CS224 – Fall 2020 – Lab #6

Examining the Effect of Cache Parameters and Program Factors on Cache Hit Rate (Version2 December 26, 9:52 am)

Dates:
Section 1: Wednesday, 23 December, 13:30-16:20
Section 2: Friday, 25 December, 13:30-16:20
Section 3: Sunday, 27 December, 8:30-11:20
Section 4: Thusday, 24 December, 13:30-16:20

Submission Date: December 28, Monday, 23:59, for all lab days.
(No preliminary work and lab work distinction.
See the submission instructions below)
Physical Lab Administration: No physical lab: All labs are online. Your TAs will be available during your lab hours. Other times you may reach them by email.

TAs:
Section 1 (Wed): Ege Berkay Gülcan, Zülal Bingöl
Section 2 (Fri): Pouya Ghahramanian, Yusuf Dalva
Section 3 (Sun): Yusuf Dalva, Zülal Bingöl
Section 4 (Thu): Berkay Gülcan, Eren Çalık

TA email Addresses:
Berkay Gülcan: berkay.gulcan@bilkent.edu.tr
Eren Çalık: eren.calik@bilkent.edu.tr
Pouya Ghahramanian: ghahramanian@bilkent.edu.tr
Yusuf Dalva: yusuf.dalva@bilkent.edu.tr
Zülal Bingöl: zulal.bingol@bilkent.edu.tr

Purpose: Studying the effect of various cache design parameters. The first part includes problem solving and writing a program. The second part involves execution of the program and preparing a report.

In your solutions and report make sure that you have proper tables, page numbering and understandable explanation with good writing. All tables must have a subtitle and table number. In tables columns must have meaning names/headers.

Summary
Part 1 (50 points): Involves problem solving related to cache memory design and a program written for cache testing.
Part 2 (50 points): Experiments with caching and experiment report.

DUE DATE: SAME FOR ALL SECTIONS: As given above.

Submit Problem Solutions, Experiment Report, and Your Code for MOSS similarity testing

Report: Use filename **StudentID_FirstName_LastName_SecNo_Lab6_report.pdf** [pdf FILE as its extension suggests, which contains all the work done for the problem solutions and Lab Experiment Report Part].

Code: For the program part Use filename **StudentID_FirstName_LastName_SecNo_Lab6_code.txt** [A NOTEPAD FILE as its extension suggests, which contains the Program Code Part].

Your program (code) will be compared against all the other programs in the class, by the MOSS program, to determine how similar it is (as an indication of plagiarism). So be sure that the code you submit is code that you actually wrote yourself! The same type of comparison is also planned for the reports.

Part 1. Cache Memory Problems and Program (50 points)

You have to provide a neat presentation prepared by Word or a word processor with similar output quality. Handwritten answers will not be accepted. At the top of the paper on left provide the following information. Please make sure that this info is there for proper grading of your work, otherwise some points will be taken off.

CS224
Section No.: ...
Spring 2020
Lab No.:
Your Full Name/Bilkent ID:

1. (5 points: With 3 or more errors you get 0 points. Otherwise full point.) Fill in the empty cells of the following table. Assume that main memory size is 0.5 GB. **Index Size**: No. of bits needed to express the set number in an address, **Block Offset**: No. of bits needed to indicate the word offset in a block, **Byte Offset**: No. of bits needed to indicate the byte offset in a word. **Block Replacement Policy Needed**: Indicate if a block replacement policy such as FIFO, LRU, LFU (Least Frequently Used) etc. is needed (yes) or not (no). If some combinations are not possible mark them.

Case No.	**Cache Size KB**	**N way cache**	**Word Size in bits**	**Block size (no. of words)**	**No. of Sets**	**Tag Size in bits**	**Index Size (Set No.) in bits**	**Word Block Offset Size in bits**[1]	**Byte Offset Size in bits**[2]	**Block Replacement Policy Needed (Yes/No)**
1	4	1	32	4						
2	4	2	32	4						
3	4	4	32	8						
4	4	Full	32	8						
5~~9~~	32	1	16	4						
6~~10~~	32	2	16	4						
7~~11~~	32	4	8	16						
8~~12~~	32	Full	8	16						

[1] **Word Block Offset Size in bits:** Log_2(No. of words in a block)
[2] **Byte Offset Size in bits:** Log_2(No. of bytes in a word)

2. (5 points: With 3 or more errors you get 0 points. Otherwise full point.) Consider the following MIPS code segment. (Remember MIPS memory size is 4 GB.) Cache capacity is 16 words, Block size: 4 words, N= 1.

```
            addi    $t0, $0, 5
loop:       beq     $t0, $0, done
            lw      $t1, 0xA4($0)
            lw      $t2, 0xAC($0)
            lw      $t3, 0xA8($0)
            addi    $t0, $t0, -1
            j       loop
done:
```

a. In the following table indicate the type of miss, if any: Compulsory, Conflict, Capacity.

Instruction	**Iteration No.**				
	1	**2**	**3**	**4**	**5**
lw $t1, 0xA4($0)					
lw $t2, 0xAC($0)					
lw $t3, 0xA8($0)					

b. What is the memory size of one set in number of bits? What is the total cache memory (SRAM) size in number of bits? Note: Include the V bit in your calculations. Show the details of your calculation.

c. State the number of AND and OR gates, EQUALITY COMPARATORs and MULTIPLEXERs needed to implement the cache memory. No drawing is needed.

3. **(5 points: With 3 or more errors you get 0 points. Otherwise full point.)** Consider the above MIPS code segment. The cache capacity is 8 words , block size is 1 word. N= 2. The block replacement policy is LRU.

a. In the following table indicate the type of miss, if any: Compulsory, Conflict, Capacity.

Instruction	**Iteration No.**				
	1	**2**	**3**	**4**	**5**
lb $t1, 0xA4($0)					
lb $t2, 0xAC($0)					
lb $t3, 0xA8($0)					

b. How many bits are needed for the implementation of LRU policy: for a set, for the entire cache memory? What is the total cache memory size in number of bits? Include the V bit and the bit(s) used for LRU in your calculations. Show the details of your calculation.

c. State the number of AND and OR gates, EQUALITY COMPARATORs and MULTIPLEXERs needed to implement the cache memory. No drawing is needed.

4. **(35 points)** Write a program to find the summation of the elements of a square matrix. Provide a user interface for user interaction to demonstrate that your program is working properly. Assume that in the main memory matrix elements are placed row by row. Create an array for the matrix elements and initialize them row by row with consecutive values. For example, a 3 by 3 (N= 3) matrix would have the following values.

1	2	3
4	5	6
7	8	9

The row by row placement means that you will have the values of the above 3 x 3 matrix are stored as follows in the memory.

Matrix Index (Row No., Col. No.)	(1, 1)	(1, 2)	(1, 3)	(2, 1)	(2, 2)	(2, 3)	(3, 1)	(3, 2)	(3, 3)
Displacement With respect the beginning of the array containing the matrix	0	4	8	12	16	20	24	28	32
Value stored	1	2	3	4	5	6	7	8	9

In this configuration accessing the matrix element (i, j) simply involves computation of its displacement from the beginning of the array that stores the matrix elements. For example, the displacement of the matrix element with the index (i, j) with respect to the beginning of the array is (i - 1) x N x 4 + (j - 1) x 4, for a matrix of size N x N.

Your user interface must provide at least the following functionalities,

1. Ask the user the matrix size in terms of its dimensions (N),
2. Allocate an array with proper size using syscall code 9,
3. Display desired elements of the matrix by specifying its row and column member
4. Obtain summation of matrix elements row-major (row by row) summation, and display
5. Obtain summation of matrix elements column-major (column by column) summation, and display

2. [50 pts] Experiments with Data Cache Parameters

Run your program with two reasonably large different matrix sizes that would provide meaningful observations. Modify your original program if needed. For large matrix initialization you may use a loop rather than user interface.

Report for Matrix Size 1: 25 Points

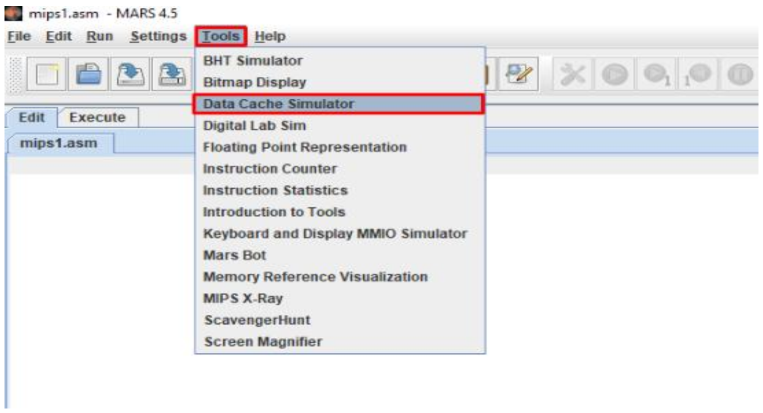
Report for Matrix Size 2: 25 Points

As described above make sure that you have an easy to follow presentation with numbered tables having proper heading etc.

Make sure that you find the summation of matrix elements by performing row-major and column-major addition. Note that the column-major addition is a simple array addition from the beginning to the end; however, the row-major addition is somewhat tricky.

a) **Direct Mapped Caches**: For the matrix sizes 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 (see below) in the textbook. This figure shows that as we increase the block size the number of compulsory misses decrease and it decreases the overall miss rate. If we continue in that fashion after a point number of conflict misses start to increase and the overall miss rate starts to increase. Make a 5 x 5 table with your values, with miss rate and # of misses as the data at each row-column location. Make the graph of miss rate versus block size, parameterized by cache size, like Figure 8.18 both for row-major and column-major additions.

Hint: You can reach the Cache Simulator from MARS/Tools/Data Cache Simulator as shown in the following image:



b) **Fully Associative Caches**: Pick 3 of your parameter points obtained in part for row-major addition 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 graph, with 3 lines: for Direct Mapped, for Fully Associative-LRU and for Fully Associative-Random. Note that this step is only for row-major addition so that by the end of the lab you'll have (2+1)x2 = 6 graphs.

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.

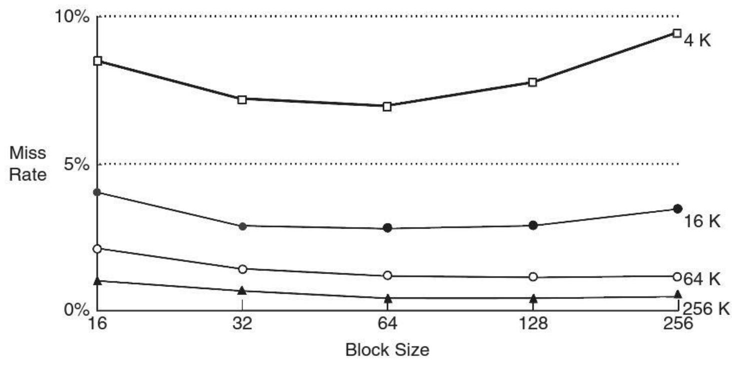


Figure 8.18 **Miss rate versus block size and cache size on SPEC92 benchmark**
Adapted with permission from Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 5th ed., Morgan Kaufmann, 2012.