CS4202 Computer Architecture 2023-24

Practical 2 - Branch Predictor

1 Introduction

This coursework is worth 50% of the practical component of this module.

Submission Deadline: Friday 5th April 2024, Week 11

1.1 Main Objectives

- To gain practical experience in simulating aspects of computer architecture.
- To understand the importance (or otherwise) of branch prediction strategies for particular programs.
- To reason about the effect of branch prediction on particular programs.
- To develop and demonstrate scientific experimentation and analysis techniques.

2 Task

Your task is to develop a **branch predictor simulator** that simulates various branch prediction strategies, ultimately calculating the misprediction rate for a given algorithm. Your program will read in some pre-generated *dynamic branch trace* files, and make a prediction for each branch.

You may use any *reasonable mainstream programming language* to implement your simulator. Suggestions include (but are not limited to):

- C
- C++
- Rust
- Python
- Java

You should also use this coursework as an opportunity to improve your coding skills!

Note: your code *must* run in the lab machine environment.

You will then *analyse your results* and present them in a *report* (up to 10 pages, excluding the bibliography and any appendices). Credit will be given for rigorous scientific experimentation, and insightful analysis.

Select a subset of the provided trace files to test your simulator, and justify your choices. Not all single programs or combinations are a good choice. You must decide what to use.

You should provide results for the following algorithms:

- Always taken
- Standard 2-bit predictor
  - Use a BP table of sizes: 512, 1024, 2048, and 4096 entries
- gshare
- Profiled

2.1 Profiled Approach

For the purposes of this practical, a profiled approach is a BP strategy where you may use information gathered from a "profile" run of the code before it is run or measured in a "real world" environment. You should think about how such information might be useful, and how it could be represented in architecture.

Be sure to explain clearly how your profiled approach works in the report, including any changes or additions you make to the hardware or toolchain.

2.2 Deliverables

Submit (via MMS) a ZIP file containing your *program source code*, and your *report as a PDF*. You should also provide a `README` file to show how to compile and run your program.

Your report should be *no more than 10 pages*, on A4 paper with a 12pt font. This is a **compulsory** page limit, and excludes the bibliography (which can be any number of pages).

You may also include any number of additional pages of appendices, if you wish–although you are not required to. Note that it should be possible to mark your report without looking at any of the appendices at all—you *must not* rely on them for *additional space*. They are optional, and you should include them only if you want to present some interesting data, not directly related to the main understanding of your work.

The compulsory page limit applying to the main body of text is subject to the penalties described in the student handbook:

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#word-count-penalties`

3 Trace Files

You are provided with a number of *trace files* for various programs (find them on `studres`). Each file contains the *dynamic branch trace* for a particular program. Trace files can be arbitrarily large, so do not try to analyse huge trace files – you can cut them to a reasonable length. When doing this cutting, think carefully about what to cut, and explain your thinking in your report.

A dynamic branch trace is simply a list of addresses of all the branch instructions that are encountered in a run of a program, along with if they were taken or not. The trace file format is structured as follows:

```
<program counter address> <target address> <branch kind> <direct>
                    <conditional> <taken>
```

Where branch kind may be one of `b`, `c`, or `r` corresponding to branch, call, and return kinds respectively. The last three values are binary values corresponding to whether the branch was direct, conditional, and taken.

An excerpt from a trace file is provided below. The first line shows that a branch at program counter `00007ff6ef77d2b3` with target `00007ff6ef77e050` was a direct, unconditional, taken call.

```
00007ff6ef77d2b3 00007ff6ef77e050 c 1 0 1
00007ff6ef77e0a9 00007ff6ef77e148 b 1 1 0
00007ff6ef77e0de 00007ff6ef77e0fa b 1 0 1
00007ff6ef77e0a9 00007ff6ef77e0e9 b 1 1 1
00007ff6ef77e0f8 00007ff6ef77e148 b 1 1 0
```

Note that the branch at address `00007ff6ef77e0a9` appears twice. This is a *dynamic* trace of a program, so every time a branch is encountered, it will appear in the file—of course, the outcome of the branch can be different throughout the execution (e.g. loops).

Branches appear chronologically as they were executed. These traces are **ground truths** of what branches were taken, not a prediction. Your simulator needs to make a prediction, and compare it to the actual operation.

4 Marking

See the standard mark descriptors in the School Student Handbook:

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#Mark_Descriptors`

You will be assessed on the quality of your report, and the design of your experiments. Important issues include which benchmarks you choose, how you measure your results, how you display your results, your critical analysis of the results, and your explanation of how the experiments were conducted, along with the rationale behind them.

You will not be assessed on the quality of your software, in terms of the quality of engineering

– although take this as an opportunity to improve your coding skills! What's important is that the software must be able to produce the results reported, and I can run it!

A *passing submission* will contain the implementation of most of the required strategies, and a report with a basic description of experimental methodology, and evidence experimentation being performed by reporting results. A level of basic analysis of the results is expected.

A *good submission* will contain, in addition, some insight into the results, further than simply a description of the numbers. The methodology will be clear, and the experimental design will be sound. The results will be presented appropriately (this is likely to include graphs, which are designed, and labelled correctly). A reasonable attempt at a profiled prediction strategy is expected. A high quality of writing is expected.

An *excellent submission* will contain, in addition, an experimental design and description such that a reader could re-implement your experiments purely by using the report. The experimental design will demonstrate an understanding of the task at hand, appropriate benchmarks chosen, and sensible measurement strategies that are designed to illuminate the differences in BP strategies. There will be detailed analysis of the results, presented in an appropriate manner.

4.1 Lateness

The standard penalty for late submission applies (Scheme A: 1 mark per 24 hour period, or part thereof):

`http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#latenesspenalties`

4.2 Good Academic Practice

The University policy on *Good Academic Practice* applies: `https://www.st-andrews.ac.uk/students/rules/academicpractice/`