Programming of Supercomputers

Assignment 1
17.10.2014

Deadline: `31.10.2014 @ 08:00 CET`

This lab course deals with the parallelization and tuning of a generalized orthomin solver from the Fire benchmark suite.

The tasks you will have to complete during this semester are: *single-core optimization, analysis of I/O behavior, visualization, 3-steps parallelization and performance tuning.*

1 Sequential Optimization - GCCG Solver

Quite often the great speedup potential of single-core execution of applications is overseen by many developers and instead all the effort is put into the parallelization step. This leads to inefficient usage of resources and poor scalability.

This first assignment concentrates on optimizing the sequential behaviour of the *gccg solver* and consist of the following subtasks:

1. Get to know the running environment (SuperMUC). Collect key characteristics of the system, e.g. *frequency, number of CPUs/Cores, memory, etc.*, and fill in the "Runtime Environment" worksheet.

2. Carry out performance measurements using PAPI, the Performance Application Programming Interface. Measure the following metrics for the *computation* phase of each of the two input files *tjunc.dat* and *cojack.dat* using the `-O1` and `-O3` optimization levels:

   - execution time
   - Mflops
   - L2 cache miss rate
   - L3 cache miss rate

   Write a short report comparing the measurements for the two input files and the correlation between metrics for one single file. Give short explanations for the observed behaviour.

3. Analyse the effect of vectorization on the *computation* phase when using the *cojack.dat* input file and `icc` compiler. Useful compiler flags: `-[no]-vec`, `-vec-report`, `-xhost`. Check the *man page* and online resources[1] for more information.

   Provide a short analysis report including measurements and conclusions.

Note: measured performance data has to be the average of at least three runs of the code. Please include both the measured values and the computed average values in the data tables you submit.

[1] http://d3f8ykwhia686p.cloudfront.net/1live/intel/CompilerAutovectorizationGuide.pdf

2 I/O Performance

Manipulating data files is very slow compared to accessing the main memory. This can be considerably improved by using a more concise data format such as binary.

1. Write a small tool to transform the text input files into a binary format;
2. Adapt the GCCG source code to read in correctly the new binary files;
3. Compare and explain the differences between the reading times and between the size of the files.

3 Visualization using ParaView (`http://www.paraview.org`)

When you start parallelizing your code, your should always be sure that the new results are still correct. This can be done by comparing the values variables like the *residual* or *number of iterations*, or by checking the computed results.

A good impression on the results though, is also provided by visualisation. Use the popular visualization software *ParaView* to check the output of the Fire benchmark.

1. Extend the source code to store the results into a VTK file format. Use the provided `vol2mesh` and `write_result_vtk` functions.
2. Use ParaView to generate plots of `VAR`, `CGUP` and `SU` for the `pent.dat` input file.

4 Resources

On the course page you can find: the Fire Benchmark source code, 3 input data files, and the template for the SuperMUC properties table.
http://lrr.in.tum.de/~berariu/teaching/superprog1415.php

5 Submission Requirements

Your GCCG code must:

- compile by simply running `make` without any arguments and generate a `gccg` executable;
- accept 3 arguments in the following format:
  `gccg <format> <input file> <output prefix>`
  - format - file format to use: (only) `bin` or `text`
  - input file - input data set
  - output prefix - a common prefix to be prepended to the name of all output files, e.g. **run1.**SU.vtk, **run1.**VAR.vtk, etc.
- generate an error if wrong arguments are given;
- generate <prefix>SU.vtk, <prefix>VAR.vtk and <prefix>CGUP.vtk upon completion;

Your binary conversion tool must:

- compile using `make binconv` and generate a `binconv` executable;
- accept 3 arguments in the following format:
  `binconv <input file> <output file>`
- generate an error if wrong arguments are given.

Commit all assignment files to your team repository:

1. Store all source code files to a folder A1/code/: *.c, *.h, and Makefile
2. Create a spreadsheet (`.xlsx` or `.ods`) with the collected performance data and store it as Data.ods or Data.xlsx to the A1/data/ folder.
3. Store the generated plots for pent.dat to the A1/plots/ folder:
   - pent.SU.jpeg & pent.SU.vtk
   - pent.VAR.jpeg & pent.VAR.vtk
   - pent.CGUP.jpeg & pent.CGUP.vtk
4. Write a lab report and store it as Report.pdf to the A1/report/ folder. The report has to include:
   - a short description of the execution environment;
   - formulas used for computing the metrics (cache miss rate & Mflops);
   - comparison of the achieved Mflops against the peak system performance;
   - the analysis required in 1.2, 1.3, and 2.3;
5. Store any processing scripts or extra files into A1/scripts/

Please note:

- Code that does not compile will not be taken into consideration.
- The results delivered in your report will be checked against those measured by the automatic tests on your code.