332 Advanced Computer Architecture

Exercise 8 (Assessed): the CFD Challenge

Background

In 1759 Leonhard Euler published equations of motion for a fluid, applying Newton's second law, i.e., the product of mass and acceleration of a body equals the external forces acting on it. Euler's idea to express knowledge about fluid dynamics in the form of partial differential equations meant a major breakthrough. A practical shortcoming of his flow model, however, was that it did not consider friction forces. In 1845 George Stokes introduced general equations, which in the case of an incompressible fluid reduce to equations already proposed by Claude Navier in 1822. These general equations are now called Navier-Stokes equations. Understanding and controlling a large class of fluid flows was reduced to the mathematical solution of a few basic differential equations. Unfortunately, analytical solutions are very rarely available. Numerical solution of the Navier-Stokes equations has developed into a major industry, and a vital support to diverse areas of engineering, medicine and science[1].

For further information see for example:

```
http://www.cfdreview.com/
http://www.cham.co.uk/
```

(The latter is an Imperial College spin-out company).

This exercise is based on a CFD application called NaSt3DGP, written by Michael Griebel, Roberto Croce, Frank Koster and Michael Meyer, based on the book Griebel/Dornseifer/Neunhoefer, Numerical Simulation in Fluid Dynamics, SIAM Philadelphia (1998). The code is freely available (although *not* open source), and can be downloaded from

```
http://wissrech.iam.uni-bonn.de/research/projects/koster/NaSt3DGP/)
```

You must abide by the licence published there. You are encouraged to visit their web site, which includes images and movies illustrating various applications of the tool.

All-out performance

This exercise can be done in groups of up to three, deadline Tuesday 4th March. Submit your work at the DoC Student General Office in a suitable labelled folder.

Basically, your job is to figure out how to run the computation specified in the next section as fast as you possibly can, and to write a brief report explaining how you did it.

[1] Historical background from http://www.cwi.nl/research/2001/Koren_Eng/.

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.
2. The numerical results reported by the application must be absolutely identical.
3. You can achieve full marks even if you do not achieve the maximum performance.
4. 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.
5. 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.

Tools and techniques

You may find it useful to find out about:

- Cachegrind and cg_annotate - see below
- gnuplot, a simple and useful graph plotting package for Linux and Windows
- 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/).
- Sun's Performance Analyzer http://docs.sun.com/source/806-3562/
- Intel's compilers (http://www.intel.com/software/products/compilers/ and installed on various Linux systems in the Department).
- PAPI, an open-source library providing access to diverse CPUs' hardware performance counters (http://icl.cs.utk.edu/projects/papi/overview/)
- The TaskGraph library; see /homes/phjk/SoftwareReleases/TaskGraphLibrary/Version0.1/
- Iterative Compilation in a Non-Linear Optimisation Space. PACT'98 F. Bodin, T. Kisuki, P.M.W. Knijnenburg, M.F.P. O'Boyle and E. Rohou: http://citeseer.nj.nec.com/bodin98iterative.html
- Tiling/blocking; Chapter 5 of the lecture notes, available online at `http://www.doc.ic.ac.uk/~phjk/AdvancedCompArchitecture/Lectures/`

- ATLAS and PhiPAC `http://acts.nersc.gov/atlas/main.html`
- AC/DC (Architecture-cognizant Divide-and-Conquer): `http://www.cs.ucsd.edu/~kgatlin/papers.html`

The application - getting started

Begin by making a copy of the files you will need:

```
cp -r /homes/phjk/ToyPrograms/Nast3dgp-2003 ./
cd Nast3dgp-2003
setenv NASTROOT `pwd`
```

The final line sets the environment variable NASTROOT to the current directory path. (if you use bash you will need to adjust this final line).

Next, build the application:

```
cd NaSt3DGP/src/
make clean
make serial
cd $NASTROOT
```

Now you can run the application. Various data files have been provided; start with a smallish one:

```
$NASTROOT/Scripts/RunNavcalc.sh Cavity2D-48x48x4x0.003
```

This command should produce (after some initial data) the following output:

```
Step:       1 Time: 0.000000 TimeStep: 0.00208333333333  Time:   0h  0m       0.00s
  it:       1  res: 2.753807e+00    itmax: 200
  it:      51  res: 4.851704e-02    itmax: 200
  it:     101  res: 1.143685e-03    itmax: 200
  it:     151  res: 1.100218e-05    itmax: 200
Iter:     200  Res: 1.63466965e-07     all It:      600
Mass Balance (rhs): -4.70586777e-16

Step:       2 Time: 0.002083 TimeStep: 0.00208333333333  Time:   0h  0m       1.00s
  it:       1  res: 1.907481e-02    itmax: 200
  it:      51  res: 5.579380e-04    itmax: 200
  it:     101  res: 3.942419e-06    itmax: 200
Iter:     138  Res: 5.59304312e-08     all It:     1014
```

Read the script "$NASTROOT/Scripts/RunNavcalc.sh" to see what this did. An input file "DataFiles/Cavity2D-32x32x4-0.003/p.nav" describes the 3d environment in which the fluid will flow, and other details of the simulation to be run. This is pre-processed by a helper program, "NaSt3DGP/bin/navsetup", to produce an initial state for the mesh on which the calculation is done, which is stored in "Junk/t.bin". The CFD solver itself, "NaSt3DGP/bin/navcalc", reads this, simulates the fluid flow for a specified number of timesteps, and writes the fluid state back to "Junk/t.bin".

You should concentrate on the Cavity problem, which is a standard CFD test example:

```
http://wissrech.iam.uni-bonn.de/research/projects/koster/NaSt3DGP/apl_cavity.htm
```

The fluid is in a box, whose lid is sliding to one side, dragging the top surface of the fluid sideways. Interesting fluid behaviour only begins to evolve after a fairly large number of timesteps. A variety of instances of this problem have been provided, of varying size and run-time, the smallest (above) being intended for use with simulators. In practice, problems of interest can involve 512^3 mesh elements or more, over hundreds of timesteps.

The second problem you might consider is more visually appealing:

```
./Scripts/RunNavcalc.sh FlowPastObstacle3D-51x51x13x0.4
```

For details see

```
http://wissrech.iam.uni-bonn.de/research/projects/koster/NaSt3DGP/mov_past_obstacle.htm
```

Validating your output It is essential that any optimised program computes exactly the same result. It is usually sufficient to check that the "residuals" (the "res:" column in the output listed above) match.

Using Cachegrind Try the following:

```
$NASTROOT/Scripts/RunNavcalc-cachegrind.sh Cavity2D-32x32x4-0.003
```

This runs the application under the CacheGrind simulator, then uses the "cg_annotate" tool to annotate the source code. The output can be found in the file "SourceCodeAnnotatedByCacheGrind".

Cachegrind annotates each line of code with the following events:

- Ir : I cache reads (ie. instructions executed)
- I1mr: I1 cache read misses
- I2mr: L2 cache instruction read misses
- Dr : D cache reads (ie. memory reads)
- D1mr: D1 cache read misses
- D2mr: L2 cache data read misses
- Dw : D cache writes (ie. memory writes)
- D1mw: D1 cache write misses
- D2mw: L2 cache data write misses

Note that D1 total accesses is given by D1mr + D1mw, and that L2 total accesses is given by I2mr + D2mr + D2mw. See the documentation (http://developer.kde.org/ sewardj/docs/manual.html) for details. You need to read the annotated listing carefully as some C++ functions are inlined (notably the array subscript operator "[]").

How to get started Here are some hints:

- Find out where the program spends most of its time.
- Does unrolling help? Try gcc's "-funroll-loops" flag.
- Consider using the Intel C++ compiler. Experiment with its flags, and read the information it (optionally) generates explaining what transformations are applicable.
- Where/when do cache misses occur?
- Can you improve the spatial locality?
- Can you improve the temporal locality?
- Have a look at the assembly code generated by the compiler (use the "`-S`" flag, "`gcc -O3 -S file.c`").
- Where are the likely pipeline stalls, missed instruction issue opportunities? How much scope for improvement is possible through improved instruction scheduling?

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, 2003