**Lab5 Report**

**20230642 이채영, 20230683 박한비**

1. Introduction

Lab4-2에서는 RISC-V instruction을 기반으로 하는 5-stage Pipelined CPU에 4-way Cache를 추가하였다. CPU는 memory에 값에 접근하기 위해 cache에 접근하고, miss가 발생한 경우 data memory에서 값을 얻어온다. Cache는 자주 쓰이는 데이터를 임시로 보관해놓는 공간으로, 메모리 접근에 사용되는 시간을 줄일 수 있다.

1. Design

이번 Lab5에서는 이전 Lab4-2까지 구현한 pipelined CPU에 이어 Cache를 구현하였다. Cache에서 miss 발생 시 data memory에 접근하여 값을 가져오고, 이를 처리하는 동안 CPU는 stall한다. 기본적으로 Lab4-2 report에서 제시한 설계도 바탕에 Data memory가 들어간 부분을 Cache로 대체하였고, cache\_stall 신호를 통해 stall을 구현했다.

텍스트, 도표, 평면도, 개략도이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Cache의 내부 구조는 교안 Lec13-Memory Hierarchy의 36쪽을 참고하여 4-way로 구현하였다. Set은 총 4개, 각 set당 cache line은 4개이고, 각 data bank는 4 byte이다. 교안에 나타난 구조에 write-back 여부를 판단할 수 있도록 dirty bit를 추가하였고, cache miss 발생 시 replacement를 구현하기 위해 각 line마다 LRU를 추가하였다.

32-bit의 address에서 set index를 나타내는 데 2-bit, block offset을 나타내는 데 2-bit, tag를 나타내는 데 나머지 26-bit를 사용하였다.

1. Implementation
2. cpu.v

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

이전 Lab에서와 다르게 load/store instruction에서 data memory에 접근할 때 cache에 우선적으로 접근한다. Cache miss가 일어난 경우 Cache 모듈 내부에서 Data memory에 접근한다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Cache miss 발생 시 data memory에 접근하는데, 이때 추가적인 사이클이 발생한다. 따라서 데이터를 처리하는 동안 CPU는 stall해야 한다. Cache의 output value였던 is\_ready, is\_output\_valid, is\_hit이 모두 1이 아니라면 CPU를 stall하여 다음 instruction을 CPU가 처리하지 못하도록 막는다. 이 cache stall 신호를 is\_cache\_ready라고 정의하였다. 그리고 pipeline register들(IF/ID, ID/EX, EX/MEM, MEM/WB registers)이 is\_cache\_ready 신호가 1일 때만 업데이트되어 다음 instruction이 fetch되도록 구현하였다.

1. Cache.v

Cache hit/miss일 때의 데이터 처리를 위해 FSM으로 구현하였다. NONE, ACCESS, WRITE\_BACK, ALLOCATE 총 4개의 state로 구성하였다.

1. NONE

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

MEM stage에 위치하는 instruction이 mem access, 즉 mem read 혹은 write를 모두 필요로 하지 않는 경우로, mem access가 들어오는지 체크하는 상태이다. mem access signal(mem\_read 또는 mem\_write)이 감지된 경우 next state가 ACCESS로 넘어가고 아닌 경우 계속해서 NONE을 유지한다.

1. ACCESS

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

mem access를 필요로 하는 것을 감지하여 cache 상에서 처리가 가능한지 cache의 hit 여부를 계산한다. hit인 경우 mem access signal이 read, 혹은 write인지에 따라 해당 cache line의 값을 dout(cache)으로 읽어오거나 positive edge 시점에 synchrous하게 din(cache)의 값을 cache line에 쓸 수 있도록 cache\_update 관련 레지스터들에 업데이트할 값들을 저장해놓는다. 또 hit인 경우에는 해당 mem access 처리를 끝낸 것이므로 next state는 NONE으로 돌아가 다시 mem access signal를 기다린다. miss인 경우 해당 addr에 해당하는 mem data가 저장될 cache line이 dirty 상태에 따라 dirty라면 WRITE\_BACK으로, 아니라면 ALLOCATE를 next state로 한다.

1. WRITE\_BACK

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

새로운 addr의 data를 저장할 cache line의 기존 data를 그에 맞는 adddress의 memory에 업데이트할 수 있도록 한다. Data Memory가 새로운 instruction을 받을 준비가 되지 않은 경우 아무 일도 하지 않고 next state를 WRITE\_BACK으로 설정해 Data Memory가 준비될 때까지 기다릴 수 있도록 한다. Data Memory가 준비가 되었다면 Data Memory에 write signal과 write할 address, write할 data를 cache line으로부터 불러와 전달해준다. 이후 next state를 ALLOCATE로 하여 다음 단계로 넘어갈 수 있도록 한다.

1. ALLOCATE

텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

Memory에서 새로운 addr의 data를 불러와 cache line에 저장할 수 있도록 한다. Data Memory에 read signal과 read할 address를 전달해준다. Data Memory가 새로운 instruction을 받을 준비가 되지 않은 경우 아무 일도 하지 않고 next state를 ALLOCATE로 설정해 Data Memory가 준비될 때까지 기다릴 수 있도록 한다. Data Memory가 준비가 되었다면 Data Memory에서 읽어서 출력한 data를 positive edge 시점에 synchrous하게 cache line에 쓸 수 있도록 cache\_update 관련 레지스터들에 업데이트할 값들을 저장해놓는다.

이후 하여 miss 처리를 완료 후 돌아갈 수 있도록 next state를 ACCESS로 설정해주었다.

텍스트, 폰트, 문서, 스크린샷이(가) 표시된 사진

자동 생성된 설명텍스트, 스크린샷, 폰트, 번호이(가) 표시된 사진

자동 생성된 설명

위의 state마다의 처리들은 asynchronous하게 처리되며 synchronous하게 업데이트 되어야할 값들은 always@(posedge clk)문에서 처리한다. current state가 next state로 업데이트되도록 하고 cache로의 write 처리가 필요한 경우 cache\_update signal을 받아 올바른 cache line의 각 데이터들을 업데이트할 수 있도록 하였다. 또한 LRU bank의 업데이트를 위하여 current state가 ACCESS였을 경우 hit 여부와 새로운 addr의 data를 저장할 cache line인지 여부에 따라 각 cache line의 LRU 상태를 업뎅이트한다. 또한 hit ratio를 확인하기 위해 hit count와 miss count 또한 hit 여부에 따라 업데이트한다.

1. Discussion
2. Associative Cache와 Direct-mapped Cache의 비교

Direct-mapped Cache는 addr에서 추출한 idx를 entry로 하는 data(및 tag, valid, dirt 등의) bank가 하나만 있기 때문에 idx가 같다면 무조건 같은 공간에 저장해야 한다. 따라서 memory상으로 접근할 두개의 다른 addr의 idx가 같다면 무조건 conflict miss가 발생하게 된다. 그러나 Associative Cache는 작은(a-way Set Associative Cache의 경우 각 capacity가 (총 capacity)/a와 같은) Direct-mapped Cache를 여러개 붙인 구조로, 따라서 idx를 entry로 하는 별개의 data bank가 여러 개(a-way Set Associative Cache의 경우 a개) 있다. 이로 인해 두 다른 addr의 idx가 같더라도 각자 다른 cache line에 저장해놓을 수 있어 conflict miss를 줄일 수 있다. 아래는 Direct-mapped Cache의 Naïve\_matmul\_unroll 및 opt\_matmul\_unroll 테스트벤치 실행 결과로 conclusion의 4-way cache 결과와 비교한다면 naïve와 opt 모두 4-way cache의 hit ratio가 더 높은 것을 알 수 있다.

텍스트, 스크린샷, 패턴, 패브릭이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 메뉴, 모노크롬이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. Replacement policy

LRU(Least recently used line)을 찾기 위해 각 Cache line마다 정수형 배열 LRU 도입하여 hit/miss마다 계산하였다. Cache hit 발생 시 해당 라인의 LRU는 0으로 초기화하고, 나머지 line들의 LRU는 1씩 증가시킨다. Cache miss 발생 시 바꾼 Line의 LRU를 0으로 초기화하였다. 이렇게 되면 least recently used line이 가장 큰 LRU 값을 갖게 되므로, cache miss 발생 시 LRU 값이 가장 큰 line으로 replace하도록 구현하였다.

1. Naïve matmul vs Optimized matmul

각각 8x8 matrix인 A와 B를 multiplication하는, A x B의 경우를 생각해보자.

스크린샷, 라인, 다채로움, 도표이(가) 표시된 사진

자동 생성된 설명

그림과 같이 Naïve matmul은 우리가 아는 방식대로 matrix multiplication을 수행한다. A의 각 row에 대하여 B의 각 column을 곱하여 결과값으로 나오는 matrix의 각 element가 될 값을 구하는 연산을 수행한다.

스크린샷, 라인, 도표, 평행이(가) 표시된 사진

자동 생성된 설명

Optimized matmul은 각 matrix를 특정한 사이즈의 tile 여러개로 나누어 tile별로 matrix multiplication을 수행하고 이를 합치는 방식으로 연산을 수행한다. 8x8 matrix인 A와 B의 경우에는 A와 B를 각각 4x4 matrix 4개로 나누어 총 4번의 부분 matrix multiplication를 수행한다.

이로 인해 cache의 hit ratio가 달라지는 이유는 cache에 데이터가 저장되는 형식과 메모리 접근 순서 때문이다. 해당 Lab에서 구현한 cache의 경우 한 cache line이 16 byte, 즉 4개의 word(4byte)만큼의 데이터를 저장할 수 있다. cache는 한 cache line에 인접한 address의 연속된 데이터를 통째로 저장하기 때문에 matrix의 형태로 본다면 한 row의 연속된 네 개의 데이터(index 상 row 범위를 넘어설 경우 인접한 두 row를 지나서)를 cache line에 한번에 저장할 것이다.

Naïve matmul은 한 row(8x1)에 각 column(1x8)을 곱하는 연산이 반복적으로 수행되는데 row는 한 cache line에 4개씩 연속적으로 같은 row에 해당하는 값을 저장할 수 있으나 column은 각 cache line 당 같은 column에 해당하는 값을 한 개씩 개별적으로 저장하므로 한 column에 접근하려면 8개의 cache line을 사용해야 한다. 따라서 연산을 수행할 때 column 접근에 많은 cache line을 사용하게 되는데 이렇게 저장된 데이터를 저장된 다음 column의 access 경우에도 그대로 사용할 수 있도록 하면 miss를 줄일 수 있으나 이미 많은 개수의 cache line을 차지하고 있어 row의 access 중 evict될 가능성이 높다.

Optimized matmul의 경우 연산을 작은 4x4 연산 네개로 나누어 진행하기 때문에 각 연산에서 column access에 사용되는 cache line의 개수가 적다(네개). 따라서 다른 evict될 가능성이 낮고 4x4 연산 내의 각 row에 column을 곱할 때마다 이미 column이 cache에 저장되어 있을 확률이 높다. 따라서 Optimized matmul이 cache를 더 효율적으로 이용할 수 있어 hit ratio가 더 높게 측정된다.

conclusion의 테스트벤치 결과와 본다면 Optimized matmul이 Naïve matmul보다 hit ratio가 더 높은 것을 알 수 있다

1. If the # of sets and # of ways are changed, what happens?

capacity는 일정한 상태에서 각각 set 혹은 way의 개수가 많아지는 경우를 보겠다. Way 개수가 많아지고 set의 개수가 적어지는 경우 인덱스가 겹쳐도 conflict miss의 경우를 줄일 수 있다. 그러나 필요한 comparator, multiplexer 등이 늘어나고 로직 복잡도와 cost가 커질 수 있다. Set 개수가 많아지고 way의 개수가 적어지는 경우 replacement policy가 단순해지고 cost를 줄일 수 있으나 conflict miss의 경우가 늘어날 수 있다.

아래는 2-way cache와 8-way cache의 Naïve\_matmul\_unroll 및 opt\_matmul\_unroll 테스트벤치 실행 결과로 왼쪽부터 순서대로 2-way naïve/opt, 8-way naïve/opt 에 해당한다.

텍스트, 스크린샷, 패턴, 책이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 메뉴, 책이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 패턴, 패브릭이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

1. Conclusion

4-way cache의 naïve\_matmul\_unroll 및 opt\_matmul\_unroll 테스트벤치 실행 결과는 아래와 같다.

텍스트, 스크린샷, 디자인이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.텍스트, 스크린샷, 패턴, 모노크롬이(가) 표시된 사진

AI가 생성한 콘텐츠는 부정확할 수 있습니다.

Directed-mapped cache, 2-way Cache, 4-way Cache, 8-way Cache에 대해naïve\_matmul\_unroll.mem과 opt\_matmul\_unroll.mem 테스트벤치 실행 결과를 비교한 표는 아래와 같다. 총 memory access(total == hit\_count + miss\_count) 수는 모든 경우에서 2499로 나왔다.

1. naïve\_matmul\_unroll

|  |  |  |  |
| --- | --- | --- | --- |
| # of way | Total cycle | # of hit | Hit\_ratio |
| 1(direct-mapped) | 72432 | 1686 | 0.67467 |
| 2 | 52601 | 1886 | 0.754702 |
| 4 | 38891 | 1959 | 0.783513 |
| 8 | 44350 | 1855 | 0.742297 |

1. opt\_matmul\_unroll

|  |  |  |  |
| --- | --- | --- | --- |
| # of way | Total cycle | # of hit | Hit\_ratio |
| 1(direct-mapped) | 77747 | 1604 | 0.641857 |
| 2 | 55348 | 1864 | 0.745898 |
| 4 | 25286 | 2258 | 0.903561 |
| 8 | 25286 | 2258 | 0.906861 |

Lab5를 통해 cache를 구현해보고 cache의 way 개수와 replacement 정책등의 선택을 고민해보면서 cache에 대한 이해도를 높이고 cache를 설계할 때 어떠한 점을 고려해야 할 지 생각할 수 있는 시간을 가지게 되었다.