High-Performance Computing (G63.2011.002/G22.2945.001), Fall 2010

Homework Set 2

Due: October 12, 2010 **Out: September 28, 2010**

In this assignment you will parallelize an iterative solver for the Poisson equation for a shared memory machine using OpenMP.

The problem is:

$$\triangle u = u_{xx} + u_{yy} + u_{zz} = f \text{ in domain } \Omega$$
$$u = g \text{ on } \partial\Omega$$

In such an equation the unknown u can represent for example the steady state heat distribution in a domain where the temperature at the boundary is kept fixed at the level specified by the function g.

You are given an initial serial version of a 3D solver that uses Jacobi's method. The solver uses a second-order finite difference scheme to approximate the derivatives, as described in class. The domain Ω is a cube from `[xlo,ylo,zlo]` to `[xhi,yhi,zhi]`. Use loop level parallelism to speed up the run time of the code as much as possible.

You should time your code for domain sizes that include 33, 65, 129 on a side. Note that you cannot directly always compare runs with different grid sizes because it may take fewer iterations to converge on a coarser grid, used for example in debugging. For timing results you have to make sure to run a fixed number of iterations. You should make speedup plots using between 1 and 8 threads. (There are 8 cores on the Union Square cluster, but you can experiment with using more threads and seeing what that does). To remind you, the speedup S is defined by $S = T_1/T_p$, where T_1 is the run time for the sequential algorithm and T_p is the run time on p processors. You should feel free to optimize in any other ways you find, including compiler flags, improvements to the algorithms, and code rearrangements. Experiment with using either the 2 norm or the max norm to determine convergence, and compare the speedups using one or the other (both appear in the original serial code). Other things to experiment with if you have the time or inclination could include using a Gauss-Seidel iteration (red-black is easiest here), or experimenting with a coarse-grained instead of loop-level parallelism.

You should submit your parallelized code along with a writeup. The discussion should include the things you tried (both successful and unsuccessful), your timing results in a table, and a graph of problem size N versus speedup S with the curves from all threads on the *same* plot. The plotting package `xmgrace` might be a good tool to learn to use for this. Please also include your compile line too, and which compiler you used (the Intel compilers are faster, try them and see).

Using USQ

You can do these tests using the interactive queue on Union Square. (It's probably easiest to write a script that runs all the cases and puts the output in a file however.) To run interactively you can use the following `qsub` command (also on the ITS wiki),

```
qsub -I -q interactive  -l nodes=1:ppn=8,walltime=00:50:00
```

This gets you 50 minutes of wall clock time on all 8 cores of 1 node.

Please remember that to invoke the OpenMP compiler the option `-openmp` (icc) and `-fopenmp` (gcc) is needed on both the compile line and the link line, as indicated in the Makefile. If you are using the Intel compiler you have to remember to load the compiler module. (You need to type `module avail` to see what modules are available, and `module load <module-name>` to add it to your path). For `xmgrace` you type `module load grace/intel/5.1.22`.

To download the starter kit for homework 2, you should follow the same instructions as for hw1 to get it from the forge website. The starter kit will contain a file called `serial_hw2.c`, and the files `timer.c` and `timer.h`, which you should already have from the last assignment. Finally a Makefile to make the serial code is included. You should edit it to add instructions to compile your parallel code.