Cache Simulator

CS 3853 Computer Architecture

4th May, 2019

Presented by:

Daniel Garcia

Seth Greco

Vishalkumar Patel

from project import index
print(index)

Particulars **Page #**

Implementation

For our implementation of the cache simulator we used python as our language of choice. The class “Cache.py” contains all the data we need to keep track whether the block we want is in the cache or if its invalid and only in memory. This class also stores data that allows us to see when was the last time it was used relative to all the other blocks currently in the cache memory. When an invalid bit is seen it quickly looks over which block of memory has the most time not used and replaces it with the block we want to access next. When we want to access something already in the cache we simply update its time and position so it becomes the most recently used. Inside this class we also import functions for a write back policy and basic stats suchs as hit rate, and the miss rate which is essentially the percentage of page faults we had.

The main class “Simulator.py” just checks the parameters for cache size and source files. Line by line it executes the correct “Cache.py” functions and fills “Cache.py” objects concluding with the function that prints the desired performance metrics.

We specifically used Python version 3.7 for our Cache-Simulator project.

Experiments

The Cache-Simulator has been tested with various test inputs. Some of the inputs tested were from the class powerpoints to see if the Cache-Simulator produces the expected outputs with similar counts. The test concuted are as follows:

```
$ python3 Simulator.py -s 16KB -a 32 -f ls.trace.txt
  Total Lines: 254652
  Hits       : 248037
  Miss       : 6615
  Miss Rate  : 2.60%

$ python3 Simulator.py -s 32KB -a 32 -f ls.trace.txt
  Total Lines: 254652
  Hits       : 249149
  Miss       : 5503
  Miss Rate  : 2.16%

$ python3 Simulator.py -s 32KB -a 16 -f ls.trace.txt
  Total Lines: 254652
  Hits       : 249128
  Miss       : 5524
  Miss Rate  : 2.17%

$ python3 Simulator.py -s 32KB -a 8 -f ls.trace.txt
  Total Lines: 254652
  Hits       : 249072
  Miss       : 5580
  Miss Rate  : 2.19%
```

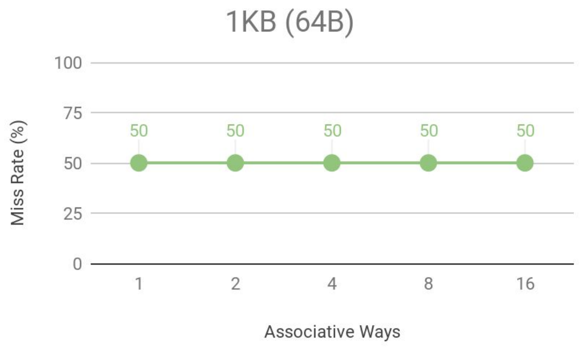
More testing was carried out on the Cahe-Simulator with test inputs provided in the Project folder to see if it calculated the same miss rates as expected. It may have the last precision rounded by 0.01 but actual miss rates produced aligned the expected outputs.

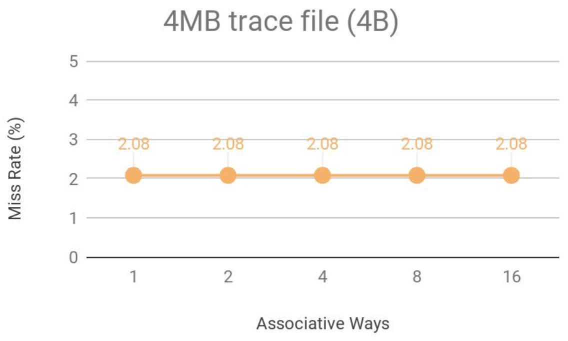
Following are the 16 way associative cache miss rates for various trace files :

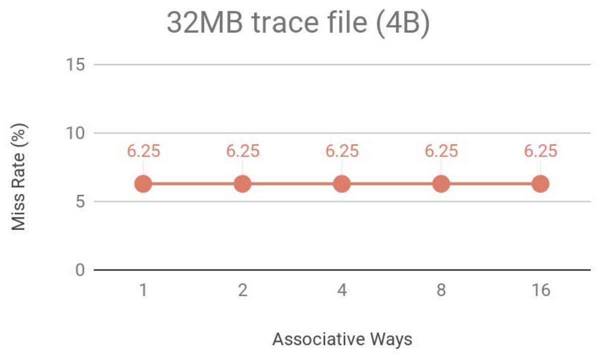
1KB_64B	Cache Size: 1KB	Ways: 16	Miss Rate: **50.00%**
4MB_4	Cache Size: 4MB	Ways: 16	Miss Rate: **2.08%**
32MB_4B	Cache Size: 4MB	Ways: 16	Miss Rate: **6.25%**
bw_mem.traces.txt	Cache Size: 4MB	Ways: 16	Miss Rate: **1.56%**
ls.trace.txt	Cache Size: 32KB	Ways: 16	Miss Rate: **2.17%**
gcc.trace.txt	Cache Size: 32KB	Ways: 16	Miss Rate: **1.90%**
naive_dgemm.trace.txt	Cache Size: 256KB	Ways: 16	Miss Rate: **50.25%**
naive_dgemm_full.trace.txt	Cache Size: 256KB	Ways: 16	Miss Rate: **49.37%**
openblas_dgemm.trace.txt	Cache Size: 256KB	Ways: 16	Miss Rate: **8.30%**
full_dgemm.traces.txt	Cache Size: 256KB	Ways: 16	Miss Rate: **7.50%**

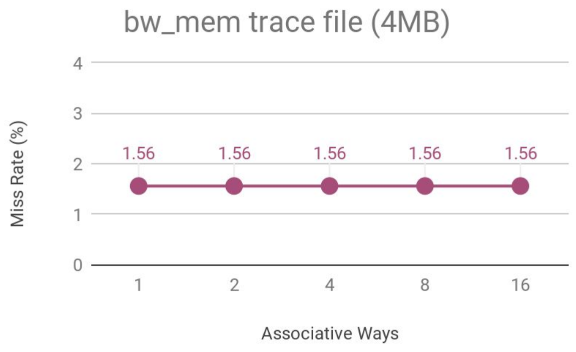
Experimentation

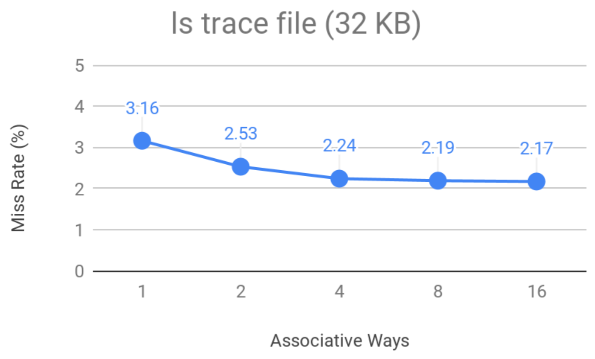
Below are the scaled graph for various trace files both small and large ones with similar cache size but different associativity ways. On the top of each graph says the file that it was tested with along with its cache-size. On the Y-axis is the Miss rates with Associativity on the X-axis; which gives the plotted graph as below.

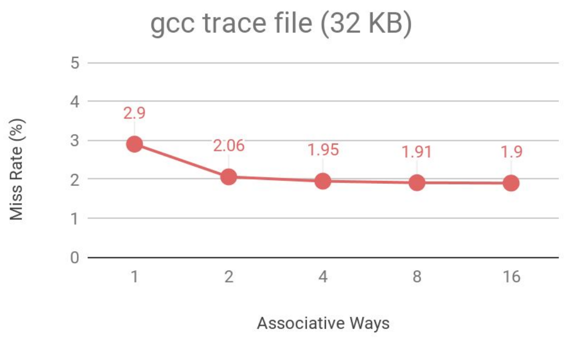

1KB (64B)
Miss Rate (%)
Associative Ways
100
75
50
25
0
1
2
4
8
16
50
50
50
50
50

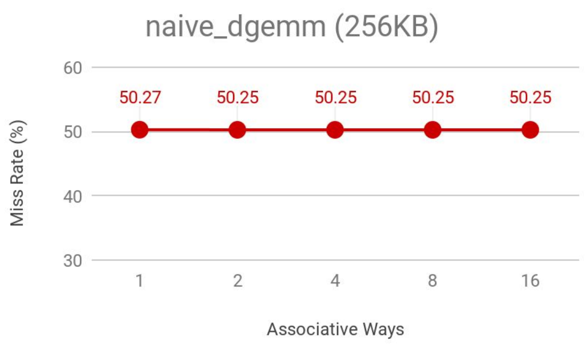

4MB trace file (4B)
Miss Rate (%)
Associative Ways
5
4
3
2
1
0
1
2
4
8
16
2.08
2.08
2.08
2.08
2.08

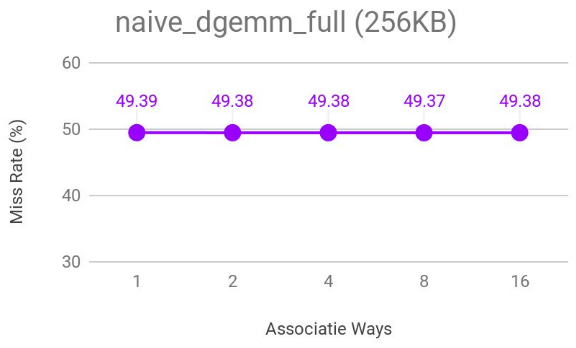

32MB trace file (4B)
Miss Rate (%)
Associative Ways
15
10
5
0
1
2
4
8
16
6.25
6.25
6.25
6.25
6.25

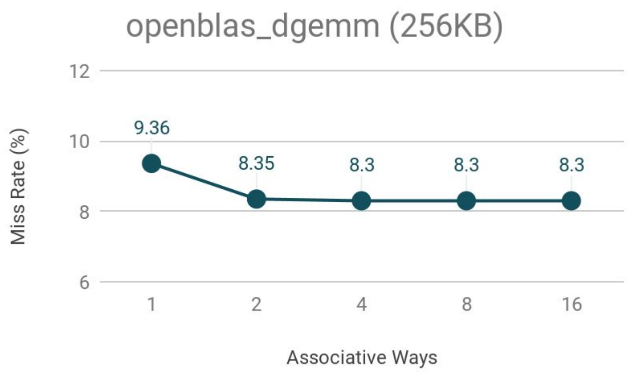

bw_mem trace file (4MB)
Miss Rate (%)
Associative Ways
4
3
2
1
0
1
2
4
8
16
1.56
1.56
1.56
1.56
1.56


ls trace file (32 KB)
Miss Rate (%)
Associative Ways
5
4
3
2
1
0
1
2
4
8
16
3.16
2.53
2.24
2.19
2.17


gcc trace file (32 KB)
Miss Rate (%)
Associative Ways
5
4
3
2
1
0
1
2
4
8
16
2.9
2.06
1.95
1.91
1.9







Observation

We observed several features when testing different test files with our Cache-Simulator. From the experiments conducted and the graphs above, we learned that when increasing the associativity ways for the trace files, we saw decrease in the miss rates with fewer conflicts but when making the associativity smaller we saw increase in miss rates with higher conflicts.

Smaller cache size has limited temporal locality and is not adventegous as useful data will have to be continously replaced. While if it is bigger it can exploit temporal locality better. We can clearly see how access time mau hurt critical path.

Contribution

Everyone had a fair share in the contributing towards the Cache-Simulator project. But the major areas where the team members focused are as follows. We regularly met to see that we were on the same page by explaining each other about whats and hows of the project.

Vishalkumar Patel	: LRU Policy
Daniel Garcia	: LRU Policy
Seth Greco	: Write Back Policy