# Homework/Lab 4

**due Monday, March 10, 2003, 11:59 PM**

This assignment includes items to be done individually and items that you may do in groups of two. You will need to submit two files. `hw04.txt` should include your answers to the first two questions and must be done individually. The program for the third item should be submitted as `hw04.tar`. If you work in a group on the program, only one group member needs to submit `hw04.tar`. *Please use the filenames specified* and be sure to include your name in each file.

1. Jordan and Alaghband Problem 4.3, p. 163. This question is to be answered individually. Include your answer in a plain text file `hw04.txt`. (4 points)

2. Jordan and Alaghband Problem 4.8, p. 164. A "prescheduled parallel loop" is essentially what we have been calling "explicit domain decomposition," and "self-scheduled parellel loop" is essentially a "bag of tasks." This question is to be answered individually. Include your answer in a plain text file `hw04.txt`. (6 points)

3. (25 points) Write a C or C++ program using OpenMP that solves Laplace's equation on a two-dimensional, uniform, square grid, using Jacobi iteration. *Don't panic, it's not as hard as that might make it sound!*

   **Some background**

   Laplace's equation is an elliptic partial differential equation that governs physical phenomena such as heat. In two dimensions, it can be written

   $$\Phi_{xx} + \Phi_{yy} = 0.$$

   Given a spatial region and values for points on the boundaries of the region, the goal is to approximate the steady-state solution for points in the interior. We do this by covering the region with an evenly-spaced grid of points. A grid of $8 \times 8$ would look like this:

```
* * * * * * * * * *
* . . . . . . . . *
* . . . . . . . . *
* . . . . . . . . *
* . . . . . . . . *
* . . . . . . . . *
* . . . . . . . . *
* . . . . . . . . *
* . . . . . . . . *
* * * * * * * * * *
```

The $8 \times 8$ grid represented by the dots is surrounded by a layer of boundary points, represented by *'s. Each interior point is initialized to some value. Boundary points remain constant throughout the simulation. The steady-state values of interior points are calculated by repeated iterations. On each iteration, the new value of a point is set to a combination of the old values of neighboring points. The computation terminates either after a given number of iterations or when every new value is within some acceptable difference $\epsilon$ of every old value.

There are several iterative methods for solving Laplace's equation. Your program is to use Jacobi iteration, which is the simplest and easily parallelizable, though certainly not the most efficient in terms of convergence rate.

In Jacobi iteration, the new value for each grid point in the interior is set to the average of the old values of the four points left, right, above, and below it. This process is repeated until the program terminates. Note that some of the values used for the average will be boundary points.

**What to do and how to do it**

- To avoid special cases at the boundaries, you should allocate your grid for an $n \times n$ simulation to be $(n+2) \times (n+2)$, so you can use row and column 0 and row and column $n+1$ to store the boundary values.
- You will need two copies of the grid, one to store the "current" solution values and one to store the "next" solution values. Do not copy the "next" solution to the "current" solution at each step. Instead, always do two iterations. The first uses one grid as "current" and the other as "next," then the second swaps their roles. This is an example of a simple *loop unrolling*.
- Each grid cell computation looks something like this:

```
nextgrid[i][j] = (grid[i-1][j] + grid[i+1][j] +
                  grid[i][j-1] + grid[i][j+1]) * 0.25;
```

  Doing `* 0.25` is usually faster than `/ 4` since multiplication is an easier operation for a computer than division.
- After each pair of iterations, the corresponding values in each cell of the two grids are compared, and the maximum such value is computed. If this value is less than $\epsilon$, the computation terminates. Otherwise, it continues.
- A maximum number of iterations is also specified, after which the computation stops even if the error tolerance has not been reached.
- Your program should take two command-line arguments: the maximum number of iterations and the error tolerance $\epsilon$.
- Initialization of the boundary and interior will determine the exact problem being solved. I suggest initializing your interior to all 0's and the boundary to have two sides set to 1, two sides set to 0, as follows:

```
1 1 1 1 1 1 1 1 1 1
1 . . . . . . . . 0
1 . . . . . . . . 0
```

```
1 . . . . . . . . 0
1 . . . . . . . . 0
1 . . . . . . . . 0
1 . . . . . . . . 0
1 . . . . . . . . 0
1 . . . . . . . . 0
1 0 0 0 0 0 0 0 0 0
```

This keeps the initialization simple, but still allows for some work during the solution phase.

- At the end of the computation, print out the total number of iterations needed, and the time taken to achieve the solution.
- Get the serial version working first. Choose values for the size of the grid, maximum number of iterations, and $\epsilon$ to get your program to run for a few minutes.
- Parallelize your program. Your parallel program should create and destroy the OpenMP threads only once.
- You may break down the computation any way you'd like. Justify your choice in the `README` file that you include in your submission.
- You will need a number of OpenMP directives. Explain each directive (what you needed to do and why you chose that particular directive) in detailed comments in your source code.
- Run your program using 1, 2, and 4 threads on the four-processor nodes of bullpen (*wetteland* or *rivera*; `ppn=4` in PBS). Submit these runs through PBS. Include the timings and your analysis of those timings in the `README` file that you include with your submission.

You may work in groups of two on this program and its writeup.

Your submitted tar file should include your `Makefile`, your C source code (including the timer code from class, if you choose to use it), your PBS script(s), and a brief `README` file that expains how to run your program, describes how and why you chose to parallelize your program, and describes and analyzes your timing results. Please do *not* include object files or your executable in your tar file.

**Honor code guidelines**: While the program is to be done only by you (meaning your group, if you choose to work in a group), along the lines of a **laboratory program**, I want to encourage you to ask questions and discuss the program with me, our TA, and with classmates outside your group, as you develop it. However, no sharing of code between groups is permitted. If you have any doubts, please check first and avoid honor code problems later.

**Grading guidelines**: Your grade for the program will be determined by correctness, design, documentation, and style, as well as the required items in the detailed comments and `README` file.