

Compiler Optimisation

2 – Coursework

Hugh Leather

IF 1.18a

hleather@inf.ed.ac.uk

Institute for Computing Systems Architecture

School of Informatics

University of Edinburgh

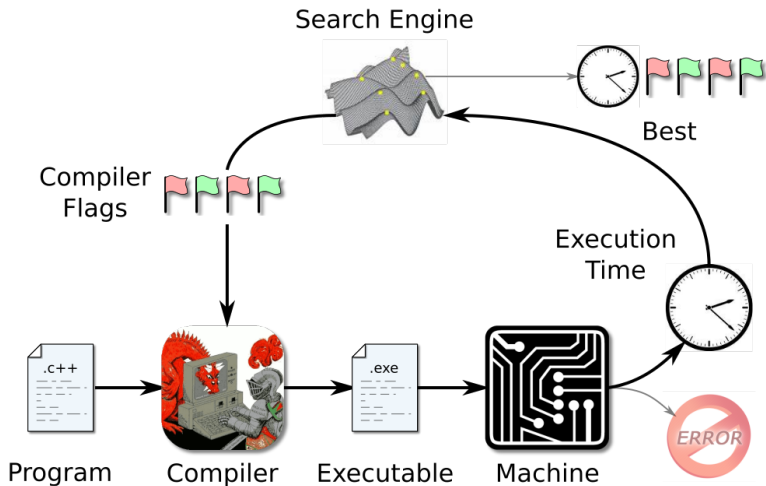
2019

Course work

- Based on GCC compiler
- One piece of course work: 25% of course mark
- Set today and due **Thursday 4pm Feb 21st 2019** week 6
- Feedback due **Thursday 4pm Mar 7th 2019** week 8
- Penalties for late submission.
- Plagiarism software used. Do your own work!

Iterative Compilation

Find the best way to compile a program



Goal

- Evaluate different compiler optimisation settings on a set of benchmarks.
- Try to beat -O3
- Write a report about your methodology and your findings.

Program Optimisation in GCC

- GCC supports some simple levels of optimisations:
-O1, -O2, -O3
- At each level, a set of optimisations are enabled
(25 for O1, 25+29 for O2 and 19+28+9 for O3)
- At higher levels, more optimisations are enabled which results in **potentially**¹ faster code, but also slows down the compilation process.
- Rather than using these pre-defined optimisation options, the users can enable individual options themselves, e.g. “-funroll-loops”.
- For more information on optimisation options see <http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>

¹Not all optimisations make code better

Methodology: Evaluating Compiler Flags

- Always use `-O3`: Some optimisations won't work without it
- Randomly choose flags (on/off) and parameter values
`http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`
- Evaluate 200 randomly chosen configurations
(i.e. combinations of optimisations)
- Use the same configurations for all benchmarks!

Running Experiments

- Avoid noise:
 - Make sure no one else is logged on to the computer (using who) and no other applications are running (using top).
 - Dont run on top of AFS \Rightarrow use /disk/scratch or /tmp.
 - BUT: move the results back to your home-directory and dont leave the data accessible to everyone
- Run benchmarks at least 10 times to get stable results.
 - Determine how many measurements you need to get a stable value.
 - Compute and report *average* runtime.
 - Also report the *variance* and the number of iterations you used.

Running Experiments - Cont.

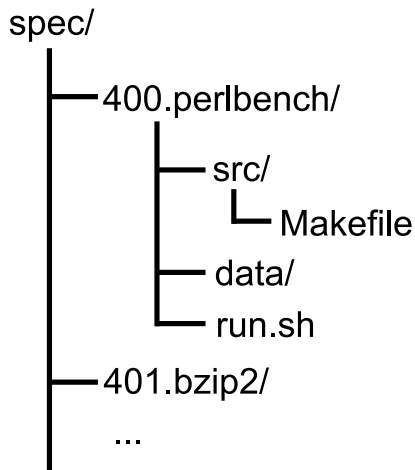
- Use scripting languages to automate the process of evaluating optimisations on the benchmark programs.
- Example (pseudo code)

```
for each b in benchmarks
  for each o in optimisations
    compile b with o
    run b N times and record runtimes
    calculate average runtime and variance
  end
end
end
```


The Benchmarks

- We use 14 benchmarks from the SPEC CPU2006 and MediaBench II suites.
- CPU intensive benchmarks developed from real user applications.
- Download and extract the programs (use wget) from:
<https://docs.google.com/file/d/0B5GasM1WJhT0aTdvaFkzUzNobDQ/edit>
- Let me know if you need more disk space!

Directory Structure



Compiling and Running the Benchmarks

- Compiling a program with certain optimisations enabled and executing it a single time:

```
cd 400.perlbench/src/  
make CFLAGS="-funroll-loops -param max-unroll-times=4"  
cd ../  
./run.sh
```

Report and Results

- Maximum 5 pages + 2 pages for results
- Explain what you have done.
- Precisely describe the experimental setup.
 - Architecture and platform. Timing method.
 - Number of runs per benchmark/configuration
- For every program report performance of:
 - Baseline -00, -01, -02, -03
 - Best found flags for individual program.
 - Best found single set of flags across *all* programs.
 - Average across all flag settings (expected random performance).
- Results should be detailed: per-program, average, variance

Report and Results - contd.

- Store all raw data in a file. For each program:
 - First line: program name
 - Following lines: flag setting and all runtimes
 - Runtimes in milliseconds, without decimal digits

```
400.perlbench
```

```
"-00" 837 833 890 850 813 828 ...
```

```
"-01" 602 620 610 611 650 580 ...
```

```
...
```

```
401.bzip2
```

```
"-00" 837 833 890 850 813 828 ...
```

```
"-01" 602 620 610 611 650 580 ...
```

```
...
```

- e-mail file to: hleather@inf.ed.ac.uk WITH the subject:
copt-results

Report Structure

- Abstract. (Summary of paper) and Introduction
- Evaluation methodology: Selection of flags, etc.
- Experimental setup: Platform. How time was measured. Number of runs.
- Results (for each program)
 - Baseline -00, -01, -02, -03
 - Best found flags for individual program.
 - Best found single set of flags across all programs.
 - Average across all flag settings (expected random performance).
- Analysis and Discussion of Results. Followed by conclusion.

Submission. Awarding of Marks

- Submit to ITO written report by Thursday 4pm Feb 21st 2019.
- Marks are awarded for clear explanation of experimental methodology and thorough analysis of results.
- Remember wish to see optimisation setting that gives best results per program AND the setting that is best for all the benchmarks.

Final Remarks

- For further questions e-mail me
- Start early!! It takes time to run the experiments!
- Deadline: **Thursday 4pm Feb 21st 2019**

PPar CDT Advert

EPSRC Centre for Doctoral Training in Pervasive Parallelism

- 4-year programme:
MSc by Research + PhD
- Research-focused:
Work on your thesis topic
from the start
- Collaboration between:
 - ▶ University of Edinburgh's
School of Informatics
 - * Ranked top in the UK by
2014 REF
 - ▶ Edinburgh Parallel Computing
Centre
 - * UK's largest supercomputing
centre
- Research topics in software,
hardware, theory and
application of:
 - ▶ Parallelism
 - ▶ Concurrency
 - ▶ Distribution
- Full funding available
- Industrial engagement
programme includes
internships at leading
companies

The biggest revolution
in the technological
landscape for fifty years

Now accepting applications!
Find out more and apply at:
pervasiveparallelism.inf.ed.ac.uk



THE UNIVERSITY of EDINBURGH
informatics

EPSRC

Engineering and Physical Sciences
Research Council