COMP35112 Laboratory Exercise 3: Relaxation
Duration: 2 weeks

Learning outcomes

On successful completion of this exercise, a student will have (1) written a multithreaded 1-D relaxation program, and (2) run it on a multicore machine and measured its performance when executing on different numbers of active cores.

Introduction

A common scientific application that can be fairly easily parallelised is the solution of partial differential equations (PDEs). This exercise aims to implement a parallel version of a solution of a particular PDE, using a technique known as *relaxation*, and evaluate its performance on the multicore machine mcore48. What makes this exercise different from the two previous exercises is the need for barrier synchronisation at the end of each iteration.

Underlying mathematics

The PDE to be solved is known as **Poisson's equation**. The normal form of Poisson's equation is 3-dimensional, as follows: $\frac{\partial^2 \mathrm{V}}{\partial \mathrm{x}^2} + \frac{\partial^2 \mathrm{V}}{\partial \mathrm{y}^2} + \frac{\partial^2 \mathrm{V}}{\partial \mathrm{z}^2} = f(x, y, z)$, where $f(x, y, z)$ is known as the *forcing function*. We shall solve a 1-dimensional version because it is easier. The 1-dimensional equation is: $\frac{\partial^2 \mathrm{V}}{\partial \mathrm{x}^2} = f(x)$, where V(x) is the unknown for which we want to solve. Of course, this is a *continuous* equation, so we need to transform it into a discrete form.

Algorithm

The 1-dimensional space is discretised into 4096 pieces, so a solution will find the values of V at 4097 discrete points. It is given, as a *boundary condition*, that V(0) = V(4096) = 0.0 at all times. You should also initialise all other points V(i) to start with a value of 0.0. The forcing function $f(x)$ is zero everywhere, except that $f(1024)$ = 1.0 and $f(2048) = 5.0$.

The relaxation technique uses arrays of doubles for the values of V and f, and replaces the continuous Poisson equation for all non-boundary points x = i with the following: `V[i-1] + V[i+1] - 2*V[i] = f(i)`. It then computes a new approximation for each non-boundary point `V[i]` by rearranging this new equation and using the previous values of `V[i-1]` and `V[i+1]`. The procedure repeats iteratively and it can be shown that the computed values gradually converge towards the true solution. In practical terms, we continue the iteration until the change (between new and old values of `V[i]`) **at every point** `i` is less than 0.1% of the new value.

What to do

To parallelise the code, assign responsibility for separate portions of V to different threads. Note that there will need to be a barrier at the end of each iteration so that all threads keep in step and make the same decision about the need for further iterations.

To check that your results are correct, rather than looking at all 4097 final individual values, the number of iterations required for convergence should be 65251.

Fully develop this application on a normal teaching domain machine before running it on mcore48. A script `Demo.sct` for the SGE batch queue that will run the `main` method in a Java class called `Demo` is available on the course unit materials webpage.

Speedup

Do not include initialisation or final checking in your timing measurements. To demonstrate speedup without interference from other students, you will need to use the SGE batch system, as for the earlier exercises. You should run with a range of numbers of threads to observe how the performance changes. Generate a performance curve from the observed timings showing how performance speeds up with the number of active threads. Make each execution time measurement more than once in order to deal with variation in observations.

Deliverables

The main deliverable for this exercise is the performance graph(s) showing how performance varies with the number of active threads/cores. In this case, it is difficult to show performance increasing significantly as the number of threads/cores increases because the barrier introduces a large time overhead compared with the time needed for one iteration of the main loop. You might want to think of (and perhaps try) ways of decreasing this overhead; however, the topic is complex, and requires the use of techniques that are research level rather than undergraduate level. (**Hint**: look for **asynchronous**, as opposed to synchronous, methods for relaxation; note that additional iterations may be needed in order to get the same degree of convergence.)

Submission

Write a brief report on this exercise, commenting on the speedup that your performance curve shows and stating what you did in order to get this (if you try some of the ideas suggested above, you may want to show how the different approaches that you have used have affected the obtained speedup). Submit this report together with the performance curve and your source code in a single PDF or ZIP file via the assignment window on the Blackboard course page. The deadline for submission is one week after the final scheduled lab session for this exercise.

Marking Scheme

This exercise is marked out of 20 marks, as follows:
Code – 9 marks
Report – 5 marks
Performance graph(s) – 6 marks