332 Advanced Computer Architecture

Exercise (ASSESSED): Pagerank Challenge

This is the second of two equally-weighted assessed coursework exercises. **You may work in groups of two or three if you wish, but your report must include an explicit statement of who did what.** Submit your work in a pdf file electronically via CATE.[1] The CATE system will also indicate the deadline for this exercise.

Background

Pagerank is the famous algorithm that enabled Google's search engine to become overwhelmingly dominant. Pagerank captures the simple idea that more important websites are likely to receive more links from other websites. We will be studying a simple implementation of the algorithm, derived from a code example for Louridas' algorithms book - see https://github.com/louridas/pagerank. The book is here, https://mitpress.mit.edu/books/real-world-algorithms.

Your task is to identify the critical factors influencing the performance of this code, and to improve its performance by whatever means necessary, including modifying the code. You are encouraged to use whatever tools you can find, such as profilers and compilers. You may choose any hardware platform you wish, though you should discuss exotic choices in advance. Assessment is based on the quality of the investigation as presented in your report, with particular attention paid to structuring the investigation around clearly-stated hypotheses, and the presentation of experimental results.

Compiling and running the Pagerank algorithm

Copy the source code directory tree to your own directory:

```
cd
cp -r /homes/phjk/ToyPrograms/ACA18/Ex2LouridasPagerank ./
```

Now compile the benchmark program:

```
cd Ex2LouridasPagerank/cpp
make clean
make
```

Now you can test the program:

```
make all-tests
```

This runs the program on a large variety of different graphs; you should get an "OK" for all of them. To run a test that takes a significant amount of time, try:

[1] https://cate.doc.ic.ac.uk/

```
make large-test
```

This should finish in a few seconds at the most, and again it should confirm that the result was correct with "OK". The execution time for the pagerank algorithm itself is given by the line "Pagerank time", which is what we are aiming to minimise (the program as a whole takes considerably longer due to reading in the input, and checking the results).

You should be able to use any Linux machine; MacOS is probably also fine.

All-out performance

Basically, your job is to figure out how to run this program as fast as you possibly can, on a hardware platform of your choosing, and to write a brief report explaining how you did it.

Choosing a problem size and data set

I have provided you with a a range of different problem sizes. You should select a problem size appropriate for the level of performance you are achieving - so a small low-power device can be exercised properly with a smaller dataset than a larger parallel machine. You need the accelerated runtime to be at least a few seconds.

The smaller datasets are included in the directory tree that you have copied. The larger ones (see the files `large`, `enormous`, `ginormous`) are accessed from a shared copy of the data in my home directory. If you want to run large experiments on your own machine, you will need to make a copy of your own (the data is really quite big).

The datasets provided for performance evaluation (small, medium, large etc) are artificial random graphs generated using the Barabasi-Albert model (https://en.wikipedia.org/wiki/Barab%C3%A1si%E2%80%93Albert_model). This approach to generating random graphs aims to reproduce statistical properties of real citation graphs, leading to a scale-free structure with some vertices much more highly-connected than others.

Validating correct execution

Check for *every* experiment that the results are validated correctly (ie the program prints "OK"). If you make changes to the source code, you should also check that `make all-tests` validates correctly.

Rules

1. You can choose any hardware platform you wish. You are encouraged to find interesting and diverse machines to experiment with. The goal is high performance on your chosen platform, so it is OK to choose an interesting machine even if it's not the fastest available. On Linux, type `cat /proc/cpuinfo` for details of the processor and clock rate, which you should document carefully in your report.

You need to recognise the limits of what is possible within the timeframe available — using GPUs is very ambitious, while using FPGAs would be beyond heroic.

2. Make sure that the results are correct *every* time you run the program. Do not disable this, and do not report results if the validation fails. If you think the validation is too strict, ask.
3. Make sure the machine is quiescent before doing timing experiments. Always repeat experiments for statistical significance.
4. Always script your experiments, so that it is easy to check how each data point was generated, and it's easy to rerun your experiments when you change something.
5. Choose a problem size and number of repeats which suits the performance of the machine you choose - the runtime must be large enough for any improvements to be reliably measured.
6. Take care to report precisely and fully the details of the hardware (CPU part number, clock rate) and software (compiler version numbers, operating system and kernel version etc).
7. If you use a laptop, make sure it's plugged into mains power, as some mobile platforms can adapt and mess up your experiments.
8. You can achieve full marks even if you do not achieve the maximum performance.
9. Marks are awarded for
    - Systematic analysis of the application's behaviour
    - Systematic evaluation of performance improvement hypotheses
    - Drawing conclusions from your experience
    - A professional, well-presented report detailing the results of your work.
10. You should produce a report in the style of an academic paper for presentation at an international conference or journal. The report should be not more than seven pages in length. An example of the kind of format and structure we are looking for is:

    > Bartosz D. Wozniak, Freddie D. Witherden, Francis P. Russell, Peter E. Vincent, Paul H. J. Kelly: GiMMiK - Generating bespoke matrix multiplication kernels for accelerators: Application to high-order Computational Fluid Dynamics. Computer Physics Communications 202: 12-22 (2016) https://doi.org/10.1016/j.cpc.2015.12.012.

Changing the rules

If you want to bend any of these rules just ask.

Performance analysis tools: You may find it useful to find out about:

- Intel's VTune - tool (Windows, Linux, MacOS) for understanding CPU performance issues and mapping them back to source code (free trial). http://www.intel.com/software/products/vtune. This is also installed on CSG's Linux systems

```
source /vol/intel/parallel_studio_xe_2017.0.035/vtune_amplifier_xe/amplxe-vars.csh
/vol/intel/parallel_studio_xe_2017.0.035/vtune_amplifier_xe/bin64/amplxe-gui
```

(the first line sets up environment variables to enable the tool to run).

- cachegrind and cg_annotate
- kcachegrind - graphical interface to cachegrind: http://kcachegrind.sourceforge.net
- oprofile: http://oprofile.sourceforge.net - performance profiling by sampling hardware performance counters. Should install with apt-get on Ubuntu; works nicely with kcachegrind.
- Likwid-perfctr (http://code.google.com/p/likwid/wiki/LikwidPerfCtr) - handy performance counter tools that you can easily install on your own (non-virtual) Linux system.
- AMD's μprof (https://developer.amd.com/amd-uprof/)
- Apple's XCode "Instruments" performance tools
- OpenSpeedshop for Linux: http://www.openspeedshop.org/

Compilers You could investigate the benefits of a more sophisticated compiler:

- Intel's compilers. You should be able to download student and evaluation copies for your own machines from https://software.intel.com/en-us/parallel-studio-xe/choose-download.

Source code modifications

You are strongly invited to modify the source code to investigate performance optimisation opportunities.

How to finish The main criterion for assessment is this: you should have a reasonably sensible hypothesis for how to improve performance, and you should evaluate your hypothesis in a systematic way, using experiments together, if possible, with analysis.

What to hand in Hand in a concise report which

- Explains what hardware and software you used,
- What hypothesis (or hypotheses) you investigated,
- How you evaluated what the potential advantage could be,
- How you explored the effectiveness of the approach experimentally
- What conclusions can you draw from your work
- If you worked in a group, indicate who was responsible for what.

Please do not write more than seven pages.

For the submission, CATE will ask you for two things: a pdf report, and a tar or zip. We will be marking the report, not the tar/zip.

We do not promise to read the tar/zip, and it is not part of the marking scheme - so please do not let it add to your stress. However it is often helpful in appreciating what you have done to be able to see your code, and the data from which the tables and graphs in the report were derived.

Paul Kelly, Imperial College London, 2018