332 Advanced Computer Architecture

Exercise (ASSESSED): The Sum of Absolute Differences 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 the CATE system,[1] which will indicate the deadline for this exercise.

Background This exercise is about the same Sum of Absolute Differences benchmark, as we studied under simulation in the first assessed exercise. This time, however, the challenge is to make it go as fast as you can. You are *encouraged* to modify the source code – up to using different algorithms and data structures.

Working with the Parboil suite

Copy the benchmark code to your own directory:

```
prompt> mkdir /path/to/your/dir
prompt> cd !$
prompt> cp -r /homes/phjk/ToyPrograms/ACA09/SAD ./
prompt> cd SAD && ls
```

Make the Parboil suite's test harness:

```
prompt> cd common/src
prompt> make PARBOIL_ROOT=/absolute/path/to/your/dir
prompt> cd ../..
```

Compile and run the fast CPU version `cpu`:

```
prompt> ./parboil run sad cpu default
Parboil parallel benchmark suite, version 0.1

IO:      0.375603
GPU:     0.000000
Copy:    0.000000
Compute: 0.061069
Pass
```

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

Working with code You can start with copying this CPU version and modifying it, e.g.:

```
prompt> cp -r benchmarks/sad/src/cpu benchmarks/sad/src/mycpu
```

Compile and run `mycpu` similarly:

```
prompt> ./parboil run sad mycpu default
```

The Parboil's test harness should let you know if the obtained output mismatches the reference one.

Working with data `default` is the default data set of 176×144 input image frames. You may wish to scale the default data set for evaluating your version. For example, to add a scaled data set of 64×32 frames, type:

```
prompt> ./scripts/add_dataset 64 32
```

Each parameter must be an integral multiple of 16. This will create subdirectories `input/64x32`, `output/64x32` and `run/64x32` in `benchmarks/sad`, and place the scaled input frames into `input/64x32` and the reference output (from running the `cpu` version) into `output/64x32`.
To remove this data set, type:

```
prompt> ./scripts/rm_dataset 64 32
```

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. The goal is to reduce the compute time, e.g. as shown by Parboil:

   ```
   Compute: 0.061069
   ```

2. 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`.

   Try the Apple G5s, ICT supercomputer resources (Itaniums, Opterons), graphics co-processors (NVIDIA, ATI), PDAs, DSP processors, or FPGAs. Please ask if you would like a suggestion.

3. Make sure the output matches the one obtained from the `cpu` version.

4. Make sure the machine is quiescent before doing timing experiments. Always repeat experiments for statistical significance.

5. Choose a problem size which suits the performance of the machine you choose – the runtime must be large enough for an improvements to be evident. The really interesting problems are, of course, the long-running ones.

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

7. 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.

8. 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.

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 (may require kernel rebuild): `http://oprofile.sourceforge.net`
- 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`.
- AMD's CodeAnalyst (installed on CSG Athlon machines but you need to build the code using a native Windows compiler): Start→Programming→AMD
- Apple's Shark tools: `http://developer.apple.com/tools/shark_optimize.html`

[2] `http://sc08.supercomputing.org/?pg=papers.html`

- Sun's Performance Analyzer (if you have a Sun Sparc machine): `http://docs.sun.com/source/806-3562`
- OpenSpeedshop for Linux: `http://oss.sgi.com/openspeedshop`

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

- Intel's compilers: `http://www.intel.com/software/products/compilers`
- The Pathscale compiler: `http://www.pathscale.com/ekopath.html`
- Codeplay's compilers (free demo download?): `http://www.codeplay.com`
- IBM's compilers, for Cell (PlayStation 3) etc.

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.

Paul H.J. Kelly & Anton Lokhmotov, Imperial College London, 2009