ECE4100/6100 CS4290/6290 - Fall 2024
Lab 1: Analyzing Benchmark Traces & Computing CPI

Prof. Moinuddin Qureshi
Version: 1.0

Due: August 23th 2024 @ 3:59 PM ET

Changelog

- Version 1.0 (8/19/2024): Initial release.

1 Rules

- **This is an INDIVIDUAL assignment.** You may discuss this assignment with classmates, but you should code your assignment individually (i.e. no code sharing). **All honor code violations will be reported to the Dean of Students. NO EXCEPTIONS.**
- These lab assignments are very difficult and take a lot of time. It is in your best interest to start early.
- See the "Late Policy for Assignments" Canvas announcement for the course's late policy.
- Please utilize TA office hours, Piazza, and recitation if you have questions. **Do not go to the professor's office hours with lab-specific questions.**
- Read the entire document before starting. Not only is it critical to understanding the assignment, but most questions can be answered by reading the corresponding section.
- Sometimes, mistakes will be found in the PDF and the lab assignment. **It is solely your responsibility to ensure you are using the most up-to-date PDF and lab files.**
- Make sure that your code works with C++11 and on the provided reference machines as this what we will run the autograder on.

2 Introduction

The objective of this lab is to test the proficiency of students in doing programming-based assignments and to ensure that the students have the basic background in computer architecture to take this graduate level course. The assignment is due the day of the Phase II registration deadline (Friday), so that the students can make well-informed decisions whether they should continue with the course or should consider taking it after getting the required prerequisites.

3 Design

Your goal is to perform analysis of the static and dynamic occurrences of instructions in a given benchmark trace. The benchmark traces are a capture of what instructions the CPU performed during a single run of a given program. We will provide a code template and traces from four benchmarks: `gcc, mcf, libquantum,` and `bzip2`. These were selected from the SPEC CPU2006 suite and are the traces we will use in all subsequent lab assignments. We will also provide a trace reader for these traces, so you do not need to worry about any form of file I/O.

This lab has been split up into 3 separate tasks for you to complete. They are as follows:

- **Task 1:** Quantify the mix of the dynamic instruction stream. The instructions in the trace are classified as five types:
  - Arithmetic (`ALU`)
  - Load (`LD`)
  - Store (`ST`)
  - Conditional Branch (`CBR`)
  - Other (`OTHER`)

  You will modify the provided code to count the number of dynamic instructions for each category in the instruction mix.

- **Task 2:** Estimate the overall CPI using a simple CPI model in which the CPI for each instruction category is provided. The CPI for each category is as follows:
  - `ALU`: 1
  - `Load`: 2
  - `Store`: 2
  - `CBR`: 3
  - `Other`: 1

  Please note that the instruction mix for each trace is different, so the CPI for each trace will also be different.

- **Task 3:** Estimate the instruction footprint by counting the number of unique PCs in the benchmark trace. Ideally, this information should be multiplied by the average bytes per instruction to get the total footprint, however for this lab we will ignore this calculation.

4 Implementation Details

You have been provided the following files:

- `studentwork.cpp` - The main file for this project. The function `analyze_trace_record()` needs to be implemented for providing analysis functionality.
- `sim.cpp` - **Do not modify.** Responsible for opening the trace, initialization and instating, and executing the project.
- `trace.h` - **Do not modify.** Header file containing structs and definitions pertaining to instructions from the traces.
- `Makefile` - **Do not modify.** A Makefile for compiling your code. Can be used to invoked the following commands...
  - `make all`: Compiles your code and creates an output file so you can run your code.
  - `make clean`: Cleans out compiled files.
  - `make fast`: Compiles your code with the `-O2` flag. See this for details.
  - `make debug`: Compiles your code with the preprocessor definition DEBUG defined. This causes code blocks of `#ifdef DEBUG ... #endif` to compile. **Highly recommend using for debugging purposes!**
  - `make profile`: Compiles your code for use with `gprof`. Helps diagnose bottlenecks if your code is running slowly.
  - `make validate`: Compiles your code and runs `runall.sh`.
  - `make submit`: Creates a tarball for your submission. Run `make validate` beforehand to generate `report.txt`.
- `traces/` - **Do not modify.** A directory containing the execution traces for this project.
- `scripts/` - **Do not modify.** A directory containing the following scripts...
  - `runall.sh`: Runs your code on `bzip2, gcc, libq,` and `mcf`. The results of each run is placed in the corresponding `.res` file in `results/`. A summary of all the results can be found in `report.txt`, which is placed in whichever directory `runall.sh` is ran from (e.g. if you call `make runall` the report will be put in `src/`).
- `results/` - **Do not modify.** A directory containing the output of your code from `runall.sh`.

4.1 Simulator Statistics

The simulator keeps track of a variety of statistics, some of which you may have to update:

- `stat_num_inst`: Counts the number of instructions read by the trade reader.
- `stat_optype_dyn`: An array which counts the number of instructions executed by the simulator for each instruction category.
- `stat_num_cycle`: Counts the number of CPU cycles.
- `stat_unique_pc`: Counts the number of unique PCs seen by your simulator.
- `PERC_ALU_OP` : The percentage of `ALU` instructions read by the simulator.
- `PERC_LD_OP`: The percentage of `LD` instructions read by the simulator.
- `PERC_ST_OP`: The percentage of `ST` instructions read by the simulator.
- `PERC_CBR_OP`: The percentage of `CBR` instructions read by the simulator.
- `PERC_OTHER_OP`: The percentage of `OTHER` instructions read by the simulator.
- `CPI`: Defined as `stat_num_cycle` / `stat_num_inst`

5 Deliverables

To prepare your submission, run `make submit` to generate a tarball (`tar.gz`) with your code. **Do NOT change the name of the "src" directory**, as the autograder will not be able to test your submission. Make sure to untar your submission to ensure all the required files are there before submitting. **You are solely responsible for what you submit.** If you submit improperly, you can face up to a 10% grade deduction. Please do not make us use this rule and submit properly.

To see if you can receive full credit, you should run `make validate` and ensure you are passing. Note that just because you pass `make validate`, this does **NOT** mean you get a 100%. We reserve the right to examine any submission if we suspect plagiarism or attempts to game the autograder in any way.

6 Grading

For both parts in total, students from all sections (ECE4100/6100, CS4290/6290) are graded out of 5 points. Note that this project is 5% of your final grade, meaning that each point equates to one point towards your final grade.

Your submission is evaluated based on how your simulator's CPI compares to the reference CPI for `bzip2, gcc, libq,` and `mcf`. The following shows the grade breakdown for each trace:

- `bzip2` (1.25 points)
  - **CPI**: 1.25 points if CPI matches, 0 otherwise
- `gcc` (1.25 points)
  - **CPI**: 1.25 points if CPI matches, 0 otherwise
- `libq` (1.25 points)
  - **CPI**: 1.25 points if CPI matches, 0 otherwise
- `mcf` (1.25 points)
  - **CPI**: 1.25 points if CPI matches, 0 otherwise

While there is no strict efficiency requirement, please ensure your simulator finishes in less than a minute. This is to ensure expedient grading, since otherwise verification would be difficult and tedious. If your simulator is taking too long, we recommend you profile your code to see where most your code spends its time running. You can use `make profile` to allow your code to be used with `gprof`.

7 Reference Machines

The autograder will run on a reference machine to ensure that everyone conforms to one standard. There are two reference machines you can use for the course:

- **ECE**: ece-linlabsrv01.ece.gatech.edu (email)
- **CS**: Any general-purpose servers here.

How to access these machines depends on whether or not your are using Georgia Tech Wi-Fi (`eduroam`). If you are not, then you need to install the Georgia Tech VPN client (see Remote Access → VPN or Virtual Private Networking). Once you are connected either via VPN or Wi-Fi, you should be able to `ssh` into one of the above servers via the command line of your operating system like so:

```
ssh gburdell3@ece-linlabsrv01.ece.gatech.edu
```

Please note that `gburdell3` should be replaced with *your* GT username. If you receive `Permission denied` or `Access denied` when trying to connect, then use the above links to request access. Please include your GT username, the course name (ECE 4100/6100), and the instructor in the request.

We highly recommend that both CS and ECE students obtain access to each other's servers as these servers have been shown to go down at times. You can contact the respective help desks to request access. Since the servers can go down, it is crucial that you do not exclusively work on the servers while working on these projects. We recommend working on your code locally using services like WSL that will simulate a Linux environment for you to work and verify your implementation in.

8 FAQ

1. **I get an error when I try to execute the runall.sh script or the runtests.sh script.**
   Check that you have execute permissions on both scripts. Add execute permissions to these files using the command: `chmod +x runall.sh runtests.sh`

2. **I get an error `unrecognized command-line option "-std=c++11"` when compiling on `shuttle3.cc.gatech.edu`.**
   In the Makefile, remove the `-std=c++11` flag on the `CXXFLAGS` line. This will allow your code to compile if you avoid using any C++11 features. Don't worry about submitting the modified Makefile. If needed, we will apply this modification when grading without additional action necessary on your part.

3. **What do I do if I get a `no space left on device` error?**
   You can run `du -sh *` to see what is taking up space in your current directory and use `rm` and `rm -r` to remove files and directories respectively. If you are running `./runall.sh`, ensure that you have cleared out the `results/` directory as the files here can be very large if you ran `./runall.sh` with your print statements.

Appendix - Plagiarism

Preamble: The goal of all assignments in this course is for you to learn. Learning requires thought and hard work. Copying answers thus prevents learning. More importantly, it gives an unfair advantage over students who do the work and follow the rules.

1. As a Georgia Tech student, you have read and agreed to the Georgia Tech Honor Code. The Honor Code defines Academic Misconduct as "any act that does or could improperly distort Student grades or other Student academic records."

2. You must submit an assignment or project as your own work. Absolutely no collaboration on answers is permitted. Absolutely no code or answers may be copied from others — such copying is Academic Misconduct. NOTE: Debugging someone else's code is (inappropriate) collaboration.

3. Using code from GitHub, via Googling, from Stack Overflow, etc., is Academic Misconduct (Honor Code: Academic Misconduct includes "*submission of material that is wholly or substantially identical to that created or published by another person*").

4. Publishing your assignments on public repositories accessible to other students is unauthorized collaboration and thus Academic Misconduct.

5. Suspected Academic Misconduct will be reported to the Division of Student Life Office of Student Integrity. It will be prosecuted to the full extent of Institute policies.

6. Students suspected of Academic Misconduct are informed at the end of the semester. Suspects receive an Incomplete final grade until the issue is resolved.

7. We use accepted forensic techniques to determine whether there is copying of a coding assignment.

8. If you are not sure about any aspect of this policy, please ask Dr. Mahajan