CS441/541 - Advanced Operating Systems
Solving Partial Differential Equations
Assignment #1

Introduction

Methods to solve partial differential equations (PDE's) fall into one of two categories: analytical and iterative. Analytical methods require that a set of equations be written to describe the behavior of the partial differential equation – sometimes this is rather difficult. Iterative methods begin by "guessing" at values at certain points, and then iterating toward (hopefully) more accurate estimates.

Consider a two-dimensional steady state temperature distribution problem. A thin steel plate is surrounded on three sides by a condensing steam bath (temperature $100°C$), and on the fourth side by an ice bath (temperature $0°C$). An insulating blanket covers the top and bottom of the plate, so all heat flow is in 2 dimensions. Imagine an $n \times n$ mesh overlaying the plate. The problem is to find the steady state temperature at each point of the $n \times n$ mesh in the plate.

This is an example of a linear second-order PDE. A set of difference equations relates the values of variables on neighboring mesh points:

$$V'_{x,y} = (V_{x-1,y} + V_{x,y-1} + V_{x+1,y} + V_{x,y+1})/4$$

where x and y refer to points on the mesh. The value on the left is the new value, computed from the old values on the right. All the new values are computed before any of the old values are updated.

This problem is simple enough to solve analytically; however, more complicated problems must be solved iteratively. To solve, each point is initially assigned a guessed value - this could also be interpreted as the starting temperature for each $V_{x,y}$ value. The difference equation is then used to calculate successively more correct values. This procedure was first proposed by Jacobi in 1845, and is called the Jacobi relaxation procedure.

The usual way to determine when a solution has been reached is to watch the maximum amount of change produced at each node at each iteration. Once the change becomes "small enough," we assume that the solution has been reached. This change amount, called the maximum error value, is represented by ϵ (epsilon).

A question arises - how many iterations are required to arrive at a solution? Analytical results have determined that around $\epsilon n^2/2$ iterations are required to reduce the error by a factor of $10^{-\epsilon}$. For example, an $\epsilon = 10^{-3}$ on a 64×64 mesh would require around $3(64)^2/2 = 6144$ iterations, a fairly slow rate of convergence.

A More Efficent Method

The most popular iterative method for solving PDE's is a modification of Jacobi called successive overrelation (SOR). SOR differs in two important respects:

1. A weighted average of $V'_{x,y}$ and $V^{old}_{x,y}$ is taken to determine the new value for $V_{x,y}$. This tends to smooth out the gyrations in value that sometimes occur.

2. New values replace old values immediately after they are computed. This eliminates the need for keeping two copies of the array, and therefore any unnecessary copying.

One final change is made that improves the convergence rate even further. It is called *odd-even ordering with Chebyshev acceleration*. Imagine the points forming a checkerboard. Every iteration over the mesh has two phases: first, all the "odd" points are calculated. Then, all the "even" points are calculated, using the odd value just calculated in the previous phase.

The convergence rate for the final algorithm has been found to be $\epsilon n/3$ for an error less than $10^{-\epsilon}$, which for the 64×64 mesh used earlier works out to be $64(3)/3 = 64$ iterations.

Programming Details

For an $n \times n$ problem, define the maximum size of the array to be $n + 2 \times n + 2$, and then run your loop from $1 \dots n$. This will leave the boundaries unchanged.

The Assignment

Write a distributed program, using MPI, that will solve the PDE described above. Since the calculation over the array is quite parallel, break up the array into"stripes" and have one processor calculate each stripe. The elements on the boundaries between each stripe need the new values that were calculated in the other stripe in order to complete their calculations. Thus, each stripe will need to coordinate with its adjacent stripes. This will require a communication between the stripes.

The exact size of the array and the number of threads should be entered on the command line. For example:

```
pde 64 4
```

would solve the problem of a 64×64 plate, using 4 threads.

Use the boundary conditions of 100 on three sides, 0 on the other side, and an ϵ of 0.05 .