p4A: Understanding Caches

Due Mar 31 by 11:59pm **Points** 20 **Submitting** a file upload
Available Mar 20 at 12am - Apr 1 at 11:59pm

This assignment was locked Apr 1 at 11:59pm.

Announcements:

- Released - Get Started Today

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. Completing part A should not take more than a few minutes for each of four small C programs you need to write. You must use your programs from this assignment to answer the questions in **p4AQuestions (https://canvas.wisc.edu/courses/330348/assignments/1852094)** .

In part B, you'll complete development on a small cache simulator.

Getting the FILES (hint: there are none provided)

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

```
[deppeler@liederkranz] (33)$ mkdir p4A
[deppeler@liederkranz] (34)$ cd p4A/
[deppeler@liederkranz] (36)$ ls -al     # NOTE: THERE ARE NO PROVIDED STARTER FILES, THIS WILL BE EMPTY UNTIL
YOU WRITE YOUR p4A programs
```

Using PIN to measure cache performance

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 (after you create and build the programs in this assignment). Don't wait, they won't take much time.

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 for the executable that was measured.

To run **pin:** cd to your **p4A** directory, build your executable, (as described in the Specifications section), and use a command line of the format:

```
/s/pin-3.16/pin -injection child -t /s/pin-3.16/source/tools/Memory/obj-ia32/dcache.so -c <capacity> -a <asso
ciativity> -b <block-size> -o <output-file> -- <your-exe>
```

In this command, you need to replace:

- `<your-exe>` with the name of your executable file
- `<output-file>` with the name of the output file (CAUTION: DO NOT PUT YOUR SOURCE FILE NAME HERE)
- the specific values of the other cache parameters. *Each cache parameter is described in next section.*

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 the 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.
For your simulations, use the capacity specified by the question (in p4A Questions).
If no capacity is specified, 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.
For your simulations, use the associativity specified by the question.
If no associativity is specified, 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 p4A Questions quiz.

Output File

```
-o <output-file>
```

This file name must be a new name for the output of this command.
DO NOT PUT YOUR SOURCE CODE FILENAME HERE
or you will overwrite the C program code that you just wrote.

SPECIFICATIONS

In order to answer the questions in the **p4AQuestions** quiz, **you will need to write four simple programs** that explore different sequences of accessing elements in arrays in the DATA segment of an executable object file.

The programs must be named "**cache1D.c"**, "**cache2Drows.c**", "**cache2Dcols.c"** and **"cache2Dclash.c"**
You will then compile them into executables as you have done in previous programs like:

```
gcc cache1D.c -Wall -m32 -std=gnu99 -o cache1D   // the program and output name change for each program
```

Each of these small programs require you to declare global arrays. To do this you will declare your array as you would for a stack allocated array but put the declaration outside of any function. This causes the array to be allocated in the **DATA** segment of memory and the 2D arrays use the same row major order layout as stack allocated 2D arrays. No code skeletons are provided for these programs.

Pre-processor Commands

You can define global constants in C source files with the #define pre-processor command, place pre-processor commands before any code. You are not required to use this command for your constants.

```
#define GLOBAL_N 10
```

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.

Answer the comparison **cache2Drows and cache2Dcols questions of p4AQuestions** quiz.

cache2Dclash.c:

cache2Dclash.c has a 2-dimensional global array of integers having dimensions 128 rows x 8 columns. In the **main()** function, you will will have three loops. The innermost loop iterates over the columns of the array. The middle loop iterates over the rows, **incrementing by 64 instead of 1**. The outermost loop repeats this 100 times. As the array is traversed, set each element of the array to the sum of the three corresponding indexes, as in:

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

Finish by answering the **cache2Dclash questions of p4AQuestions** quiz.

REQUIREMENTS

- Write and compile your four test programs on CSL workstations, using Linux IA-32 machine settings (or you may get different results for p4A quiz).
  - Your simple programs must be reasonable in coding style and only need a header comment 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 cacheX.c -Wall -m32 -std=gnu99 -o cacheX** on the Linux lab machines. So your program must compile there, and without warnings or errors.

- Submit your p4A source code files (described above)
- After submitting your source code for this assignement (see below), use your p4A executable files and the pin command described above to answer p4A Questions (quiz).

SUBMITTING Your Work

Complete the p4AQuestions quiz before the deadline based on your work in this project.

Leave plenty of time before the deadline to complete the two steps for p4A submission found below. Submit your source files to Canvas via this form before the due date and before you use them for p4A Questions quiz. This submission form is available for 24 hours after the assignment due date. Work that is not submitted before the due date and time, is marked LATE and incurs a 10% deduction. No submissions or updates to submissions are accepted after the LATE period.

1.) Submit only the files listed below under Project p4A in Assignments on Canvas as a single submission. Do not zip, compress, submit your files in a folder, or submit each file individually.

```
cache1D.c
cache2Drows.c
cache2Dcols.c
cache2Dclash.c
```

Repeated Submission: You may resubmit your work repeatedly so we strongly encourage you to use Canvas to store a backup of your current work. If you resubmit, Canvas will modify your file names by appending a hyphen and a number (e.g., cache1D-1.c).

2.) Verify your submission to ensure it is complete and correct. If not, resubmit all of your work rather than updating just some of the files.

- **Make sure you have submitted all the files listed above.** Forgetting to submit or not submitting one or more of the listed files will result in you losing credit for the assignment.
- **Make sure the files that you have submitted have the correct contents.** Submitting the wrong version of your files, empty files, skeleton files, executable files, corrupted files, or other wrong files will result in you losing credit for the assignment.
- **Make sure your file names exactly match those listed above.** If you resubmit your work, Canvas will modify your file names as mentioned in **Repeated Submission** above. These Canvas modified names are accepted for grading.

Project p4A (1) (1)

Criteria	Ratings			Pts
All programs compile without warnings or errors	**4 pts** **Full Marks**		**0 pts** **No Marks**	4 pts
cache1D.c outputs look correct pin output plausible with block sizes of 32 and 64 bytes. Hit rates look right for the resulting pin output.	**4 pts** **Full Marks**	**2 pts** **Partly Correct**	**0 pts** **No Marks**	4 pts
cache2Drows.c outputs look correct pin output plausible with a block size of 64 bytes. Hit rates look right for the resulting pin output.	**4 pts** **Full Marks**		**0 pts** **No Marks**	4 pts
cache2Dcols.c outputs look correct pin output plausible with a block size of 64 bytes. Hit rates look right for the resulting pin output.	**4 pts** **Full Marks**		**0 pts** **No Marks**	4 pts
cache2Dclash.c outputs look correct pin output plausible with cache associativities of 1 and 2. Hit rates look right for the resulting pin output.	**4 pts** **Full Marks**	**2 pts** **Partly Correct**	**0 pts** **No Marks**	4 pts
				Total Points: 20