332 Advanced Computer Architecture

Exercise 7 (Assessed): the "YDL-PIJ" Challenge

This is the second of two assessed coursework exercises. This one is based on the "YDL-PIJ" computational chemistry application.

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 electronically via CATE.

Background This exercise is about "YDL-PIJ, a simplified quantum chemistry application, given to us by Mike Bearpark in Imperial's Chemistry department. The program was written in Fortran 77.

The methods implemented by the program are described in more detail in:

> **Excited states of conjugated hydrocarbon radicals using the molecular mechanics - valence bond (MMVB) method**. Bearpark MJ, Boggio-Pasqua M. *THEORETICAL CHEMISTRY ACCOUNTS 110 (2): 105-114 SEP 2003*. (http://dx.doi.org/10.1007/s00214-003-0461-3)

It's a very stripped-down model quantum chemistry application, which does two things:

1. For the specified number of electrons, all possible spin configurations are generated (each electron can have up or down 'spin'), together with any non-zero interactions between them.
2. These interactions are assembled into a matrix (Hamiltonian), which is diagonalised to obtain energy levels of ground and excited states of the system being modeled, along with the corresponding weighting coefficients of the individual electron configurations.

Much of a standard computational chemistry code (e.g. www.gaussian.com) is missing[1].

Compiling and running YDL-PIJ

Copy the source code directory tree to your own directory:

```
cd
cp -r /homes/phjk/ToyPrograms/ACA07/YDLPIJ/ ./
```

Now compile the program:

```
cd YDLPIJ
make
```

Now you can run the program:

```
time ./run.sh 13_d3h
```

[1] Rather than specifying a molecular geometry and working out any interactions between atoms from first principles (ab initio), we guess numbers between 0 and 1. In a slightly different context, this approach has a long history in quantum chemistry (as Huckel Molecular Orbital theory).

This reads input from the file `13_d3h.dat`, and writes its output to a new file called `13_d3h.out`.

You have been provided with a selection of input files of various sizes: the short-running ones are for use with simulators such as valgrind; for serious runs on real hardware use the longer-running examples like "coronene" (139s on a 2.2GHz Opteron) and "coronene_slater" (233 seconds).

Take care not to run on the same machine as another student as the program uses fixed file names in "/tmp/".

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".

   Try the Apple G5s, ICT supercomputer resources (Itanium, Opteron) possibly PDAs, DSP processors, graphics co-processor or FPGAs.

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. The really interesting problems are, of course, the long-running ones.

4. The numerical results reported by the application need not be absolutely identical, but if not you must justify the correctness of your results[2].

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:

- Cachegrind and cg_annotate
- kcachegrind - kcachegrind.sourceforge.net - graphical interface to cachegrind

[2] The gcc/gfortran flag -ffloat-store needed with this application to ensure convergence. This does impact performance somewhat and may be avoidable with other compilers.

- gprof - standard command-line profiling tool.
- kprof - kprof.sourceforge.net - graphical interface to gprof
- 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.
- AMD's CodeAnalyst (installed on CSG Athlon machines - Start→Programming→AMD) (if you have an AMD machine)[3].
- 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)

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

- Intel's compilers
  (http://www.intel.com/software/products/compilers/)
- The Pathscale compiler
  (http://www.pathscale.com/ekopath.html)
- 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.

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, 2007

[3] To do this you will need to build the code using a native Windows compiler. This is easier if you can use the Fortran sources, see the "OriginalFortran/" subdirectory.