p4A: Understanding Caches

Due Mar 31 by 10pm **Points** 20 **Submitting** a file upload
Available until Mar 31 at 10:33pm

This assignment was locked Mar 31 at 10:33pm.

GOALS FILES PIN SPECIFICATIONS REQUIREMENTS SUBMITTING

- This project must be done individually.
    - *It is academic misconduct to share your work with others in any form including posting it on publicly accessible web sites, such as GitHub.*
    - *It is academic misconduct for you to copy or use some or all of a program that has been written by someone else.*
- All work for this project is to be done on the CS Department's instructional Linux machines (https://csl.cs.wisc.edu/services/instructional-facilities) . You're encouraged to remotely login using the ssh command in the terminal app on Macs, or on Windows using an ssh client such as MobaXterm.
- All projects in this course are graded on CS Department's instructional Linux machines (https://csl.cs.wisc.edu/services/instructional-facilities) . To receive credit, make sure your code runs as you expect on these machines.

Learning GOALS

This assignment has 2 parts - part A and part B. The purpose of the part A is to understand more about how caches work and to learn a bit about cache simulation. In part B, which will be assigned later, you'll work on developing a small cache simulator.

Getting the FILES

Create the directory **p4A** inside your **cs354** directory as shown below.

```
[skrentny@jimbo] (33)$ mkdir p4A
```

Next copy the directory **pin-fw** into your **p4A** directory as shown below. Note: This may take several minutes.

```
[skrentny@jimbo] (34)$ cd p4A/
[skrentny@jimbo] (35)$ cp -r /p/course/cs354-skrentny/public/code/p4A/pin-fw  ./
[skrentny@jimbo] (36)$ ls
pin-fw/
```

Using PIN

pin is a dynamic binary instrumentation framework that can be used to explore how a computer's architecture affects a program's execution. We'll use **pin** to measure cache performance for several simple programs run with different cache block sizes. You'll report your results by completing the "**p4AQuestions**" quiz on canvas.

<u>NOTE: To ensure your results are as expected, you must complete this project on the CS Department's instructional Linux machines</u> (either physically or remotely via ssh).

pin runs an executable to generate a trace of the memory addresses the executable accesses as it runs. Each address represents either a read for an instruction fetch or a read/write of some data stored in memory. This address trace is then internally shared with a particular tool in pin, which for us will be the **cache simulator**. The cache simulator determines for each address in a trace whether it causes a cache hit or a miss. It tracks these hits/misses so that it can display final statistics.

To run a simulation, from inside your **p4A** directory, you will use a command line of the format:

```
pin-fw/pin -injection child -t pin-fw/source/tools/Memory/obj-ia32/dcache.so -c <capacity> -a <associativity> -b <block-size> -o <output-file> -- <your-exe>
```

In this command, you need to replace `<your-exe>` with the name of your executable file, and `<output-file>` with the name of the output file. The specific values of the other cache parameters are specified below. The simulations you'll be running will focus on the L1 (cache level 1) **D-cache**, which caches data that is read or written while a program runs. At cache level 1, there is a separate cache for machine instructions that are read while the program does instruction fetches.

There are 3 cache parameters for D-cache (Note: The maximum number of sets that the D-cache supports is 1024.):

Capacity:

```
    -c <capacity>
```

This sets the total capacity of the D-cache in kilobytes. <u>For all your simulations</u>, use a 2KB or 2048 bytes size for the D-cache as in: `-c 2`

Associativity:

```
    -a <associativity>
```

This sets the set associativity of the D-cache. <u>For all your simulations</u>, use an associativity of 1 (direct-mapped) as in: `-a 1`

Block size:

```
    -b <block-size>
```

This sets the block size in bytes of the D-cache. You'll be changing this parameter to answer the questions in the quiz.

SPECIFICATIONS

In order to answer the questions in the **p4AQuestions** quiz, you will need to write 3 simple programs named "**cache1D.c"**, "**cache2Drows.c**" and "**cache2Dcols.c"**, and then compile them into executables as you have done in previous programs like:

```
gcc cache1D.c -Wall -m32 -std=gnu99 -o cache1D
```

cache1D.c:

Declare a global array of integers of size 100,000 before the **`main()`** function, so that this array will be in the data segment. Use a for loop in the **`main()`** function to iterate over the entire array and set the value of each element in the array to its index, as in:

```
    arr[i] = i;
```

Do not do anything else (e.g., print to the console). All this program should do is iterate over the global array to set its values.

Use the executable from this program to answer the **cache1D questions of p4AQuestions** quiz.

cache2Drows.c:

cache2Drows.c has a 2-dimensional global array of integers having dimensions 3000 rows x 500 columns. In the **`main()`** function, traverse the array in row-wise order. In other words, **have an inner loop iterate through the elements of a single row of the array, and have an outer loop iterate through the rows of the array**. As the array is traversed, set each element of the array to the sum of its row and col indexes, as in:

```
    arr2D[row][col] = row + col;
```

Use the executable from this program to answer the **cache2Drows questions of p4AQuestions** quiz.

cache2Dcols.c:

cache2Dcols.c has the same 2D array as above but now the traverses the array in a column-wise order. In other words, **have an inner loop iterate through the elements of a single column of the array, and have the outer loop iterate through the columns of the array**.

Use the executable from this program to answer the **cache2Dcols questions of p4AQuestions** quiz.

Finish by answering the comparison **cache2Drows and cache2Dcols questions of p4AQuestions** quiz.

REQUIREMENTS

- Your simple programs must be reasonable in coding style and only need, for comments, a header with your name.
- Your simple programs must operate exactly as specified or you'll get the wrong results for the **p4AQuestions** quiz.
- We will compile your programs with

  ```
  gcc -Wall -m32 -std=gnu99
  ```

  on the Linux lab machines. So, your programs must compile there without warnings or errors.

SUBMITTING Your Work

In addition to completing the **p4AQuestions** quiz before the deadline, submit the following **source files** under Project p4A in Assignments on Canvas before the deadline:

1. cache1D.c
2. cache2Drows.c
3. cache2Dcols.c

It is your responsibility to ensure your submission is complete with the correct file names having the correct contents. The following points will seem obvious to most, but we've found we must explicitly state them otherwise some students will request special treatment for their carelessness:

- **You will only receive credit for the files that you submit.** You will not receive credit for files that you do not submit. Forgetting to submit, not submitting all the listed files, or submitting executable files or other wrong files will result in you losing credit for the assignment.
- **Do not zip, compress or submit your files in a folder, or submit each file individually.** Submit only the text files as listed as a single submission.
- **Make sure your file names exactly match those listed.** If you resubmit your work, Canvas will modify the file names by appending a hyphen and a number (e.g., cache1D-1.c) and these Canvas modified names are accepted for grading.

Repeated Submission: You may resubmit your work repeatedly until the deadline has passed. **We strongly encourage you to use Canvas as a place to store a back up copy of your work.** If you resubmit, you must resubmit all of your work rather than updating just some of the files.

Project p4A

Criteria	Ratings		Pts
All programs compile without warnings or errors	**5.0 pts Full Marks**	**0.0 pts No Marks**	5.0 pts
cache1D pin outputs looks right pin output looks plausible with block sizes of 4, 32, and 64 bytes. Hit rates look right for the resulting pin output.	**5.0 pts Full Marks**	**0.0 pts No Marks**	5.0 pts
cache 2D rows pin outputs looks right pin output plausible with block sizes of 4, 32, and 64 bytes. Hit rates look right for the resulting pin output.	**5.0 pts Full Marks**	**0.0 pts No Marks**	5.0 pts
cache 2D cols pin outputs looks right pin output plausible with block sizes of 4, 32, and 64 bytes. Hit rates look right for the resulting pin output.	**5.0 pts Full Marks**	**0.0 pts No Marks**	5.0 pts
			Total Points: 20.0