ECE #563

Level – 1 Cache

Zhiping Wang | Unity ID: zwang79 | Student ID: 200265085

05.03.2019

Objective

The goal of this project is to design and implement a cache simulator (level-1 cache only). Also, generate several memory access traces for matrix multiplication code for cache behavior analysis.

Result

All testcases are matched. 4 memory access traces, for N = 4, 10, 20 and 100, have been generated. This report is focused on trace generation and cache behavior analysis regarding matrix multiplication.

Main Aspects

Cache Design:

Original files have been improved to meet cache requirement: cache.cc, cache.h.

Trace Generation:

The multiplication method I have follow is:

```
for (int i=0;i<N;i++){
   for (int j=0;j<N;j++){
      tmp=0;
      for (int k=0;k<N;k++){
         tmp+=a[i][k]*b[k][j];
      }
       c[i][j]= tmp;
   }
}
```

The memory operation is generated by a 3-layed iteration.

```
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++) {
            for (int k = 0; k < N; k++) {
                cout << "r" << setfill(' ') << setw(3) << "0x" << setiosflags(ios::uppercase) << hex << address_1 + 4 * (i * N + k) << endl;
                cout << "r" << setfill(' ') << setw(3) << "0x" << setiosflags(ios::uppercase) << hex << address_2 + 4 * (k * N + j) << endl;
                cout << "r" << setfill(' ') << setw(3) << "0x" << setiosflags(ios::uppercase) << hex << address_4 + 4 * (i * N + j) << endl;
                cout << "w" << setfill(' ') << setw(3) << "0x" << setiosflags(ios::uppercase) << hex << address_4 + 4 * (i * N + j) << endl;
            }
            cout << "w" << setfill(' ') << setw(3) << "0x" << setiosflags(ios::uppercase) << hex << address_3 + 4 * (i * N + j) << endl;
        }

    }
}
```

I choose four values: 4, 10, 20, 100, for matrix size N, in order to compare different cache behavior whether eviction takes place or not.

4 trace files and testcases are created:

trace_N4.t   trace_N10.t   trace_N20.t   trace_N100.t

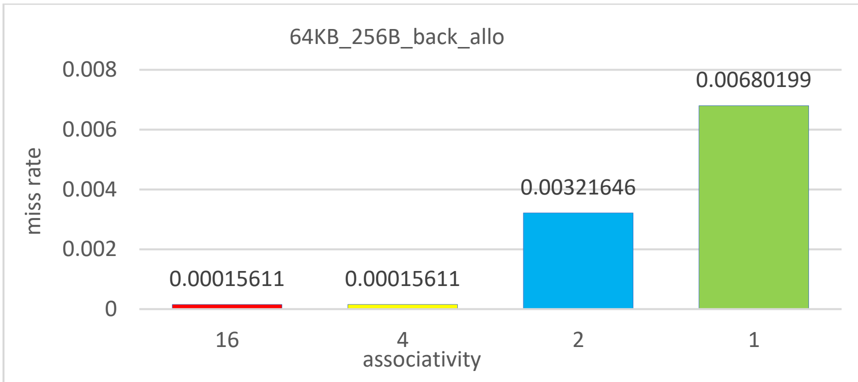
testcase_N4.cc   testcase_N10.cc   testcase_N20.cc   testcase_N100.cc

I also included an Excel document which record all my analysis data as back-up.

Cache Behavior Analysis (write-back/write-allocate)

1. Influence of Associativity

As we are supposed to use 16-way cache to approximate the behavior of a fully associative cache, this part of analysis is not important. I take 64KB cache size, 256B block size cache as an example with N = 100.



Eviction takes place for this kind of cache:

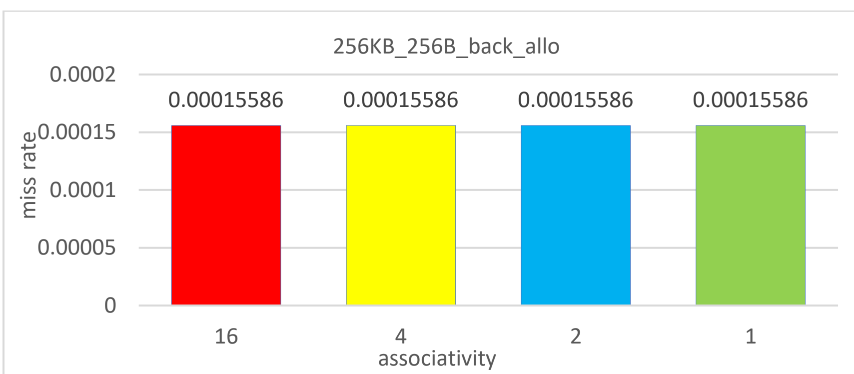
associativity = 16, miss rate = 0.00015611, number of evictions = 370;

associativity = 4 , miss rate = 0.00015611, number of evictions = 370;

associativity = 2 , miss rate = 0.00321646, number of evictions = 12642;

associativity = 1 , miss rate = 0.00680199, number of evictions = 27020;

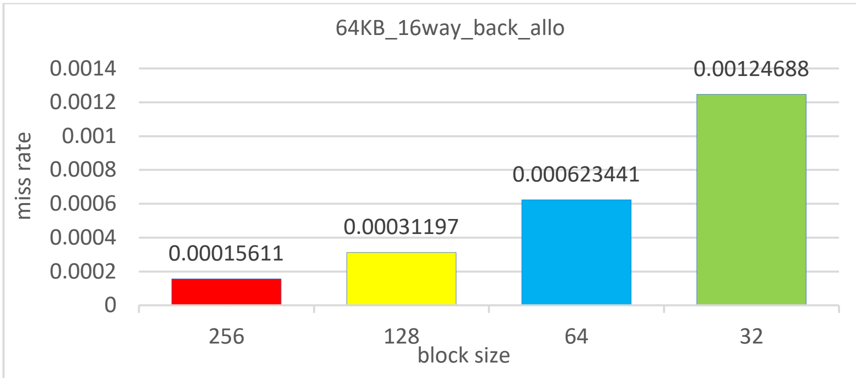
Also, if I change the cache size to 256KB, miss rate is not affected by associativity.



CONCLUTION: as shown that the miss rate decreases as associativity goes up. The reduction is rapid when associativity changes from 1 to 2, and it becomes less obvious when associativity goes larger.

2. Influence of Block Size

For a 64KB cache size, 16-way cache with N = 100:



Eviction also takes place:

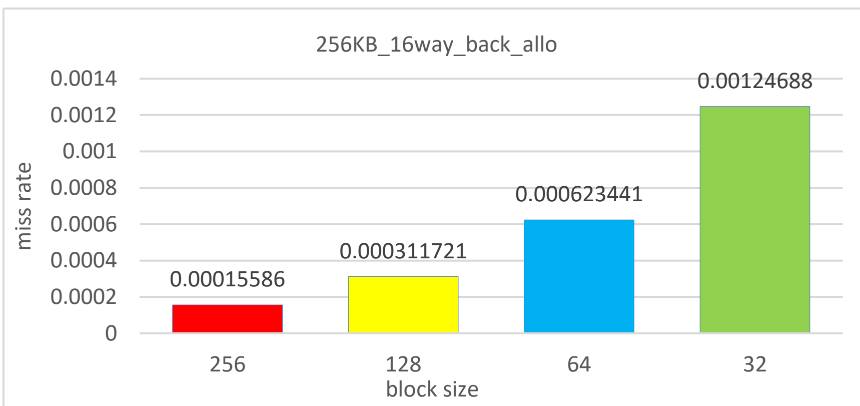
block size = 256B, miss rate = 0.00015611 , number of evictions = 370;

block size = 128B, miss rate = 0.00031197 , number of evictions = 739;

block size = 64B , miss rate = 0.000623441, number of evictions = 1476;

block size = 32B , miss rate = 0.00124688 , number of evictions = 2952;

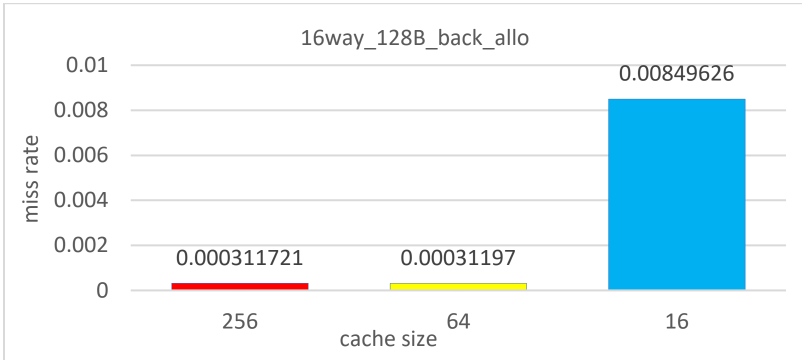
For a 256KB cache size, 16B block size cache:



CONCLUTION: miss rate decreases as block size goes up.

3. Influence of Cache Size

For a 16-way, 128B block size cache with N = 100:

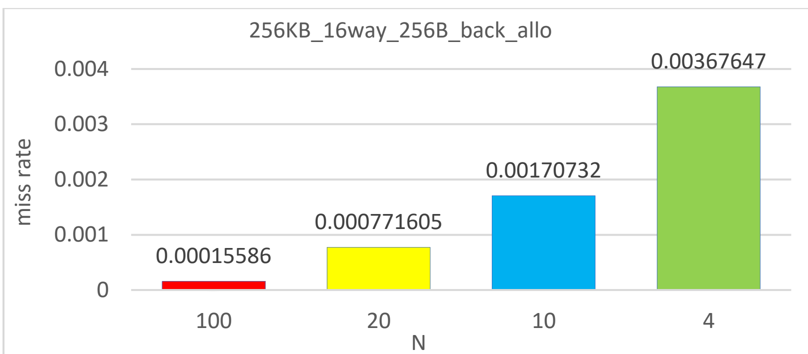


Eviction happens when cache size is less than 256KB.

CONCLUTION: miss rate decreases as cache size goes up.

4. Influence of Cache Size

For a 256KB cache size, 16-way, 128B block size cache:



CONCLUTION: miss rate decreases as N goes up. I believe the main reason for this is the number of memory access increases as N goes up. The number of misses, however, does not change much.

Summary

The lowest miss rate overall is when cache size = 256KB, associativity = 16, block size = 256B.