332 Advanced Computer Architecture

Exercise 7 (Assessed): the Susan Challenge

This is the second of two assessed coursework exercises, both based on SUSAN edge detector.

You may work in groups of two or three if you wish, but your report must include an explicit statement of who did what.

Deadline: Monday 7th March. Submit your work electronically via CATE.

Compiling and running Susan

Copy the source code directory tree to your own directory:

```
cd
cp -r /homes/phjk/ToyPrograms/ACA05/Susan ./
```

Now compile the susan.c program:

```
cd Susan
make
```

Now you can run the program:

```
time ./susan.x86 Ins/input_Cupboard.pgm Outs/Cupboard.pgm -e
```

This reads input from the file `Ins/input_large.pgm`, and writes its output to the `Outs` directory. Both input and output files are in PGM (Portable Gray Map) format (see "`man pgm`". The "time" command runs the command line and reports how much elapsed, user and system time it took.

You have been provided with a selection of input files of various sizes:

```
input_small.pgm  (76x95 pixels; from Susan distribution)
input_large.pgm  (384x288 pixels; from Susan distribution)
Cupboard.pgm     (1760x1168)
ToySoldier.pgm   (3072x2048)
```

If you need a larger one, see /homes/phjk/ToyPrograms/ACA05/Susan_ExtraImages.

All-out performance

Basically, your job is to figure out how to run susan 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".

   Try the Apple G5s, possibly PDAs, DSP processors, graphics co-processor or FPGA.

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

3. 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. You might also enjoy working with video - if so you will need to set up the experiment. You are very welcome to explore Susan's performance in its other modes, "-s" for smoothing, "-c" for corners.

4. The numerical results reported by the application need not be absolutely identical, but if not you must justify the correctness of your results[1].

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 compact report in the style of an academic paper for presentation at an international conference such as Supercomputing (www.sc2000.org). The report must not be more than 7 pages in length.

Hints, tools and techniques

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

- AMD's CodeAnalyst (installed on CSG Athlon machines - Start→Programming→AMD) (if you have an AMD machine)[2].
- Cachegrind and cg_annotate
- gprof - standard command-line profiling tool.
- kprof - kprof.sourceforge.net - graphical interface to gprof
- kcachegrind - kcachegrind.sourceforge.net - graphical interface to cachegrind
- VTune - Intel's (Windows and Linux) tool for understanding CPU performance issues and mapping them back to source code
  (http://www.intel.com/software/products/vtune/). Free trial.
- Sun's Performance Analyzer
  http://docs.sun.com/source/806-3562/ (if you have a Sun Sparc machine)
- oprofile
  http://oprofile.sourceforge.net/news/ (requires kernel rebuild)

[1] The gcc flag -ffloat-store is sometimes useful to check whether the difference in output is due purely to register allocation.

[2] To use this, use a Windows machine and copy the susan.x86.cygwin-gcc3-3-3.exe or susan-msvc-clv12.exe version of the code (and data).

Compilers You could investigate the potential benefits of more sophisticated compiler techniques:

- Intel's compilers (http://www.intel.com/software/products/compilers/ and installed on various Linux systems in the Department).
- Codeplay's compilers (www.codeplay.com) (free demo download?)
- IBM's compilers for Apple G5 - XL C/C++ Advanced Edition (a beta download was available, possible donation from Apple or IBM?)

Source code modifications You are strongly invited to modify the source code to investigate performance optimisation opportunities. You might wish to start with a cleaned-up version of the code, called "susan_arrays.c".

Susan's brightness lookup table ruins performance on some platforms, such as GPUs but you could try doing the calculation (or approximating it) in the loop. Note that most of the lookup table entries are zero (and many of the rest are 1).

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 Kelly, Imperial College, 2005