Programming Concurrent Systems
Sequential

Peter Rutgers
P.Rutgers@gmail.com
Student number:
???

Thomas Schoegje
T.M.Schoegje@students.uu.nl
Student number:
10068767
UvAnetID:
6323839

11 november 2013

1 Introduction

Our introduction into the world of programming concurrent systems starts with building a reference point for future assignments to compare to. By sequentially implementing a problem that could easily be implemented in a more concurrent manner it is possible to compare future concurrent implementations of the same problem in terms of performance.

The problem that will be addressed and solved in a sequential and later on concurrent methods is a simulation of heat dispersion on a surface. By representing the surface as a grid with a certain resolution, the next temperature at some point on this grid at some point in time can be calculated from the temperatures of all cells in the (Moore) neighbourhood. This simulation will run either until the number of iterations reaches some limit or the system reaches some convergence (i.e. the difference between two time iterations is less than some threshold, and changes in the system are halted/halting).

The system used for this model should of course be the same that will be used for later tests with concurrent implementations of the problem. For the sake of comparability it should allow concurrent systems to display their strengths, and for this reason the DAS-4 system was chosen. By using a cluster with multiple nodes it is possible to spread the work either to several nodes or to assign a sequential version to a single node. In this first assignment, the sequential approach is briefly discussed. This includes a simple overview of the compilation options that might increase its chances against concurrent competitors as well as a more slightly detailed look at the problem.

2 Compiler options

Several compiler options were tested to see how they influenced the performance of the program. The following parameters were used for all of the benchmarks: N=500, M=500, threshold = 0.01 and iteration limit = 100000. This resulted in a convergence after 4330 iterations, where T(min) is near -1, T(max) near 1 and the T(diff) is 0.9999401 . How quickly this was achieved differed depending on the compiler options used only, and the results can be seen in Table 1. The scores listed are the averages of two runs in order to somewhat decrease the effects of a bad run (wallclocktime was used).

Note that for the compiling using the flag -march=XXX was done by using qrsh to get access to a node so that it was possible to easily both compile and run the code on the same machine by using the value native.

Options	Execution time (s)	FLOP/s
-O2	27.27545	476252825
-O3	27.28475	476090825
-Os	29.01059	447767925
-O2 -march=native	27.722125	468582250
-O2 -ffast-math	27.56047	471327795
-O2 -fomit-frame-pointer	27.29234	475958105
-O2 -fprofile-generate	27.27985	476176150
-O2 -fprofile-use	27.29475	475916005

It should firstly be noted that the results found were the average of a very small sample size, and many of these conclusions may be foregone. Notably the -Os option stands out as by far the worst option, decreasing performance a lot compared to all alternatives. It differs almost a factor 4 from the best option compared to any other flag.

The next cluster of options in terms of increase of performance are the -ffast-math and -march=native flags. Interestingly the -ffast-math flag chooses to sacrifice accuracy in calculations in order to gain in speed, and it turned out this time that it was in fact one of the slowest benchmarks. Regardless of chance, it seems that the promised gain in speed may not be very significantly benificial. The -march=native option attempts to optimise code trying to take advantage of domain knowledge of the processor architecture. It turned out to be detrimental in this case, and possibly the architecture-specific optimisations are either not as efficient as the general optimisations or do not have as strong of a synergy with the other optimisations used.

This leaves most of the optimisations near the 27.27 - 27.29 mark, with the -O2 optimisation edging out all others. This is the default optimisation when compiling, and it should its robustness in this set of tests. It beats the -O3 improvement, which attempts more changes to the code, and these may end up actually being detrimental to the performance. Similarly for the -Os option. This leaves the -fprofile options and the -fomit-frame-pointer flags. Like the -march flag, the -fprofile options attempt to make a profile based on the system for which the code is compiled, but by allowing this to happen runtime instead of making a priori assumptions on the general architecture allows it to be more effective. In this run however it did not edge out the use of the default -O2 option.

Overall, it seems that the -O2 option is a robust solution for compilation.

3 Measuring performance on task

For analysing the influence of the N and M parameters (width and length of the cylinder respectively) on the computational complexity, we set the convergence threshold to 0.001 and the iteration threshold to a number that would never be reached. The rest of the parameters and options were kept the same. The results follow in Table 2 (the best result of 3 runs was chosen in each case). The test runs were ordered to underline differences and similarities.

N	M	iterations	T(diff)	T(avg)	Time (s)	FLOP/s
100	100	25050	9.9988.66e-04	-1.516323e+01	6.579761e+00	4.568559e+08
100	50	20248	9.999955e-04	1.207586e+00	2.651454e+00	4.581943e+08
50	100	9719	9.999819e-04	-6.043420e+01	1.276005e+00	4.570048e+08
100	200	28210	9.999688e-04	-3.790535e+01	1.428616e+01	4.739137e+08
200	100	43387	9.999725e-04	8.791846e+00	2.198645e+01	4.736049e+08
100	1000	21136	9.999088e-04	-6.538679e+01	5.315528e+01	4.771534e+08
1000	100	53207	9.999991e-04	-2.813677e+01	1.351674e+02	4.723658e+08

These results show us that although we are calculating the 'uniform' neighbourhood of a cell, that it matters whether the cylinder is long or wide. It appears that it is easier to calculate wider cylinders than long ones. There is notably a difference in calculation due to the rollover at the sides.