332 Advanced Computer Architecture

Exercise (ASSESSED): The 3D Stencil 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

This exercise concerns the same 3D stencil problem that we studied in simulation in Exercise 4. The goal this time is to achieve high performance on real hardware.

Running the DEM benchmark

Getting the benchmark code

Copy the benchmark code to your own directory, e.g.

```
prompt> mkdir /homes/yourid/ACA12 (you will probably have already done this)
prompt> cd /homes/yourid/ACA11
prompt> cp -r /homes/phjk/ToyPrograms/ACA12/StencilChallenge ./
```

(The ./ above is the destination of the copy – your current working directory).

List the contents of the benchmark directory:

```
prompt> cd StencilChallenge
prompt> ls
common.h     main.test.c  probe_heat_blocked.c  probe_heat_timeskew.c
cycle.h     Makefile  probe_heat.c  run.h
find_best_block.sh  min.pl  probe_heat_circqueue.c  util.c
main.c     probe  probe_heat_oblivious.c  util.h
```

To compile and run the code:

```
prompt> make clean; make probe ; ./probe 602 602 602 0 0 0 5
```

This builds the most straightforward implementation of the program. You may wish to explore some of the optimised versions:

```
prompt> make clean; make circqueue_probe ; ./probe 602 602 602 75 75 75 5
prompt> make clean ; make oblivious_probe ; ./probe 602 602 602 0 0 0 5
```

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

The parameters of the probe program are as follows:

```
./probe <gridx> <gridy> <gridz> <tilex> <tiley> <tilez> <iterations>
```

In the simplest version, the tile size parameters are ignored. The more sophisticated versions use them in various ways, and they may have to divide into the grid size (-2) exactly.

All-out performance

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

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`.

   You are encouraged to look at graphics processors, mobile devices, etc. If in doubt please ask.

2. Make sure that the results are correct *every* time you run the program. The program prints a checksum - check this matches the "make probe" version for the specific problem size and number of iterations that you ran.

3. Make sure the machine is quiescent before doing timing experiments. Always repeat experiments for statistical significance (see `NUM_TRIALS` in "run.h").

4. Choose a problem size (`gridx, gridy, gridz, iterations`) which suits the performance of the machine you choose - the runtime must be large enough for any improvements to be evident. You may find it interesting to increase the number of iterations; smaller numbers are less interesting.

5. You can achieve full marks even if you do not achieve the maximum performance.

6. 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.

7. You should produce a report in the style of an academic paper for presentation at an international conference such as Supercomputing.[2] The report should be not more than seven pages in length.

[2] http://sc12.supercomputing.org/

Changing the rules

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

Hints, tools and techniques

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

- 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.
- 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/
- AMD's CodeAnalyst (http://developer.amd.com/tools/codeanalyst/pages/default.aspx)
- Apple's XCode "Instruments" performance tools
- OpenSpeedshop for Linux: http://www.openspeedshop.org/wp/

Compilers You could investigate the potential benefits of using more sophisticated compilers:

- Intel's compilers: these are not installed on CSG linux systems but you should be able to download evaluation copies for your own machines from http://www.intel.com/software/products/compilers
- AMD's APP SDK for OpenCL on x86 CPUs and AMD GPUs: http://developer.amd.com/gpu/AMDAPPSDK
- NVIDIA's CUDA and OpenCL compilers: http://developer.nvidia.com/object/gpucomputing.html

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.

The 3D Stencil Probe benchmark

The benchmark code we are using comes from this website:

`http://www.cs.berkeley.edu/~skamil/projects/stencilprobe/`

It was developed by Shoaib Kamil at Berkeley. Some small changes have been made for the purposes of this exercise, notably the addition of the checksum correctness check.

Paul H.J. Kelly, Imperial College London, 2012