CprE 381 - Computer Architecture and Assembly

Level Programming

Fall 2017

# Lab-2

**INTRODUCTION:**

This introductory lab is aimed at introducing you to the *Simplescalar* simulator*,* whileletting you explore some of the topics introduced in section 1.3 of your textbook. It is assumed that you are familiar with C programming and the basics of UNIX command line (see optional readings for materials to brush up your C and command line skills).

**What is SimpleScalar?**

SimpleScalar is a suite of processor simulators and supporting tools. It enables the user to model a virtual computer system with CPU and Memory. It consists of a collection of microarchitecture simulators which emulate the processor at different levels of detail. The user can run programs on these models using execution driven simulation and evaluate performance parameters. You only need to know a few of them for the course, however you are encouraged to play around with them to see the different performance metrics each simulator evaluates.

SimpleScalar is installed on the ECpE Remote Servers(linux-1, linux-2, linux-3 and linux-4). The installation directory contains the simulators (*sim-safe, sim-fast, sim-profile, sim-outorder etc.)* along with some sample benchmark programs located in *tests-pisa* directory. These benchmark files have been precompiled for you. You only need to pass them as arguments to the simulator and analyze how the processor performs.

**Part 1:** **Getting Started**

Create a directory(folder) inside **Drive (X:) ->** **cpre381** named **Lab2**. You will save all your Lab-2 related files here.

Next, run **Putty** from the Start menu and connect to any of the Linux servers (1, 2, 3 or 4).

* To connect to Linux-2 (for example), type **linux-2.ece.iastate.edu** into the host name field, and click **Open**.
* Click **YES** if a pop-up window appears asking for key authentication.
* Enter your login credentials.

You should now get a command-line interface to the server. Navigate to SimpleScalar installation directory using the following command

cd /usr/local/ss/simplesim-3.0

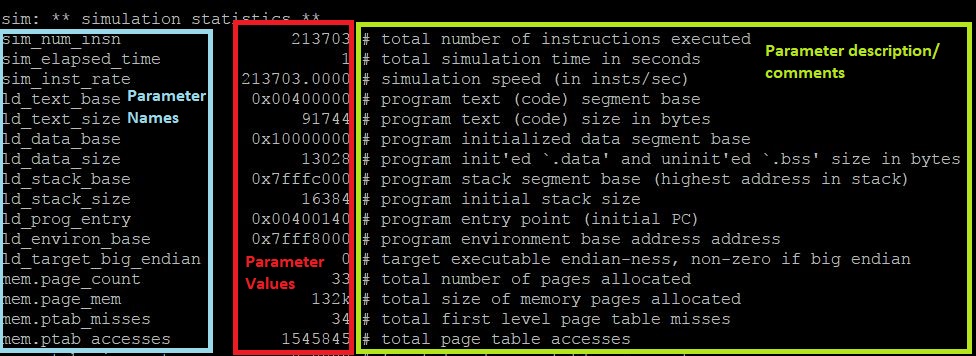
Note: To view all files/directories in your present working directory, use the **ls**command (type **ls** and press the return key).

To view the path to your Present Working directory, use the **pwd** command

(type **pwd** and press the return key).

Next, simulate the program **test-math**located inside **tests-pisa/bin.little**on the **sim-fast** simulation tool*.* To do this, run:

./sim-fast tests-pisa/bin.little/test-math

The simulator will dump the program output and the performance metrics on your console screen. Locate the line ***sim: \*\* starting \*fast\* functional simulation \*\*.*** The program output can be found after this. Towards the end of the output, you will find ***sim: \*\* simulation statistics \*\**** below which the performance metrics are listed (see image below). 

Simulating **test-fmath, test-printf, test-lswlr** and **test-llong***.* Note down the endianness (big or little) and total number of instructions executed for each program using the table below.

|  |  |  |
| --- | --- | --- |
| Test Program | # Instructions | Endian-ness |
| *Test-fmath* | *53459* | *Little* |
| *Test-printf* | *1813891* | *Little* |
| *Test-lswlr* | *8877* | *Little* |
| *Test-llong* | *29642* | *Little* |

**Part 2: Compilers Optimization:**

The compiler plays an important role in your program’s performance. It processes your high level code and turns it into machine code that the processor can understand. In doing so, it performs several tasks such as semantic analysis, syntactic analysis, optimization etc. For this lab, the optimization part is of interest to us. Code optimization is done, in the general sense, to make your code run efficiently or use fewer resources.

We will use the GCC compiler in all of the Simplescalar experiments. GNU Compiler Collection (GCC) was developed by the GNU project, initially for C alone, but now supports other high level languages as well. It can perform different levels of optimization, depending on the requirement. Without optimization (level 0), gcc attempts to make the compilation process faster. Optimization levels 1, 2 and 3 attempt to improve performance of the generated code but at the cost of compilation time, where level 3 implements the most rigorous optimization methods.

**Section(i)**

Let us try observing performance differences between a compiler optimized code and the non-optimized version of the same.

* Go back to Windows and extract the binary files **math\_unopt** and **math\_opt** from lab2.zip and place them inside your drive **X:** -> **cpre381** -> **lab2**. Now go back to Putty.

Make sure you are in the **simplesim-3.0** directory (use **pwd** command to check this).

Run the above two binaries using **sim-profile**like you did for *sim-fast* (but replace “*tests-pisa/bin.little/test-math”* with “*/home/<username>/cpre381/lab2/*math\_unopt”.

./sim-profile /home/<username>/cpre381/lab2/math\_unopt

Do the same for **math\_opt**.

**math\_unopt** is precompiled with optimization level 0 (no optimization) using gcc. **math\_opt** is precompiled with optimization level 3 (max optimization) using gcc.

Compare #instructions using the table below. Also find the % reduction in #instructions.

|  |  |  |
| --- | --- | --- |
| Program | Optimization level 0 number of instructions | Optimization level 3 number of instructions |
| Math\_unopt/math\_opt | 2813753 | 1013734 |

Percent in Reduction is 36%

**Section(ii)**

Write a selection sort algorithm (or any sorting algorithm you are familiar with) in C that sorts 50 random integers and save it in your **lab2** folder. Now compile it with 0 optimization using the following commands:

cd /usr/local/ss

./bin/sslittle-na-sstrix-gcc –O0 –o /home/<username>/cpre381/lab2/<filename> <space> /home/<username>/cpre381/lab2/<filename>.c

Next compile using level 3 optimization

./bin/sslittle-na-sstrix-gcc –O3 –o /home/<username>/cpre381/lab2/<filename> <space> /home/<username>/cpre381/lab2/<filename>.c

Compare #instructions using the table below. Also find the % reduction in #instructions. Submit your code as well.

|  |  |  |
| --- | --- | --- |
| Program | Optimization level 0 number of instructions | Optimization level 3 number of instructions |
| Sort.c | 36855 | 20114 |

Percent in Reduction 54%

**Part 3: Hardware Configuration**

**Section (i):**

In this section, we are going to find the best simulator configuration for a given program. Download the file **hardware** and **hardware.c** into **Lab2**. In Putty, navigate to /usr/local/ss (use `pwd` to verify) and run the following (all one line):

./simplesim-3.0/sim-outorder –dumpconfig <space> /home/<username>/*cpre381*/*lab2*/config\_out.txt <space> /home/<username>/*cpre381*/*lab2*/hardware

This creates a configuration file (config\_out.txt) for you that you can edit.

* Open up the config\_out.txt file.
* Locate the line **run pipeline with in-order issue** and change it value to **true** (out of order issue is beyond the scope of this class).
* Towards the end of the file, you will find the number of integer and floating point ALUs/multipliers/dividers defined. Make the values for the ALUs, multipliers, dividers equal to **1**, save the file and run the following:

./simplesim-3.0/sim-outorder –config <space>

/home/<username>/*cpre381*/*lab2*/config\_out.txt <space>

/home/<username>/*cpre381*/*lab2*/hardware

Report the CPI and # instructions that you get.

Number of Instructions : 420558 CPI: 1.8995

**Section (ii):**

From your understanding of the code in **hardware.c**, make appropriate changes to the number of functional units (ALUs, Multipliers etc.) in **config\_out** file( do not go beyond 8) and report the best CPI (which is an improvement over (i), if any).

To simulate with the modified config\_out.txt file, use the following:

./simplesim-3.0/sim-outorder –config <space>

/home/<username>/*cpre381*/*lab2*/config\_out.txt <space>

/home/<username>/*cpre381*/*lab2*/hardware

This lab should have given you some basic idea about what machine code looks like and the Simplescalar simulator. Feel free to play around with the simulator. Good luck!

**Submission:**

All submissions are through Blackboard. Write your answers in the report file. Submit your modified file(s) (config\_out.txt) as well, along with your report.

**Optional Readings:**

|  |  |
| --- | --- |
| 1. C programming basics: | <http://www.tutorialspoint.com/cprogramming/index.htm> |
| 2. Linux terminal basics: | <http://linuxcommand.org/learning_the_shell.php> |
| 3. The simulator: | [www.**simplescalar**.com/](http://www.simplescalar.com/) |