

CS 420 SPRING 2006

MACHINE PROJECT 4

Due date: Friday, April 14, 2006

Solving Laplace Equation in Parallel

In this Machine Project, you are to iteratively numerically solve a two-dimensional *Laplace equation*. Laplace equation is one of the most frequently encountered second-order *partial differential equations* (PDEs). Its general form is

$$u_{xx} + u_{yy} = 0 \quad (1)$$

where $u = u(x, y)$ is a real-valued function of two space variables, x and y , such that $u(x, y)$ is twice differentiable with respect to both x and y . In the equation above, u_{xx} denotes the second partial derivative of $u(x, y)$ with respect to x , and similarly for u_{yy} .

Laplace's equation is an *elliptic* PDE; in particular, it is time-independent. You are to numerically solve it using the *finite difference method*. An example of how to implement finite difference method for a Laplace equation sequentially is given in the pseudo-code below:

```
while (max_change > close_enough) do
  max_change = 0;
  for i = 2, n-1
    for j = 2, n-1
      old_value = v(i, j)
      !Comment: replace each value with the arithmetic average of its four neighbors
      v(i, j) = (v(i - 1, j) + v(i + 1, j) + v(i, j - 1) + v(i, j + 1))/4
      !Comment: maintain max_change equal to the largest change seen thus far
      max_change = max{max_change, abs(old_value - v(i, j))}
    end for
  end for
end while do
```

Due to its time-independent nature, Laplace's equation is well-suited for parallelization (admittedly, quite unlike the Gauss-Seidel method from MP3). Notice that the i and j loops only update the mesh points with coordinates from 2 to $n-1$. That's because the boundary values of the 2D array v are fixed; they define what is called *boundary conditions*. The region in the Cartesian plane on which we want you to solve the Laplace equation will be the unit square, and the fixed boundary

values defining the boundary conditions are

$$v(1, 2 : n) = v(n, 2 : n) = 0.0; \quad v(1 : n, 1) = 1.0; \quad v(1 : n, n) = 0.0 \quad (2)$$

Since you are to solve the Laplace equation on the unit square in the plane, the grid point $(i, j) = (1, 1)$ corresponds to the origin, whereas the grid point $(i, j) = (n, n)$ corresponds to the point $(x = 1, y = 1)$ in R^2 . For this MP, set $n = 200$.

Your project is made of four parts:

- sequential implementation of Laplace equation solver on the unit square (with the given boundary conditions etc.);
- parallel implementation using the *wavefront method*;
- parallel implementation executing the alternate rows in parallel; and
- parallel implementation executing the alternate columns in parallel.

For the wavefront method, you'll first need the master thread to initialize the boundary values. The master thread will then send to each processor the grid point values which that processor needs (in particular, no broadcasting!). Each processor will then update its grid point value $v(i, j)$. The processors need to be synchronized; once everyone completed the current iteration, the master thread will send to everyone the next set of values, etc.

The second and third parallel versions will execute the alternate rows/columns in parallel so as to avoid race conditions. For instance, in the row version, in the first pass the master thread will send the elements from rows 1, 3, 5,... to the respective processors, and in the second pass it will send the elements from rows 2, 4, 6,...

The Extra Credit Assignment

Graduate students taking the course for 4 credit hours should also implement the block parallel version of the finite difference algorithm for solving Laplace's equation. You may make the number of blocks the same as the number of processors you use. Do this exercise for $p = 4$ and $p = 16$ blocks/processors (you may check how it works for other values, but please include the performance plots and summary for these two particular values of p). You are welcome to choose how are the blocks themselves going to be processed: two obvious candidates are the wavefront order or the alternate (even-odd) row or column order. Please describe briefly in your report what your block parallel implementation does.

The Practical Matters

You should run your programs on the *Turing cluster*, which can be viewed as a distributed memory parallel system. For more on *Turing*, visit <http://www.cse.uiuc.edu/turing/>.

Choose either Fortran or C as your implementation language. The parallel part of your project is to be implemented using MPI. You are expected to submit (i) a single program file and (ii) a .doc or .pdf file with a brief analysis and performance plots. Please try to measure performance by inserting the appropriate timing commands right before and right after the nested loops where the actual finite difference iterations are being executed.

Insofar as the content, shape and format of your report, please refer to the prior MP descriptions and also to the detailed instructions posted to the newsgroup in the aftermath of MP1.