Computer Organization

Assignment 3

2018-14245 김익환

June 04, 2020

1. What's the impact of row-by-row or column-by-column accesses? Given the cache hit ratio you observed, can you interpret those numbers? You may mention 1) the characteristics of matrix multiplication algorithms, 2) how each cache policy in Task1 and Task2 impacts the cache hit ratio, etc.

```
1  // Matrix multiplication: C = A * B
2  for (i = 0; i < MAT_ORDER; i++) {
3      for (j = 0; j < MAT_ORDER; j++) {
4          for (k = 0; k < MAT_ORDER; k++) {
5  #ifdef ROW_MAJOR
6                  sum = sum + A[i][k]*B[k][j];
7  #else
8                  sum = sum + A[j][k]*B[k][i];
9  #endif
10         }
11         MAT(C, i, j) = sum;
12         sum = 0;
13     }
14 }
```

Listing 1: Matrix multiplication

Listing 1은 `ROW_MAJOR`일 때와 아닐 때 matrix multiplication의 차이를 보여준다.

```
1  ./emu-rv32i-dir-cache ./mat-col2
2  >>> cache hit ratio: 0.9832 (hit:1459, miss:25)
3  ./emu-rv32i-set-cache ./mat-col2
4  >>> cache hit ratio: 0.9832 (hit:1459, miss:25)
5  ./emu-rv32i-dir-cache ./mat-row2
6  >>> cache hit ratio: 0.9832 (hit:1459, miss:25)
7  ./emu-rv32i-set-cache ./mat-row2
8  >>> cache hit ratio: 0.9832 (hit:1459, miss:25)
```

```
9
10 ./emu-rv32i-dir-cache ./mat-col4
11 >>> cache hit ratio: 0.9835 (hit:7502, miss:126)
12 ./emu-rv32i-set-cache ./mat-col4
13 >>> cache hit ratio: 0.9920 (hit:7567, miss:61)
14 ./emu-rv32i-dir-cache ./mat-row4
15 >>> cache hit ratio: 0.9835 (hit:7502, miss:126)
16 ./emu-rv32i-set-cache ./mat-row4
17 >>> cache hit ratio: 0.9919 (hit:7566, miss:62)
18
19 ./emu-rv32i-dir-cache ./mat-col8
20 >>> cache hit ratio: 0.9839 (hit:45469, miss:743)
21 ./emu-rv32i-set-cache ./mat-col8
22 >>> cache hit ratio: 0.9795 (hit:45266, miss:946)
23 ./emu-rv32i-dir-cache ./mat-row8
24 >>> cache hit ratio: 0.9848 (hit:45508, miss:704)
25 ./emu-rv32i-set-cache ./mat-row8
26 >>> cache hit ratio: 0.9819 (hit:45376, miss:836)
27
28 ./emu-rv32i-dir-cache ./mat-col16
29 >>> cache hit ratio: 0.9666 (hit:283861, miss:9813)
30 ./emu-rv32i-set-cache ./mat-col16
31 >>> cache hit ratio: 0.9686 (hit:284441, miss:9233)
32 ./emu-rv32i-dir-cache ./mat-row16
33 >>> cache hit ratio: 0.9774 (hit:287049, miss:6625)
34 ./emu-rv32i-set-cache ./mat-row16
35 >>> cache hit ratio: 0.9802 (hit:287865, miss:5809)
```

Listing 2: Difference by row and column

Listing 2를 보면 같은 `MAT_ORDER` 내에서 column access보다 row access가 cache hit ratio가 더 높은 것을 알 수 있다. 행렬 크기가 2와 4일 때는 별 차이가 없었지만 크기가 8과 16일 때 row access가 direct-mapped cache와 set associative cache 모두에서 cache hit ratio가 더 높았다.

1) row major는 행렬을 계산할 때 한 행을 먼저 계산하고 다음 행으로 넘어간다. Column major는 반대로 한 열을 먼저 계산하고 다음 열로 넘어간다. 행렬이 메모리에 저장될 때 한 행이 저장되고 다음 행이 저장되므로 row-by-row access가 spatial locality가 더 높다고 볼 수 있다. 따라서 row-by-row access가 메모리를 연속적으로 접근하므로 collision이 적어 cache hit ratio가 더 높다. Column-by-column access는 반대로 행렬 크기에 따라 불연속적으로 메모리를 접근하는데, 만약 행렬 크기가 충분히 커서 불연속적으로 접근하는 크기가 index를 넘어가면 tag가 달라져 cache hit ratio가 낮아

진다. Listing 2는 행렬 크기가 클 때 row major와 column major간의 cache hit ratio 차이가 커지므로 이러한 현상을 잘 보여준다고 할 수 있다.

2) 행렬 크기가 2와 4일 때는 두 cache policy의 차이가 크지 않지만, 행렬 크기가 8과 16일 때 cache hit ratio의 차이가 크다. 행렬 크기가 8일 때 set associative cache의 cache hit ratio가 더 낮은데, 행렬 크기가 8일 때 행렬 전체의 32 bit integer 수가 2^6 개이므로 cache access granularity가 2 bit일 때 총 6 bit의 index가 필요하다. Direct-mapped cache의 경우 index가 6 bit이므로 이를 만족하지만 set associative cache의 경우 index가 4 bit이므로 cache size가 working set size보다 작아 cache hit ratio가 낮다. 행렬 크기가 16일 때는 필요한 index가 8 bit이므로 두 cache policy 모두 collision이 발생한다. 이러한 상황에서는 set associative cache의 cache hit ratio가 더 높았다. Direct-mapped cache는 2^t개의 tag가 1개의 bank로 매핑되지만 set associative cache는 4개의 bank로 매핑되기 때문이다.

2. What's the impact of matrix order (i.e., `MAT_ORDER`) to the cache hit ratio of direct-mapped cache and 4-way set-associative cache, respectively? Do you have any interesting findings that you want to share? You may try to relate your answer with the concept of working set.

```
./emu-rv32i-dir-cache ./mat-col2
>>> cache hit ratio: 0.9832 (hit:1459, miss:25)
./emu-rv32i-dir-cache ./mat-col4
>>> cache hit ratio: 0.9835 (hit:7502, miss:126)
./emu-rv32i-dir-cache ./mat-col8
>>> cache hit ratio: 0.9839 (hit:45469, miss:743)
./emu-rv32i-dir-cache ./mat-col16
>>> cache hit ratio: 0.9666 (hit:283861, miss:9813)

./emu-rv32i-dir-cache ./mat-row2
>>> cache hit ratio: 0.9832 (hit:1459, miss:25)
./emu-rv32i-dir-cache ./mat-row4
>>> cache hit ratio: 0.9835 (hit:7502, miss:126)
./emu-rv32i-dir-cache ./mat-row8
>>> cache hit ratio: 0.9848 (hit:45508, miss:704)
./emu-rv32i-dir-cache ./mat-row16
>>> cache hit ratio: 0.9774 (hit:287049, miss:6625)

./emu-rv32i-set-cache ./mat-col2
>>> cache hit ratio: 0.9832 (hit:1459, miss:25)
./emu-rv32i-set-cache ./mat-col4
>>> cache hit ratio: 0.9920 (hit:7567, miss:61)
./emu-rv32i-set-cache ./mat-col8
>>> cache hit ratio: 0.9795 (hit:45266, miss:946)
```

```
./emu-rv32i-set-cache ./mat-col16
>>> cache hit ratio: 0.9686 (hit:284441, miss:9233)

./emu-rv32i-set-cache ./mat-row2
>>> cache hit ratio: 0.9832 (hit:1459, miss:25)
./emu-rv32i-set-cache ./mat-row4
>>> cache hit ratio: 0.9919 (hit:7566, miss:62)
./emu-rv32i-set-cache ./mat-row8
>>> cache hit ratio: 0.9819 (hit:45376, miss:836)
./emu-rv32i-set-cache ./mat-row16
>>> cache hit ratio: 0.9802 (hit:287865, miss:5809)
```

Listing 3: Difference by matrix order

Listing 3을 보면 direct-mapped cache의 경우 행렬 크기가 8일 때까지 cache hit ratio가 증가하지만 크기가 16일 때 cache hit ratio가 급격히 감소한다. 그 이유는 행렬 크기에 따라 working set size가 달라지기 때문이다. 행렬 크기가 8일 때 1에서 언급한 것과 같이 필요한 index는 6 bit이다. Direct-mapped cache의 index가 6 bit이므로 direct-mapped cache는 행렬 크기가 8이하일 때까지 $C \geq$ working set size를 만족한다. 하지만 행렬 크기가 16일 때는 $C <$ working set size가 되므로 collision이 발생하여 cache hit ratio가 급격히 감소한다.

Set associative cache의 경우 행렬 크기가 4일 때까지 cache hit ratio가 증가했다가 행렬 크기가 8 이상일 때부터 감소한다. 그 이유는 위와 같은데, 행렬 크기가 4일 때 working set size가 $2^4 \times 4$ bytes여서 필요한 index가 4 bit이다. Set associative cache의 index가 4 bit이므로 행렬 크기가 4일 때까지 $C \geq$ working set size를 만족한다. 하지만 행렬 크기가 8 이상일 때부터 $C <$ working set size가 되어 collision이 발생한다. 행렬 크기가 16일 때의 결과를 보면 두 cache policy 모두 $C <$ working set size일 때 set associative cache가 더 성능이 좋은 것을 알 수 있다. 이는 1에서 언급하였듯이 set associative cache가 tag에 매핑되는 bank의 수가 더 많기 때문이다.

3. Any thoughts on how to further improve the cache performance? How should you change the cache design?

   Cache performance를 향상시키는 가장 좋은 방안은 cache의 size를 늘리는 것이다. 당연히 $C \geq$ working set size를 만족하는 범위가 넓어지므로 collision이 적어 cache hit ratio가 높아질 것이다. 다른 방법으로는 set associative cache를 사용할 때 더 좋은 replacement policy를 사용하는 것이 있다. 이번 과제에서 사용한 policy는 LRU인데, LFU, LFRU, LFUDA, LIRS 등의 알고리즘이 있다. 수행하는 연산에 적합한 알고리즘을 택하면 cache performance를 향상시킬 수 있을 것이다.