ECE232 Lab Project 2

MONITORING CACHE BEHAVIORS USING SPIM-CACHE

Due Thursday, April 2, 11:00 PM EST (via Moodle)

1. Objectives

- Understand how direct mapped, 2-way set associative caches, 4-way set associative caches and fully associative caches work.
- Understand the impact of cache size and block size on cache performance.
- Understand the system performance impact of using caches.
- Regain familiarity with converting a C language to assembly code.
- Gain experience with the Spim-Cache simulator

2. Spim-Cache

A link to download the Spim-Cache simulator is provided on the course website under the "Resources" page. A tutorial is also provided on the same page. This software is similar to the QTSpim simulator. A sample assembly code program (matrix_first_row.s) is also provided.

Both the data cache and the instruction cache configuration can be modified in the Spim-Cache simulator. Please follow the steps in the tutorial. In this lab assignment, **you only need to change the configuration of the data cache**. The configuration for the instruction cache should remain default.

3. Symmetric Matrix Conversion

A general quadratic form Q of order n is written as

$$Q = \sum_{j=1}^{n} \sum_{k=1}^{n} a_{jk} x_j x_k \tag{1}$$

In matrix form, it can be written as

$$Q = \mathrm{x}^{\mathrm{T}} \mathrm{A} \mathrm{x} \tag{2}$$

Any quadratic form can be written in terms of a symmetric matrix. If the given matrix is not symmetric, it can be replaced by the symmetric matrix *B* without changing the value of the form.

$$B=\frac{1}{2}(A+A^T) \tag{3}$$

In this lab, you are provided an assembly code program to perform symmetric conversion of a matrix using Equation (3) (*Equations 1 and 2 are background information. You don't really need these two equations in this lab.).* You will use the SpimCache simulator to simulate this assembly code program and monitor the performance of the data cache. Then you are asked to improve the cache performance by either changing the cache configuration or modifying the input matrix.

Before you can simulate the program, the first task for you is to initialize the matrix A. All elements in matrix *A* are **stored in words** in consecutive memory spaces. The matrix is stored in a **row by row** style: all elements in each row are stored in consecutive memory space. Please refer to sample code for detailed matrix storing information.

Similar to Lab 1, you can use the "load address" (`la`) pseudoinstruction to get the base address of a matrix (e.g. Matrix *A* shown below). (You may use other registers as well).

```
la $t0, Matrix    # register #t0 gets the base
                    address of Matrix A
```

To access a particular element in matrix A, you have to figure out the byte offset of the element *A(i, j)*. In our case, the offset can be calculated using the following formula.

$$offset = i\times 68 + j\times 4 \quad i = 0,1,\ldots,16 \text{ and } j = 0,1,\ldots,16 \tag{5}$$

You can also refer to the sample assembly program provided with the Spim-Cache tutorial, which calculates the sum of the element in the first column (four elements, 1, 2, 3 and 4) of a matrix.

The following code shows how to calculate the multiplication of two 32 bit values, stored in `$t0` and `$t1` respectively (You need to perform multiplication operations as shown in Equation 5). The lower 32 bit of the result (64 bit in total) can be retrieved and stored in register `$t2`. Don't worry about the higher 32 bits since they are always all zeroes in our case.

```
mult $t0,$t1 # calculate $t0 × $t1, result is 64 bit
mflo $t2     # move the low 32 bit of the result to $t2
```

4. Materials for Submission

For this lab assignment you must turn in a written report (in pdf or Word format) that includes all the answers to the following questions and separate files including your MIPS assembly code (e.g. lab2_question3.s, lab2_question4.s). Your final score for this lab is based on your answer to the following questions and the assembly code.

For the following questions, **PLEASE PROVIDE DETAILED EXPLANATIONS TO YOUR ANSWERS. YOUR SCORE IS BASED ON HOW GOOD YOUR EXPLANATIONS ARE.**

1. The C program and assembly program to perform symmetric conversion are given in Section 5 and Section 6, but the initialization of the matrix in the assembly program is missing. Please initialize a 17 by 17 matrix using the numbers from 0 to 288, then run the program and measure the data cache miss rate with the following data cache configuration: Cache size 256 B, Block size 8 B, Cache Mapping: Direct Map, Writing Policy: Write Through-Allocate, Replace Policy: LRU. Provide a screenshot of the first row of resultant values in the **data memory** and a screenshot which shows the **data cache hit rate**. (10 points)

   Note:

   1) **You do not need to change the configuration of the instruction cache. Please refer to the Spim-Cache tutorials for how to change data cache configurations.**

2) **You have two ways of initialize the matrix, the first way is the same way as the sample code, manually typing all the numbers row by row; the second way is: initialize a 17 by 17 empty array and use a loop to initialize all the elements in the program. You can use either way. For the second way, you can refer: http://pages.cs.wisc.edu/~smoler/cs354/onyourown/arrays.html.**

2. Keep the same cache configuration, if you want to increase the data cache hit rate to more than 80%, how would you change the data matrix? Do an experiment to prove your assumption. Show your new **data cache hit rate** result with a screenshot and compare this number with the result in question 1. Explain why the change of the matrix increased the hit rate. (20 points)

3. Keep the matrix the same as the 17 by 17 matrix in question 1. If you are allowed to change one parameter in the data cache configuration in question 1, what would you change to increase the data cache hit rate? Do an experiment to prove your assumption. Show your new **data cache hit rate** result with a screenshot and compare this number with the result in question 1. Explain why the hit rate went up. (20 points)

4. Provide the assembly code for question 1, and question 2. (10 points)

5. Short answer questions (not related to the experiments you performed for Q1 through Q4):
   a) Why is cache used in computer systems? (5 points)
   b) How big is cache relative to main memory? (5 points)
   c) What is write back policy and write through policy, and what is the main difference between them? (10 points).

6. Solve the following cache problem. (20 points)

   Given the following assumptions, fill out the table of data cache content on the next two pages based on memory access sequence shown in the following table.

- Cache is initially empty
- The data cache is a direct mapped cache
- The width of memory address is 10-bit and it is a **byte address**
- The cache block size is 4 bytes
- Write through strategy is used in this cache

Memory Accesses

#	Access Size	Access Type	Byte Address in Binary
1	Byte	Read	11 0101 1011
2	Byte	Read	11 0101 1010
3	Word	Read	11 1111 1100
4	Word	Read	11 0101 1000
5	Byte	Read	00 0000 0001
6	Word	Write	10 0000 0000
7	Word	Read	00 0001 0000
8	Byte	Read	00 0000 0011
9	Word	Read	00 0001 1000
10	Byte	Read	10 0000 0010

Data Cache Content

Note: a.) For locations never accessed, leave the corresponding tag and data boxes blank.

b.) Indicate the valid bit for each cache block. Use “V” for valid and “I” for invalid.

c.) The block data should be given in the format of **M[*block address*],** where ***block address*** is the 10-bit byte address of the first data byte in the block

d.) Mark each access in order for up to 4 accesses. The first access should be the first one listed.

Index	V	Tag	Data
000			
001			
010			
011			
100			
101			
110			
111			

5. C Program

A C program performing the symmetric conversion (Equation 3) mentioned in Section 3 is shown below. Your correspondent assembly program is in Section 6.

Note:

1. ***The data set in Matrix is not fully written out but it should be initialized by you.***
2. ***We perform shift right by 1 instead of divide by 2 just for simplicity. The fraction is ignored here, so don't worry about that.***

```
int Matrix[17][17] = {{0, 1, 2, 3,  … 16},   \
                      {17, 18, 19, 20, … 33},   \
                      … …                       \
                      {272, 273,  274, 275,  … 288}};

void main()
{
  int temp;

  //outer loop for each row
  for(int i = 0; i < 17; i++ )
  {
    //inner loop for each column;
    //only calculate the upper triangle
    //lower triangle holds the same values
    for(int j = i; j < 17 ; j++ )
    {
      //sum of an element and its diagonal reflection
      //shift right by 1 is equivalent to divide by 2
      temp = (Matrix[i][j]+Matrix[j][i])>>1;

      //store averaged data to upper and lower triangle
      Matrix[i][j]= temp;
      Matrix[j][i]= temp;
    }
  }
 }
```

6. Assembly Code Template

Please copy the following code to your .s file as the beginning of your program. Replace the “Initialize the matrix” section with your own matrix initialization code.

```
        .data
Matrix: ##########################
        # Initialize the Matrix
        ##########################

       .text
       .globl  __start
__start:  addi $s0,$zero,0         # outer loop counter
          addi $s1,$zero,0         # inner loop counter
          la   $t0,Matrix          # Matrix, base address
          addi $t6,$zero,17        # number of columns
          addi $t7,$zero,17        # number of rows
          sll  $t1,$t6,2           # byte numbers in a row
ext_loop: addi $s1,$s0,0     # reset the inner loop counter
inn_loop: mult $s0,$t1
          mflo $t2          # calculate the low byte offset
          sll  $t3,$s1,2      # calculate the column byte offset
          add  $t4,$t0,$t2         # add low offset
          add  $s2,$t4,$t3         # add column offset
          lw   $s3,0($s2)          # load this element
          mult $s1,$t1        # same for the other data
          mflo $t2
          sll  $t3,$s0,2
          add  $t4,$t0,$t2
          add  $s4,$t4,$t3
          lw   $s5,0($s4)
          add  $s6,$s3,$s5         # addition
          srl  $s7,$s6,1           # divided by 2
          sw   $s7,0($s2)          # store the result back
          sw   $s7,0($s4)          # store the result back
          addi $s1,$s1,1        # increment the inner loop
          blt  $s1,$t7,inn_loop    # next inner loop
          addi $s0,$s0,1         # increment the outer loop
          blt  $s0,$t6,ext_loop    # next outer loop
          .end
```