CSc 422/522 — Parallel Programming Project

due Tuesday, March 9

You are to write four programs as specified below and then to perform experiments and write a report describing your results. Details on the experiments and reports will be handed out later. This assignment is worth 40 points for undergraduates and 60 points for graduate students.

Laplace's equation is an example of a two-dimensional partial differential equation. (See Section 11.1 of the text and also Chapter 15 of the SR book.) This kind of equation can be approximated numerically by a finite difference method. In particular, cover a two-dimensional spatial region with a uniformly-spaced grid of points. Given fixed values for points on the boundary—and given initial values for interior points—the steady-state values of the interior points can be calculated by repeated iterations. On each iteration, the new value of a point is a combination of the old and/or new values of neighboring points. The computation terminates either after a fixed number of iterations or when every new value is within some acceptable tolerance `epsilon` of its old value.

There are three basic iterative techniques: Jacobi, Gauss-Seidel, and successive over-relaxation (SOR). Multigrid is an advanced technique that is harder to program but that leads to faster convergence. These techniques are described in detail in Section 11.1 of the text.

Programs for Undergraduates

Start with the sequential SR program for Jacobi iteration handed out in class. (It's stored in the usual place on Lectura.) Then write four SR programs—or four C plus `pthreads` programs:

an efficient version of sequential Jacobi
an efficient version of Gauss-Seidel
a parallel version of your sequential Jacobi
a parallel red/black version of your Gauss-Seidel

By "an efficient version", we mean that you hand optimize the basic program to get rid of expensive operations such as copying arrays, dividing rather than multiplying, and calling functions that could easily be placed in line.

For the parallel programs, divide the grid into strips and use one worker process per strip. For the parallel red/black program, each worker should further divide its strip of points into a red region (on top) and a black region (on the bottom). Implement an efficient *dissemination* barrier and use it when you need barrier synchronization.

Programs for Graduate Students

Start with the sequential SR program for Jacobi iteration handed out in class. (It's stored in the usual place on Lectura.) Then write four SR programs—or four C plus `pthreads` programs:

an efficient version of sequential Jacobi
an efficient version of sequential multigrid using a single V cycle
a parallel version of your sequential Jacobi
a parallel version of your multigrid program

By "an efficient version", we mean that you hand optimize the basic program to get rid of expensive operations such as copying arrays, dividing rather than multiplying, and calling functions that could easily be placed in line.

For the parallel programs, divide the grids into strips—including each grid in the multigrid program—and use one worker process per strip. Implement an efficient *dissemination* barrier and use it when you need barrier synchronization.

For the multigrid programs, use four levels (granularities); you may use separate matrices for each level. Use the restriction and interpolation operators described in the text on pages 1117 and 1118. The steps in a four-level V cycle are:

> iterate on the fine grid 4 times
> restrict to a 2h grid, then iterate 4 times
> restrict to a 4h grid, then iterate 4 times
> restrict to an 8h grid, then iterate until solved (see `iterations` below)
> interpolate back to the 4h grid, then iterate 4 times
> interpolate back to the 2h grid, then iterate 4 times
> interpolate back to the fine grid, then iterate 4 times.

Use Jacobi iteration at each level.

Common Specifications

Each program should have three command-line arguments:

> `n`, the grid size (not including the boundaries)
> `iterations`, the number of times you compute new values for points
> `w`, the number of worker processes (for the parallel programs only)

If you unroll the main computational loop 2 times to avoid using an extra array in Jacobi iteration, then the loop should execute `iterations/2` times.

You may assume that `w` is between 1 and 4, and that `n` is a multiple of `8*w`. Use constant values of `1.0` for the boundaries of the grid and initialize interior points to `0.0`.

The output from each program should be:

> the command-line arguments
> the elapsed time (not including initialization or printing)
> the final value of epsilon (maximum difference between pairs of points)
> the final grid values

Write the arguments, elapsed time, and epsilon to standard output, and write the final grid values to a data file. Use the SR `age` function to compute the elapsed time of the computational part. If you use `pthreads` for your parallel programs, use the `times` function, not the `clock` function described earlier. See the manual page for `times` (it is in section 2 of the manual, so view it by executing "`man -s 2 times`").

Develop your programs on Lectura but run tests on Parallel (`par`) in order to get realistic performance results.