Optimising the Serial Jacobi Code

Adam Matheson
Student Number: P 1757290

October 17, 2017

1 Introduction

As parallelization continues to increase in the world of high-performance computing (HPC), optimising the serial code to begin with is just as important (if not more important) than ever. Parallelizing one's code may seem like a tempting choice when it has the potential to offer such large performance increases without the tedium of profiling and analyzing code as required with optimisation, but it would be remiss to leave such performance on the table. What follows is a walkthrough of an attempt to optimise serial code running the Jacobi algorithm for solving a set of linear equations.

2 Optimisations

2.1 Baseline

In order to make meaningful assessments of the performance gains made whilst optimising, baseline performance is required beforehand. From here on unless stated otherwise, performance will be tested on matrix orders 1000, 2000, and 4000 to give a range of figures, so places where Xs/Yz/Zs is seen refers to run time for matrix order 1000 = X (in seconds), 2000 = Y (in seconds), and 4000 = Z (in seconds).

Running the code as-is on BlueCrystal Phase 3, compiled with GNU compiler without any optimisation flags or debugging or any improvements to the stock code, gave times of 10.91/130.75/1180.81s.

- Time: 10.91s/130.75s. Changes: N/A
- Time: 3.27s/53.35s. Changes: Compiler from gcc to icc.
- Time: 14.37s. Changes: Compiler flag -g, added profiling code to jacobi.c, profiled with pprof –text –lines ./jacobi jgperf.txt. Also had to install gperftools, requiring libunwind. Then installed dot, gv and EPEL for -gv visualisations. First jgperf.txt generated.

- Line 64 analysed. Looping through data -¿ data structure issues?
- Examine data init block. Simplified to 3X3 and worked through using pen and paper. Going through column-wise across a virtual 2D array - jumping across the massive array, terrible for memory. Not contiguous at all.
- Transposed matrix. Solver time down, overall time up. Profiler says massive amounts of time copying array.
- Direct solver to solver test: n = 400 w/ transpose: 34.28s/0.78s, n = 400 w/out transpose: 0.79/0.78s. No perf. improvement.
- Time: 1.1/90s for 4000. Changes: Rewrote init instead of transposition. Rewrote traversal in run.