Hayden Miedema and Douglas Money

CIS 451

Professor Kurmas

Lab 11: Cache (part 2)

1. Write a simple C program for which a 4-way associative cache has a significantly lower miss rate than a equally sized 2-way set associative cache. You may choose any cache size and block size you wish, but they must remain the same for the entire problem. Submit the source code, all cache parameters, and resulting hit rates. Submit the source code, all cache parameters, and resulting hit rates. Hint: Write the simplest program you can that will produce a 100% miss rate for the 2-way cache.

Command to run:

2-way: ~kurmasz/public/Simplescalar/bin/sim-cache -cache:dl1 dl1:32:8:**2**:l -redir:prog /dev/null -redir:sim num6out num6

4-way: ~kurmasz/public/Simplescalar/bin/sim-cache -cache:dl1 dl1:32:8:**4**:l -redir:prog /dev/null -redir:sim num6out4 num6

4-way

dl1.miss\_rate 0.0004 # miss rate (i.e., misses/ref)

  2-way

dl1.miss\_rate 0.7497 # miss rate (i.e., misses/ref)

int main()

{

char array[ARRAY\_SIZE];

register int outer\_loop;

register int inner\_loop;

register int solution = 0;

for (outer\_loop = 0; outer\_loop < NUM\_LOOPS; outer\_loop++)

{

solution \*= array[0];

solution \*= array[4095];

solution \*= array[8191];

solution \*= array[12287];

}

return solution;

}

1. Write a simple C program for which an associativity of 2 has a *higher* miss rate than a direct-mapped cache. You may choose any cache size and block size you wish, but they must remain the same for the entire problem. Submit the source code, all cache parameters, and resulting hit rates. As with the previous problem, start by writing a program that has a 100% miss rate for a 2-way cache.

Direct mapped miss rate

dl1.miss\_rate 0.6664 # miss rate (i.e., misses/ref)

2-way miss rate

dl1.miss\_rate 0.9993 # miss rate (i.e., misses/ref)

int main()

{

char array[ARRAY\_SIZE];

register int outer\_loop;

register int inner\_loop;

register int solution = 0;

for (outer\_loop = 0; outer\_loop < NUM\_LOOPS; outer\_loop++)

{

solution \*= array[0];

solution \*= array[4096];

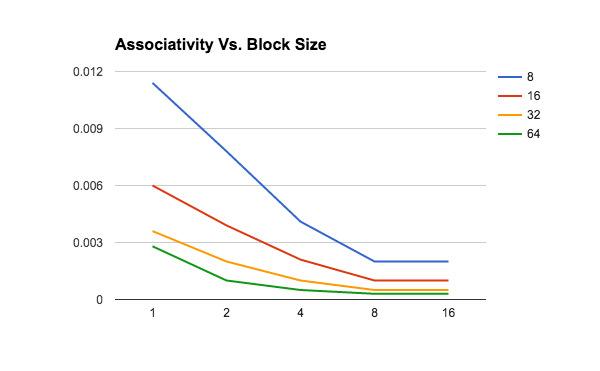
solution \*= array[8192];

}

return solution;

}

1. Choose qsort (or another interesting program of your choice) and a cache size. Produce a graph showing miss rates as associativity ranges over 1, 2, 4, 8, 16, and fully associative. Your graph should have associativity on the x-axis, and miss-rate on the y-axis. It should also contain four lines: one for each block size. Be sure to clearly label your graph with the cache size. Your graph should have a form similar to [Figure 5.30](http://www.cis.gvsu.edu/~kurmasz/Teaching/Courses/F16/CS451/Assignments/Cache/effectsOfAssoc.jpg) in Patterson and Hennessey (4th edition, revised).



1. Write a simple C program for which a random replacement policy results in a significantly lower miss rate than LRU. (Use an 8-way set associative cache with a size and block size of your choosing.) Submit the source code, all cache parameters, and resulting hit rates.

We did not get time to have this fully implemented, but the idea would be to use an N-way cache and keep filling the same spot over and over in the loop and nothing else. This way, the LRU algorithm would have a 100% miss rate because the spot that is being written to every time will never be kicked out. Random replacement would have surely behave better due to it’s randomness selection.

1. Choose a cache size and block size. Plot the effects of associvity and replacement policy on qsort. Your graph should have associvity on the x-axis and miss rate on the y-axis. There should be three lines: one for each replacement policy.

