CS224 – Fall 2017 - Lab #6

Examining the Effect of Cache Parameters and Program Factors on Cache Hit Rate

Dates:

Section 1:	Wed, Dec. 13th	13:40-17:30 in EA-Z04
Section 2:	Fri, Dec. 15th	13:40-17:30 in EA-Z04
Section 3:	Wed, Dec. 13th	08:40-12:30 in EA-Z04
Section 4:	Thu, Dec. 14th	13:40-17:30 in EA-Z04

Purpose: In this lab you will study the effect of various cache design parameters, and also some program factors, on cache hit rate for a particular program. Using the bubble sort program, on an array of random integers, you will tune the cache for maximum hit rate under various conditions. The bubble sort routine (including a call to swap 2 adjacent elements) is given in MIPS assembly code, posted on Unilica. A test program to exercise it is also given; the test program obtains an array of memory locations in heap, fills the array with random integer values, then calls bubble sort to sort them.

Summary

Part 1 (30 points): Preliminary Report/Preliminary Design Report: Using the bubble sort program, on an array of random integers, you will tune the cache for maximum hit rate under various conditions. (Due date of this part is the same for all).

Part 2 (70 points): Simulation of cache parameters and observing effects.

DUE DATE OF PART 1: SAME FOR ALL SECTIONS Dear students please bring and drop your preliminary work into the box provided in front of the lab before 3:59 pm on Tuesday December 12th. No late submission!

LAB WORK SUBMISSION TIMING: You have to show your lab work to your TA by **12:15** in the morning lab and by **17:15** in the afternoon lab. Note that you cannot wait for the last moment to do this. If you wait for the last moment and show your work after the deadline time 20 points will be taken off.

If we suspect that there is cheating we will send the work with the names of the students to the university disciplinary committee.

Part 1. Preliminary Work / Preliminary Design Report (30 points)

a) Go to the Web (Wikipedia, etc) and refresh your knowledge about the bubble sort algorithm.
b) Read and understand the header of the bubble sort, where the C code is given.
c) Study the MIPS assembly implementations of *bubblesort* and *swap* until you understand how they work.
d) Notice that in the .data segment, a constant named 'size' is given. Originally it is set to 1000, but it can be changed. Since bubble sort is $O(n^2)$, changing it will have a big effect on runtimes.
e) Open MARS, assemble and run the program. Notice the first Run I/O message (in the window at the bottom) comes relatively quickly, but the second message takes much longer. Why?
f) Reset the program, and run it again, but this time use your watch or phone (or the Windows "Date and Time" function) to time how many seconds it takes for
   i) the first message to appear
   ii) for the second message to appear.
   Practice until you are recording the time with the best accuracy you can.

   Program Analysis (or "Using Big-O in practice") From the given MIPS program, it is clear that the time for the first message to appear is proportional to m0 + m1*n, since some tasks are done just once, and some (filling the locations in the array) are done in a loop n times, where n is the size of the array. Also from the program, it is clear that the time for the second message to appear is proportional to k0 + k1*n + k2*(n^2), since this time includes the sorting, which is $O(n^2)$. By subtracting Time(2nd message) - Time(1st message), you can get reasonably close to k2(n^2). If the run times are long enough, k0 will be trivial, dominated by k1*n and k2*(n^2). So we will make the following simplifications:

   T(fill) = k1*n = Time(1st message)

   T(sort) = k2*(n^2) = Time(2nd message) - Time(1st message)
g) Now, make a series of tests, with 'size' set to different values, and record the values in the table

n (= array size)	Fill time (in seconds)	Sort time (in seconds)
500		
1000		
2000		
4000		

h) Calculate the following, approximately:
   k1, the linear constant, = ?
   k2, the quadratic constant, = ?

[Hint: if the T(fill) times you tried to measure were not distinguishable enough to get an estimate of k1, then do runs with one or two much much larger values of size, but don't run to completion (since the $O(n^2)$ phase will be TOO long). Instead, pause, then reset, the run after seeing the first message .]

i) How could you get more accurate measurements? (Hint: see the table of syscalls offered in MARS.) How would you use this syscall's return value? What changes would you need to make to your program in order to benefit from this method of getting the time?

j) Now set the 'size' parameter in the program to a small enough number so that the run times will be reasonable. You will do many runs of bubble sort, so make sure each run doesn't last too long. Open the Data Cache Simulation Tool, accept the default values, and hit "Connect to MIPS". Now whenever a MIPS program is run in MARS, the data cache simulator tool will run also (interrupting each time a data access (lw or sw) is made in program execution. Now re-run the program again, and time how long it takes for the 1st and 2nd messages to appear. If it runs too slowly, reduce the size of the array until you are satisfied with the run time. How much of a slow-down effect did adding this tool have? (Remember that the cache simulation tool, and all of MARS itself, are written in Java, simulating MIPS using JVM, but actually running in Windows on Intel/AMD processor hardware. That's a lot of translations-- no wonder things are slow!)

Part 2: Implementation and Testing

Data Cache Parameters

a) Direct Mapped Caches: For the array size you have chosen, conduct tests with various cache sizes and block sizes, to determine the hit rate, miss rate and number of misses. Use at least 5 different cache sizes and 5 different block sizes (make sure your values are reasonable) in order to obtain curves like those of Figure 8.18 in the textbook. Make a 5 x 5 table with your values, with miss rate and # of misses as the data at each row-column location. Make a graph of miss rate versus block size, parameterized by cache size, like Figure 8.18

b) Fully Associative Caches: Pick 3 of your parameter points obtained in part a), one with good hit rate, one with medium hit rate, and one with poor hit rate. For these 3 results, there were 3 configuration pairs of cache size and block size that resulted in the data. Take the same 3 configuration pairs, but this time run the simulation with a fully associate cache, using LRU replacement policy. Compare the results obtained: the Direct Mapped good result versus the Fully Associative good result, the Direct Mapped medium result versus the Fully Associative medium result, and the Direct Mapped poor result versus the Fully Associative poor result. How much difference did the change to fully associative architecture make? Now change the replacement policy to Random replacement, and run the 3 tests again (using the same 3 configuration pairs). Does replacement policy make a significant difference? Record these 9 values in a new table, with 3 lines: for Direct Mapped, for Fully Associative-LRU and for Fully Associative-Random.

c) N-way Set Associative Caches: to save on hardware costs, fully set-associative caches are rarely used. Instead, most of the benefit can be obtained with an N-way set associative cache. Pick the medium hit rate configuration that you found in a) and used again in b), and change the architecture to N-way set associative. For several different set sizes (at least 4) and LRU replacement policy, run the program and record the hit rate, miss rate and number of misses. What set size gives the best result? How much improvement is gained as N (the number of blocks in a set) increases each step? Now repeat the tests, but for the good hit rate configuration from a) and b). Record these data and answer the same question again. Finally, repeat for the poor hit rate configuration.

Program Factors

Besides the cache configuration, the other thing that affects the hit rate/miss rate is the program's locality of reference. For data references, the program you have been given has a few places where loops move through the array. If the array is larger or smaller, a different percentage of it will be contained in cache, possibly affecting the hit/miss rate. Run a few tests and record the results, changing only array size (run it once with larger array size and once with smaller array size) for several of your previously measured data points. In other words, keep all other factors the same, and change only the size of the array of data. What is the effect of changing the data size on the hit rate/miss rate? Record these results in a table as well, with old values next to new values.

Oral Quiz with TA and Explanation of your Data (70%)

Get ready for the interview with your TA, by gathering and analyzing all your data, having it ready to submit in a clean understandable format. Be sure that you understand what you have done, and can interpret your data to the TA. Then call him over, and answer the questions he asks you, and turn in your data to the TA. (It should have your name, ID, "Lab 6", etc written at the top).

Part 3: Submit Your Lab6 Document

Submit your Lab6 document to the Unilica > Assignment specific for your section. You will upload one file: Yourname_Yoursurname_Lab6 from the results you created in Parts 1 and 2. All students must upload their code to Unilica > Assignment while the TA watches. Submissions made without the TA observing will be deleted, resulting in a lab score of 0.

Part 4. Cleanup

1) After saving any files that you might want to have in the future to your own storage device, erase all the files you created from the computer in the lab.
2) When applicable put back all the hardware, boards, wires, tools, etc where they came from.
3) Clean up your lab desk, to leave it completely clean and ready for the next group who will come.

LAB POLICIES

1. You can do the lab only in your section. Missing your section time and doing in another day is not allowed.
2. Students will earn their own individual lab grade. The questions asked by the TA will have an effect on your individual lab score.
3. Lab score will be reduced to 0 if the code is not submitted for similarity testing, or if it is plagiarized. MOSS-testing will be done, to determine similarity rates. Trivial changes to code will not hide plagiarism from MOSS—the algorithm is quite sophisticated and powerful. Please also note that obviously you should not use any program available on the web, or in a book, etc. since MOSS will find it. The use of the ideas we discussed in the classroom is not a problem.
4. You must be in lab, working on the lab, from the time lab starts until your work is finished and you leave.
5. No cell phone usage during lab.
6. Internet usage is permitted only to lab-related technical sites.
7. For labs that involve hardware for design you will always use the same board provided to you by the lab engineer.