Computer Architecture
Assignment 5 - Analysing Cache Performance

Instructor : Abhishek Bhattacharjee

Due : May 5, 2018 11:55 PM

1 Overview

The goal of this assignment is to help you understand caches better. This assignment is comprised of two parts. In the first part, you are required to write a cache-friendly C program by improving cache behaviour. For the first part you should run your code on the iLab machine. In the second part, the focus is on writing a microbenchmark using the C programming language to understand the memory hierarchy. For the second part, the program should be run on both your laptop and on iLab machines.

2 Part 1 (20 points)

2.1 Overview

In this part, you are required to write a cache-friendly C program by improving cache behaviour. We provide a naive matrix multiplication C code. Your goal is to rewrite a matrix multiplication code by using the following methods in the following section (Loop fusion, Loop interchange, and Loop blocking) and then compare the cache performance on iLab machine.

2.2 Usage interface

Download the given matrix multiplication C code (`matrix.c`) and run with the sample file. The first line of the sample file will contain 2 space-separated numbers, m1 and n1, where m1 is the number of rows in the first matrix and n1 is the number of columns in the first matrix. This will be followed by the m1 lines of first matrix, containing n1 space-separated values, followed by a blank line, then 2 space-separated numbers, m2 and n2, where m2 is the number of rows in the second matrix and n2 is the number of columns in the second matrix. Again, this is followed by the m2 lines of the second matrix. Each row will contain n2 space-separated values, the same as the first matrix. The `matrix.c` code multiplies these two matrices which are matrix1 and matrix2. Run this naive C code as:

```
$ gcc -O -o matrix matrix.c
$ ./matrix sample1.txt
```

The result will show the time of running the program. Rewrite this naive matrix multiplication code by using loop fusion, loop interchange, and loop blocking. Then run your code as:

```
$ gcc -O -o first first.c
$ ./first sample1.txt
```

and compare the measured time with the naive C code.

2.3 Ways to improve cache performance

2.3.1 Loop Fusion

Many programs have separate loops that operate on the same data. Simply combining these loops allows a program to take advantage of temporal locality by grouping operations on the same cached data together.

2.3.2 Loop Interchange

Some programs have nested loops that access data in memory in nonsequential order. Simply exchanging the nesting of the loops can make the code access the data in the order in which they are stored. Assuming the arrays do not fit in the cache, this technique reduces misses by improving spatial locality; reordering maximizes use of data in a cache block before they are discarded.

2.3.3 Loop Blocking

This optimization improves temporal locality to reduce misses. We are again dealing with multiple arrays, with some arrays accessed by rows and some by columns. Storing the arrays row by row or column by column does not solve the problem because both rows and columns are used in every loop iteration. Such orthogonal access mean that transformations such as loop interchange still leave plenty of room for improvement.

Instead of operating on entire rows or columns of an array, blocked algorithms operate on submatrices which are called *blocks*. The goal is to maximize accesses to the data loaded into the cache before the data are replaced. For example, suppose we want to compute C = AB, where A, B, and C are each 8 x 8 matrices. Then we can partition each matrix into four 4 x 4 submatrices:

$$\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

where,

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

2.4 Comparing Cache performance

In this assignment we will use *'perf'* command to measure the cache performance. 'Perf' is a performance analyzing tool in Linux. It supports hardware performance counters, tracepoints, software performance counters, and etc. We will use *'stat'* options to measure the cache performance. To be specific we will use *L1-dcache-loads, L1-dcache-load-misses, L1-dcache-stores, L1-dcache-store-misses, LLC-loads, LLC-load-misses, LLC-stores, LLC-store-misses*. These options will show the total number of L1 data cache, last level cache loads, stores, and load, store misses respectively. The usage of 'perf' is:

```
$ perf stat -e cache-misses,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-store-misses,LLC-loads,LLC-load-misses,LLC-stores,LLC-store-misses ./matrix sample1.txt
```

```
$ perf stat -e cache-misses,L1-dcache-loads,L1-dcache-load-misses,L1-dcache-stores,L1-dcache-store-misses,LLC-loads,LLC-load-misses,LLC-stores,LLC-store-misses ./second sample1.txt
```

For more details, please refer to: https://perf.wiki.kernel.org/index.php/Tutorial for more information.

2.5 Report

You should submit a report in the PDF format. In this assignment, your report will be also graded and it is important as well as your source code. In your document, you should include the time comparison of the given code and your code. Include all the results after running perf. Also, you should compare the cache miss rate of the given code and your code, then explain the reason of your result.

3 Part 2 (80 points)

3.1 Background

As you know from class, CPUs operate at a faster rate compared to the main memory. In some implementations, this speed discrepancy is of several orders of magnitude.

To run a program, the CPU needs to fetch an instruction and possibly data from the memory at each clock cycle. Without caching instruction and data on a faster memory, the CPU has to stall on every instruction until the data is available to it. This greatly diminishes the speed of a program. That's why most commercial CPUs contain a faster memory called CPU cache that stores instruction and data.

CPU caches are transparent to the application programmer. However, almost all programs, to varying degree, benefit from the inclusion of the cache. If a programmer is aware of the cache layout and the principles of spatial and temporal locality in memory references, they can write their program in a way to benefit more from the cache. This can result in a noticeable decrease of the running time of the program.

Caches are characterized by three parameters which are cache size, associativity, and block size. It is assumed that you know about these parameters. For more information about these parameters you can check those from your lecture slides.

3.2 Microbenchmark

You will implement a microbenchmark to analyse your cache. In this part you should assess the **cache size, set assocativity,** and **block size** of caches both on your **own laptop and iLab machine**. Use one of the iLab machines from **H254 Machines**. You can find the list of the machines in https://report.cs.rutgers.edu/nagiosnotes/iLab-machines.html for more information. Before you run your code on the iLab machine, please check the status of the iLab machine and use the ones that are not busy.

Below is the outline how your program should be done.

- Since the cache is transparent to the application programmer, there is no direct method to measure the cache size. Therefore, the cache should be measured indirectly.
- The way we achieve this is by running a program that accesses the memory with certain patterns. These accesses lead to capacity misses on the cache. By measuring the running time of the program we can infer the cache size.
- Assume that the cache size, the block size, and the set associativity are a power of two.
- Make sure to declare and initialize all your data structures before measuring the running time.

Your program should support the following usage interface:

```
$ ./second
```

3.3 Report

You should submit a report in PDF format. In this assignment, your report will be also graded and it is important as well as your source code. In your document, you should identify the system you chose to benchmark. For your laptop machine include CPU-core manufacturer, model name, Number of CPU cores on chip, and CPU-core frequency. For the iLab machine just include the name of the machine that you have used. You should then include,

- A brief explanation of what your microbenchmark does and why the microbenchmark should measure the parameters it targets.
- Three graphs with the 'raw' measurements from the microbenchmarks (For example, if you measure the time it takes to do something with varying sizes, a graph of the measurement of those sizes).
- The best guess at the value of each corresponding parameter from the microbenchmark. Namely, the size of the cache, the size of the block, the number of set associatives, and how you determined them. If you were not able to obtain the value of the parameter from your benchmark, explain why you interpret the results that way, what the likely causes are.

4 Important notes

1. We should be able build your program by just running make.
2. Your program should follow the format specified above for the usage interface.
3. Your program should strictly follow the input and output specifications mentioned above. (Note: This is perhaps the most important guideline: failing to follow it might result in you losing all or most of your points for this assignment. Make sure your programs output format is exactly as specified.
4. The report is critical in this assignment. We expect to have minimum of 2 pages of report.
5. You should clearly write a report that analyse your result and explain why it happened.

5 Submission

You have to e-submit the assignment using Sakai. Put all files (source code + header file + Makefile) into a directory named pa5. Then, create a tar file (follow the instructions in the previous assignments to create the tar file). Your submission should be only a tar file named pa5.tar. You have to e-submit the assignment using Sakai. Your submission should be a tar file named pa5.tar. To create this file, put everything that you are submitting into a directory named pa5. Then, cd into the directory containing pa5 (that is, pa5s parent directory) and run the following command:

```
$ tar cvf pa5.tar pa5
```

To check that you have correctly created the tar file, you should copy it (pa5.tar) into an empty directory and run the following command:

```
$ tar xvf pa5.tar
```

This should create a directory named pa5 in the (previously) empty directory. Your pa5 folder should contain the following:

- **Source code**: all source code files necessary for building your programs.
  Your code should contain these files: first.c, first.h, second.c, and second.h.
- **Makefile**: There should be at least three rules in your Makefile:
  first: build the executables (first).
  second: build the executables (second).
  clean: prepare for rebuilding from scratch.
- **Report**: You should submit your report in pdf format containing part 1 and part 2 Please write your Part 1 and Part 2 in **one PDF file**.