Cache

Homework 4
모바일시스템공학과
32143153 이대훈

1. Project Introduction

그 동안 Memory에서 값을 불러오면서 작업을 했다면 이제는 CPU내의 Cache에서 값을 불러와 사용할 수 있도록 한다. 이 때 Cache에 값이 들어가 있지 않은 경우에 cold miss라 하고 값이 들어가 있지만 틀린 값이 들어가 있는 경우를 conflict miss라 하여 replacement가 일어난다. Cache를 사용함으로써 Memory에 접속하는 횟수가 줄어들어 효율적으로 사용할 수 있다.

2. Motivation

Memory에서 값을 가지고 오거나 Memory에 쓸 때에는 Memory에 직접 접속해야 했다. 하지만 Cache를 사용함으로써 우리가 Memory에 직접 접속하는 횟수는 줄어들고 그만큼 효율이 커진다.

3. Concepts used in Multi cycle

Pip-line을 바탕으로 한다.
처음 Cache를 초기화한 상태에서 값이 입력이 없이 처음으로 들어갈 때 cold miss라고 한다.
Hit는 Cache의 값을 Addr를 통해 나타낸 값으로 확인하여 찾았을 때 맞는 값(Tag)이 있을 때를 말한다.
Conflict miss는 Cache에서 Addr를 통해 나타낸 Index를 보고 이 때 Tag값이 다르면 일어난다. 이런 경우 가지고 있던 값이 다르므로 기존에 가지고 있던 Cache값을 Memory에 저장하고 새로운 값을 Cache에 불러 온다.
4-Way의 경우 oldest라는 변수를 사용하여 4개의 way에서 어떤 way가 바뀌는지를 설정해 준다. 이 때 LRU방식을 사용한다. Hit인 경우 sca를 1로 변환하고 기억하고 있다가 conflict_miss에서 oldest가 가르키는 곳을 보고 이때 sca가 1이면 0으로 값을 바꾸고 oldest는 다음으로 넘겨준다. sca가 1인 경우에는 다음번에 다시 쓰일 확률이 있으므로 한번 더 기회를 주는 것이다.
Cache를 사용할 때 Wirteback을 사용하여 Memory값에 저장한다. Writeback을 사용하면 Writethrough와 달리 값이 바뀔 때마다 동시에 Cache와 Memory를 바꾸는 것이 아니라 Cache값을 바꾼 후 그 Cache block이 바뀔 때 Memory에 Cache block을 update 시켜준다. 이를 확인하기 위해 dirty bit을 사용하는데 dirty bit은 Cache와 Memory간의 값이 다르다는 것을 나타낼 때 1, 같을 때를 0으로 한다.

4. Program Structure

main fuction에서 두 개의 while loop을 사용
전역 변수 선언
Structure 선언 및 member 입력
Open file

각각의 instruction을 memory에 저장

Fetch
(이 때 memory가 아닌 cache에서 불러오도록 Readmem 함수 사용)
Decode(control, J type)
Exe(Data dependancy)
Memory
(이 때 memory가 아닌 cache에서 불러오도록 Readmem, Writemem 함수 사용)
WriteBack

Latch Update

4-Way를 사용한다.
Readmem : Addr를 사용하여 tag, index, dest, offset을 계산한다. for문을 사용하여 모든 way에서 값을 비교할 수 있도록 하였다. 처음 Hit인 경우와 miss인 경우로 구분하고 Miss인 경우에는 또 다시 dirty bit이 1이면서 sca bit이 0인 경우 sca bit 1인 경우 나머지 경우로 구분하여 3가지로 나눈다. dirty bit이 1이면서 sca bit이 0인 경우는 conflict miss가 일어나고 sca bit가 1인 경우에는 sca bit를 0으로 바꾸고 oldest를 증가한다. 나머지의 경우 cold miss를 나타내며 이 때 valid bit을 사용하여 값이 저장되었다는 것을 나타낸다. 여기서 각각 Hit과 cold miss, conflict miss에서는 값을 return해주어 더 이상 for문이 진행하지 않도록 해주었다.

Wirtemem : Readmem과 같으며 다른 점은 Wirtemem에서는 return할 필요가 없다. 따라서 for문을 끝내기위해 return대신 break를 사용하였다. 또한 parameter로 받아온 value를 cache에 저장하는 작업을 한다.

5. Problems and solutions

Memory에서 Cache로 바꾸는 과정에서 그 전 과제까지는 sp와 Memory의 크기를 같게 맞추었다. 하지만 Addr를 받아와 사용할 때 Addr/4를 해야해서 sp를 Memory보다 4배 큰 값으로 설정하였다.
Writethrough에서 Writeback으로 바꿔주기 위해서 dirty bit을 사용하였고 각각의 경우에 dirty bit을 보고 update의 필요성을 판단하였다. 또한 현재의 Cache block을 저장하는 것에 있어서 처음에는 지금 가지고 들어온 주소에 대해서 만들어진 dest를 사용하였는데 이런 경우

Error가 발생 하였다. 현재의 Cache block을 update 해주는 memory의 주소는 현재 들어온 Addr가 아닌 바뀌는 Cache block의 tag와 index를 사용하여 다시 주소 값을 만들어주어 그 주소 값의 memory에 저장하였다. 그 후에 Cache block에는 Memory에서 값을 가지고 오며 이 때 사용하는 Addr가 parameter로 받아온 Addr다.
Hit이면 sca bit이 1로 바뀐다고 생각하여 Writemem과 Readmem 모두에서 Hit인 경우 sca bit를 1로 바꾸었다. 하지만 Writemem의 경우에는 sw와 같은 경우처럼 읽어오는 것이 아닌 기록하는 작업을 하는 것이다. 이때는 Hit지만 sca bit를 1로 바꾸지 않는다.
Index를 계산할 때 Cache의 용량과 cacheline의 크기를 사용하여 구하며 이 때 계산 실수를 하여 디버깅 과정에서 계속 error가 났다. 이런 경우에 gdb를 사용하여 error난 코드를 보고 display 명령어를 사용하여 안의 값을 들여다보았다. 내 경우에는 Index값이 내가 설정한 값보다 큰 값이 들어갔고 이런 경우에 쓰레기 값이 출력되어 error가 발생하였다.

6. Build enviroment

Compilation : Linux Assam, with GCC
To compile : cd test_prog
gcc cache_4way.c, cache_direct.c
To run :
./a.out
input file : simple.bin, simple2.bin, simple3.bin, simple4.bin, fib.bin, gcd.bin, input4.bin

두 경우 모두 256KB로 Cache 크기 설정.

7. Personal feelings

이번 과제를 하면서 Memory에서 불러오는 대신 Cache에서 불러오는 것이 효과적이라는 것을 알았다. CPU에 있는 Cache를 사용하면 Memory에서 값을 불러오는 것보다 시간적으로 단축이 된다. 특히 CPU와 Memory 사이에서 일어나는 병목현상을 줄일 수 있다고 한다. Cache의 크기를 설정 할 때 Cache가 크면 클수록 Miss의 횟수는 줄고 Hit의 횟수가 늘어나 Hit rate이 좋게 나왔다. 하지만 Cache의 용량이 커지면 그만큼 가격이 비싸지기 때문에 최적화하여 사용할수록 좋다. 또한 data store에서 cacheline 크기가 커진다고 좋기만 하지는 않다. 여기서도 마찬가지로 크기가 커지면 담을 수 있는 값이 많지만 한번 update를 할 때 Memory에 저장하는 값도 많아진다. 이처럼 여러 방면에서 생각하여 Hit rate를 높이도록 해야 한다.

8. Screen capture

Readmem

```
int
ReadMem(int Addr)
{
        int tag;
        int index;
        int offset;
        int dest;
        int i;
        int temp;
        int j;
        int h;


        tag = (Addr & 0xFFFF0000) >> 16;
        index = (Addr & 0x0000FFC0) >> 6;
        offset = (Addr & 0x0000003F);
        dest = (Addr & 0xFFFFFFC0);

        for(j=0;j<way;j++)
        {
                h = (oldest[index]%way);

                if((Cache[j][index].tag == tag) && (Cache[j][index].valid)) //Hit
                {
                        Cache[j][index].sca = 1;
                        Hit++;
                        return Cache[j][index].data[offset/4];
                }
                else
                {
                        if((Cache[h][index].dirty) && (Cache[h][index].sca == 0)) // conflict miss
                        {
                                for(i=0;i<16;i++)
                                {
                                        temp = (Cache[h][index].tag << 16) | (index << 6);
                                        M[temp/4 + i] = Cache[h][index].data[i];
                                        MA++;
                                }
                                for(i=0;i<16;i++)
                                {
                                        Cache[h][index].data[i] = M[dest/4 + i];
                                        MA++;
                                }
                                oldest[index]++;
                                conflict_miss++;
                                Cache[h][index].tag = tag;
                                Cache[h][index].dirty = 0;
                                return Cache[h][index].data[offset/4];
                        }
                        else if(Cache[h][index].sca == 1)
                        {
                                Cache[h][index].sca = 0;
                                oldest[index]++;
                        }
                        else // cold miss
                        {
                                for(i=0;i<16;i++)
                                {
                                        Cache[h][index].data[i] = M[dest/4 + i];
                                        MA++;
                                }
                                Cache[h][index].tag = tag;
                                Cache[h][index].valid = 1;
                                oldest[index]++;
                                cold_miss++;
                                return Cache[h][index].data[offset/4];
                        }
                }
        }
}
```

Writemem

```
WriteMem(int Addr, int value)
{
        int i;
        int tag;
        int dest;
        int index;
        int offset;
        int temp;
        int j;
        int h;

        tag = (Addr & 0xFFFF0000) >> 16;
        index = (Addr & 0x0000FFC0) >> 6;
        dest = (Addr & 0xFFFFFFC0);
        offset = (Addr & 0x0000003F);

        for(j=0;j<way;j++)
        {
                h = (oldest[index]%way);

                if(Cache[j][index].tag == tag) //Hit
                {
                        Cache[j][index].data[offset/4] = value;
                        Cache[j][index].dirty = 1;
                        Hit++;
                        break;
                }
                else
                {
                        if((Cache[h][index].dirty) && (Cache[h][index].sca == 0)) // Conflict miss
                        {
                                for(i=0;i<16;i++)
                                {
                                        temp = (Cache[h][index].tag << 16) | (index << 6);
                                        M[temp/4 + i] = Cache[h][index].data[i];
                                        MA++;
                                }
                                for(i=0;i<16;i++)
                                {
                                        Cache[h][index].data[i] = M[dest/4 + i];
                                        MA++;
                                }
                                Cache[h][index].tag = tag;
                                Cache[h][index].data[offset/4] = value;
                                oldest[index]++;
                                conflict_miss++;
                                break;
                        }
                        else if(Cache[h][index].sca == 1)
                        {
                                Cache[h][index].sca = 0;
                                oldest[index]++;
                        }
                        else // cold miss
                        {
                                for(i=0;i<16;i++)
                                {
                                        Cache[h][index].data[i] = M[dest/4 + i];
                                        MA++;
                                }
                                Cache[h][index].data[offset/4] = value;
                                Cache[h][index].dirty = 1;
                                Cache[h][index].tag = tag;
                                Cache[h][index].valid = 1;
                                oldest[index]++;
                                cold_miss++;
                                break;
                        }
                }
        }
}
```

9. 결과 창

< cache_4way.c > - 기본 코드

simple.bin

```
--------------------------------------------------------------------------
Fianl : R[2] = 0
Number of cycle : 12
R-type : 4, I-type : 4, J-type : 0
Total number of instructions : 8
Memory Access : 14
Register Access : 7
Branches : 0
Not-taken branches : 0
Number of Jumps : 0
Hit : 10
Cold Miss : 4
Conflict Miss : 0
Hit Rate : 71.428574
daehoon14@assam:~/test_prog$
```

simple2.bin

```
--------------------------------------------------------------------------
Fianl : R[2] = 100
Number of cycle : 14
R-type : 3, I-type : 7, J-type : 0
Total number of instructions : 10
Memory Access : 18
Register Access : 8
Branches : 0
Not-taken branches : 0
Number of Jumps : 0
Hit : 14
Cold Miss : 4
Conflict Miss : 0
Hit Rate : 77.777779
```

simple3.bin

```
--------------------------------------------------------------------------
Fianl : R[2] = 5050
Number of cycle : 1537
R-type : 410, I-type : 920, J-type : 1
Total number of instructions : 1331
Memory Access : 2150
Register Access : 921
Branches : 101
Not-taken branches : 1
Number of Jumps : 1
Hit : 2145
Cold Miss : 5
Conflict Miss : 0
Hit Rate : 99.767441
daehoon14@assam:~/test_prog$
```

simple4.bin

```
--------------------------------------------------------------------------
Fianl : R[2] = 55
Number of cycle : 296
R-type : 110, I-type : 153, J-type : 11
Total number of instructions : 274
Memory Access : 396
Register Access : 193
Branches : 9
Not-taken branches : 1
Number of Jumps : 11
Hit : 384
Cold Miss : 12
Conflict Miss : 0
Hit Rate : 96.969696
daehoon14@assam:~/test_prog$
```

gcd.bin

```
--------------------------------------------------------------------------------
Fianl : R[2] = 1
Number of cycle : 1294
R-type : 498, I-type : 637, J-type : 65
Total number of instructions : 1200
Memory Access : 1780
Register Access : 827
Branches : 45
Not-taken branches : 28
Number of Jumps : 65
Hit : 1755
Cold Miss : 25
Conflict Miss : 0
Hit Rate : 98.595505
daehoon14@assam:~/test_prog$
```

fib.bin

```
--------------------------------------------------------------------------------
Fianl : R[2] = 55
Number of cycle : 3173
R-type : 1200, I-type : 1697, J-type : 164
Total number of instructions : 3061
Memory Access : 4268
Register Access : 2131
Branches : 54
Not-taken branches : 55
Number of Jumps : 164
Hit : 4255
Cold Miss : 13
Conflict Miss : 0
Hit Rate : 99.695412
daehoon14@assam:~/test_prog$
```

input4.bin

```
--------------------------------------------------------------------------------
Fianl : R[2] = 0
Number of cycle : 27430555
R-type : 10153047, I-type : 13219823, J-type : 103
Total number of instructions : 23372973
Memory Access : 34547243
Register Access : 18288211
Branches : 2028789
Not-taken branches : 910
Number of Jumps : 103
Hit : 34532615
Cold Miss : 13536
Conflict Miss : 1092
Hit Rate : 99.957657
daehoon14@assam:~/test_prog$
```