CSED211 LAB3 report

20210054 정하우

<Lab 시간을 통해 배운점>

A lab을 배울때는 cache에서 값이 어느set에 가고 어느 line에 들어가는지 확실하게 알 수 있게되어 cache의 전반적인 개념을 잡은데 좋았다.

B lab을 배울때는 값을 다루는 여러 경우 중에서 행렬값을 다룰 때 hit와 miss가 어떻게 나오는지 배웠다. 행렬이라는 좋은 예시를 바탕으로 hit와 miss가 일어나는 메커니즘을 배우니 확실하게 알 수 있게 되었다.

수업을 들어본 결과 이번 cache lab을 하면 전반적인 cache 에 대한 지식이 확실하게 잡힐 것 같았다.

A.

이 lab은 cache가 작동하는 방식을 구현하는게 과제다. 주소를 받고, tag와 set 를 얻어서 내가 직접 동적할당한 cache에 넣어야 한다.

```
int tag = address >> (s + b);//tag bit 추출
int set_index = (address >> b) - (tag << s);//set_index bit 추출
```

Tag와 set는 다음과 같이 추출 가능하다.

```
bool hit_cache = false;//보고자 하는 set 안에 있는 모든 line을 검사했을때
bool eviction_cache = true;//보고자 하는 set에 이미 있는 line을 다른 line

int pos = 0;//이 반복문을 다 돌았을때 최종적으로 있었던 위치. miss가 났을때

int i;
for (i = 0; i < E; i++)//보고자 하는 set 안에 있는 모든 line을 검사하는 반
{
    if (cache[set_index][i].time < time_temp)
    {
        time_temp = cache[set_index][i].time;
        pos = i;//hit가 될 경우에는 위치정보를 알 필요가 없고, valid가 0일
    }

    if (cache[set_index][i].valid != 0)//vaild가 1일때이다. 이 경우 bloc
    {
        if (cache[set_index][i].tag == tag)//block에 원하는 정보가 있는
        {
            hit_cache = true;
            break;//밑에서 hit_count++말고는 더 이상 진행해야하는 일이 없
        }
    }
    else//vaild가 0일때다. 이 코드에서 set배열(cache[i][]배열)에는 line이
    {
        pos = i;//i 보다 큰 경우 모두 valid가 0일것이므로 뒤에는 볼 필요도
        eviction_cache = false;//이 위치에 있는 비어있는 line에 정보를 넣
        break;//뒤에 부분은 더이상 검사할 필요가 없다. break를 이용해 반복
    }
}
```

Set의 위치를 파악했으면, 안에있는 line들을 for문을 통해 검사해 준다.

```
if (cache[set_index][i].valid != 0)//vaild가 1일때이다. 이 경우
{
    if (cache[set_index][i].tag == tag)//block에 원하는 정보가
    {
        hit_cache = true;
        break;//밑에서 hit_count++말고는 더 이상 진행해야하는 일
    }
}
```

두번째 if문이다. 이 경우는 valid=1이고 tag값도 일치하므로 hit임을 알 수 있다.

```
if (hit_cache)
{
    hit_count++;
    cache[set_index][i].time = c_time;
}
```

Hit_count++해주고, 가장 최근에 참조하였으므로 time값을 업데이트 해준다.

```
else//vaild가 0일때다. 이 코드에서 set배열(cache[i][]배열)에는 line
{
    pos = i;//i 보다 큰 경우 모두 valid가 0일것이므로 뒤에는 볼 필요
    eviction_cache = false;//이 위치에 있는 비어있는 line에 정보를
    break;//뒤에 부분은 더이상 검사할 필요가 없다. break를 이용해 반
}
```

Valid=0일때이다. 이 경우 아무것도 생각하지 말고 그냥 이 line에 무조건 정보를 넣어야한다.

```
if (cache[set_index][i].time < time_temp)
{
    time_temp = cache[set_index][i].time;
    pos = i;//hit가 될 경우에는 위치정보를 알 필요가 없고, valid
}
```

만약 위의 경우 모두 일어나지 않으면 break구문이 없으므로 for문은 끝까지 돈다. 즉 모든 line을 검사하는데, 이 경우 위의 구문을 통해서 가장 옛날에 참조했던 line을 구할 수 있다.

```
else//miss를 한 경우이다.
{
    miss_count++;

    if (eviction_cache)//교체할 line이 있는 경우이다.
    {
        eviction_count++;
    }
    else//교체할 line이 없는 경우이다.
    {
        /*정보가 없는 line에 정보를 넣는다*/
        cache[set_index][pos].valid = 1;
    }
    cache[set_index][pos].tag = tag;//교체된 line에 있는 tag 값을 최근에 추가한 정
    cache[set_index][pos].time = c_time;
}
```

Miss한 경우에서 valid=1이면 line이 교체됐다는 뜻이다. 따라서 eviction_count++해 주고, 만약

valid=0이면 비어있는 line에 값을 넣는 것이다.

이 알고리즘에 따르면 cache에서 miss발생 시 line을 교체할 때 항상 과거 마지막 사용 시점이 가장 오래된 라인을교체하는 LRU 정책을 따른다.

```
Points (s,E,b)    Hits  Misses  Evicts    Hits  Misses  Evicts
     3 (1,1,1)       9       8       6       9       8       6  traces/yi2.trace
     3 (4,2,4)       4       5       2       4       5       2  traces/yi.trace
     3 (2,1,4)       2       3       1       2       3       1  traces/dave.trace
     3 (2,1,3)     167      71      67     167      71      67  traces/trans.trace
     3 (2,2,3)     201      37      29     201      37      29  traces/trans.trace
     3 (2,4,3)     212      26      10     212      26      10  traces/trans.trace
     3 (5,1,5)     231       7       0     231       7       0  traces/trans.trace
     6 (5,1,5)  265189   21775   21743  265189   21775   21743  traces/long.trace
    27
```

모든 조건을 충족하였다.

B.

이 문제에서 cache는 s=5, E=1, b=5인 직접매핑 cache이다. 따라서 cache 크기는 2^5*1*2^5=2^10이다.

1) 32x32 행렬

이 경우 행렬의 크기가 cache 크기보다 크다. 하지만 어느정도 비슷하므로 eviction이 많이 생기지 않을 것이다. 한 block당 2^5byte 즉 8개의 int를 담을 수 있으므로, 한번에 8개씩 받아와서 (8x8행렬로 나눠서)전치시키면 좋은 지역성을 얻을 수 있다.

```
Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          27.0        27
Trans perf 32x32           8.0         8         287
Trans perf 64x64           0.0         8     invalid
Trans perf 61x67           0.0        10     invalid
          Total points    35.0        53
```

만점조건을 충족하였다.

2)64x64행렬

이 행렬은 cache보다 크기가 크므로 행렬이 돌다보면 어느순간 이 전에 값이 들어가있던 set를 접근 할 수 밖에 없다. 이 경우 eviction을 무시할 수 없다. 이런 경우 miss가 많이 생길 수 밖에 없는데, miss rate를 줄이기 위해서는 다음 값이 같은 set값을 참조하기 전에 최대한 많은 값을 block에서 참조해서 hit 개수를 증가시키는 것이다.

그리고 위 경우와 마찬가지로 한 block 당 8개의 int값을 넣을 수 있으므로 8x8행렬로 나눠서 전치를 진행한다.

64x64행렬은 4개의 행이 넘어가면 64*4*4=2^10byte만큼 주소가 넘어간다. 즉 모든 cache의 다 사용하므로 eviction 생긴다는 것이다. 따라서 A행렬과 B행렬을 다룰 때 최대한 연속된 4개의 행에서 값을 다뤄주어야 한다.

먼저 A의 윗부분값들을(0-3행) B로 옮겨준다. A의 k(0<=k<4)번째 행의 0-3번째 값들은 B의 k번째 열의 0-3째 값에 넣어준다.그리고 A의 k(0<=k<4)번째 행의 4-7번째 값들은 B의 k+4번째 열의 0-3째 값에 넣어준다. 이 상태에서는 B가 A의 전치행렬을 만족한다고 볼 수 없다. 이 과정을 거치는 이유는 위에서 말했듯 최대한 4개의 행에서 값을 다뤄줘서 eviction을 생기지 않게 하기 위함이다.

```
for(c=0;c<4;c++){
    n1=A[a+c][b];
    n2=A[a+c][b+1];
    n3=A[a+c][b+2];
    n4=A[a+c][b+3];
    n5=A[a+c][b+4];
    n6=A[a+c][b+5];
    n7=A[a+c][b+6];
    n8=A[a+c][b+7];

    B[b][a+c]=n1;
    B[b+1][a+c]=n2;
    B[b+2][a+c]=n3;
    B[b+3][a+c]=n4;
    B[b][a+c+4]=n5;
    B[b+1][a+c+4]=n6;
    B[b+2][a+c+4]=n7;
    B[b+3][a+c+4]=n8;
}
```

이제 A의 윗 4행값을 B로 옮겼으므로 다시 A의 밑 4행을 B로 옮겨주어야한다. 이때 전치행렬에 맞게 B에 있는 값들을 옮겨줄 것이다. B의 k(0<=k<4)번째 행의 4-7번째 값들은 B의 k+4번째 행의 0-3째 값에 넣어준다.(전치행렬을 맞춰준다) 그리고 A의 k(0<=k<4)번째 열의 4-7번째 값들은 B의 k번째 열의 4-7째 값에 넣어준다.

```
for(c=0;c<4;c++){
    n1=B[b+c][a+4];
    n2=B[b+c][a+5];
    n3=B[b+c][a+6];
    n4=B[b+c][a+7];

    n5=A[a+4][b+c];
    n6=A[a+5][b+c];
    n7=A[a+6][b+c];
    n8=A[a+7][b+c];

    B[b+c][a+4]=n5;
    B[b+c][a+5]=n6;
    B[b+c][a+6]=n7;
    B[b+c][a+7]=n8;

    B[b+c+4][a]=n1;
    B[b+c+4][a+1]=n2;
    B[b+c+4][a+2]=n3;
    B[b+c+4][a+3]=n4;
```

마지막으로 남은 1/4값을 전치행렬에 맞게 값을 넣어준다. 이러면 A와 B모두 최대한 연속된 4개의 행에서 값을 다뤘으므로 miss rate가 크게 줄 것이다.

```
for(c=0;c<4;c++){
    n1=A[a+c+4][b+4];
    n2=A[a+c+4][b+5];
    n3=A[a+c+4][b+6];
    n4=A[a+c+4][b+7];

    B[b+4][a+c+4]=n1;
    B[b+5][a+c+4]=n2;
    B[b+6][a+c+4]=n3;
    B[b+7][a+c+4]=n4;
```

```
Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          27.0        27
Trans perf 32x32           8.0         8         287
Trans perf 64x64           8.0         8        1227
Trans perf 61x67           0.0        10     invalid
        Total points      43.0        53
```

miss값이 만점 조건을 충족했다.

3)61x67행렬

이 경우는 만점 기준이 낮다고 생각해서 32x32행렬처럼 8x8행렬로 나눠서 전치행렬을 만들었다. 정사각행렬이 아니므로 56x64 행렬을 제외하고 나머지 부분을 예외처리해주면 만점기준을 통과한다. 이 문제는 8x8로 나눠서 지역성을 좋게 해주는 것만으로도 통과가 가능했다.

```
Cache Lab summary:
                        Points   Max pts      Misses
Csim correctness          27.0        27
Trans perf 32x32           8.0         8         287
Trans perf 64x64           8.0         8        1227
Trans perf 61x67          10.0        10        1993
        Total points      53.0        53
```