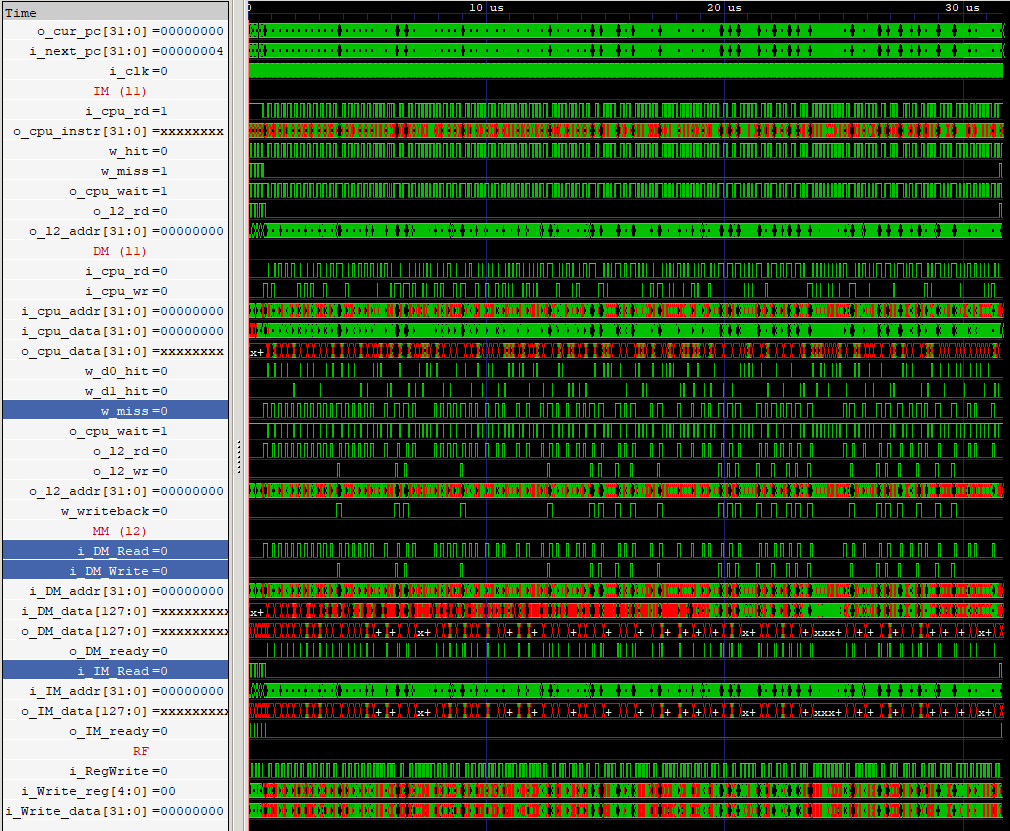
DM의 경우도 유사하게, Bubble Sort는 배열을 순차적으로 접근하는 구조이기 때문에, 초기 정렬 대상 데이터들을 L2에서 한 번씩 읽은 후에는 캐시 hit이 지속된다. 또한, Write-Back 정책에 의해 write는 캐시에만 반영되고 L2에는 아직 반영 안 한 상태에서, 캐시가 꽉 차지 않았고 교체가 일어나지 않아 writeback이 불필요하게 되어 L2에 대한 write 시도는 일어나지 않는다. D1 또한 채워지지 않는다.

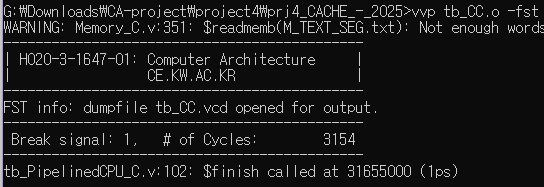
결과적으로, Bubble Sort는 IM과 DM 모두에서 캐시 hit률이 매우 높은 구조를 가지며, 인접한 두 데이터를 반복적으로 비교하며 정렬을 수행하는 알고리즘이다. 이처럼 연속적인 메모리 접근 패턴을 가지기 때문에, 현재 접근한 데이터와 바로 옆에 있는 인자 역시 캐시에 함께 포함되어 있다면 캐시 hit 확률이 크게 증가하는 것이다.

◼ What if data size increase?

Bubble Sort는 인접한 두 데이터를 반복적으로 비교하며 정렬을 수행하는 알고리즘으로, 높은spatial locality을 가진다. 데이터 크기가 증가하더라도 동일한 패턴의 연속적인 메모리 접근이 지속되므로, 일정 수준까지는 캐시 hit 비율이 유지된다. 특히, Block size가 충분히 커지고 캐시 용량이 데이터 집합을 수용할 수 있다면 miss가 크게 증가하지 않는다.

그러나, 데이터 크기가 캐시 용량을 초과하게 되면, 각 반복에서 접근하는 데이터가 캐시에 존재하지 않게 되고, 이로 인해 L2 접근 빈도와 캐시 교체 발생 횟수가 증가하여 AMAT와 실행 시간 모두 증가하게 된다.





Random Access 실행 시, Instruction Memory(IM) L2 접근은 버블 소트와 마찬가지로 대부분 프로그램의 초반부에 집중되어 있다. 그러나, Data Memory(DM)의 L2 접근은 버블 소트보다 훨씬 많이 프로그램 전반적으로 수행된다. 총 실행 사이클 수는 3154 cycle로 측정되었다.

IM의 경우, 프로그램이 반복문 기반의 구조를 가지므로, 초기 반복문에 필요한 명령어들을 4개씩L2에서 가져와 캐시에 저장하고, 프로그램 종료 직전(break) 가장 후반부에 IM miss가 추가로 발생하는 구조는 버블 소트와 같다.

DM의 경우, 버블 소트와 다르게 D0 hit과 D1 hit, MM에 대한 Read와 Write 시도가 프로그램 전반적으로 일어난다. Random Access 프로그램은 Bubble Sort와 달리, 배열을 순차적으로 접근하지 않고 불규칙한 주소들에 대해 load/store 명령어를 반복적으로 수행한다. 이로 인해 Set index가 겹치는 서로 다른 주소들이 DM 내 같은 set에 반복적으로 접근하게 된다. 그 결과, D0 또는 D1 중 하나에 먼저 저장된 데이터가 LRU 정책에 따라 교체되면서 D1 hit도 발생하고, 충돌(conflict miss)이 많아져서 L2에 대한 Read 요청이 자주 발생한다. 또한, sw 명령어가 많기 때문에 Write-Back 정책 하에서도 Dirty Block이 교체되며 L2에 Write가 발생하게 된다.

결국, Random Access는 불규칙한 메모리 접근으로 인해 다양한 set과 way를 교체하며 사용하는 구조를 가지므로, Bubble Sort보다 DM의 충돌과 교체가 빈번하다. 이로 인해 D1 hit 발생, L2 read/write 시도, write-back 발생 등이 프로그램 전체에 걸쳐 분산되어 발생한다

◼ What if data size increase?

Random Access는 불규칙한 주소에 대해 load/store 명령어를 수행하는 구조로, 공간 및 시간 지역성이 낮은 프로그램이다. 데이터 크기가 증가할수록 서로 다른 주소가 동일한 cache set에 매핑될 가능성이 높아지면서 conflict miss가 급격히 증가하고, 교체도 더 자주 발생하게 된다. 또한 Write-Back 정책 하에서도 dirty block이 자주 교체되어 L2 write 트래픽도 증가하게 된다.

결과적으로, 데이터 크기 증가에 따라 캐시의 효율이 빠르게 저하되며, L2 접근 지연, AMAT 상승, 전체 실행 시간 증가로 이어진다.

3. 문제점 및 고찰

⚫ 프로젝트 내용 전체 정리 및 고찰 ⚫ Benchmarks에 최적화된 cache와 정렬 프로그램에 최적화된 cache 비교 및 분석