CS5334 Spring 2016 Name ______________________________
Shirley Moore, Instructor
Lab 3 Due May 3
Part 1 is due April 21

# Parallel Solution and Analysis of the Time-Dependent 2D Heat Equation Problem

The purpose of this lab assignment is to practice MPI programming as well as analysis of parallel algorithms. You may do this assignment individually or in groups of up to four people. You should do the assignment on the Stampede supercomputer at Texas Advanced Computing Center. Please turn in your source code, Makefile, README file explaining how to build and run your code, and a report with your analysis to your SVN repository for the class.

We will develop a parallel numerical solver for the 2D heat equation using an explicit Euler time-stepping method. The equation is given as

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

on the domain $\Omega = [0,1]^2$ with the initial condition

$$u(x, y, 0) = f(x, y)$$

and the boundary conditions

$$u(0, y, t) = u(1, y, t) = u(x, 0, t) = u(x, 1, t) = 0.$$

We divide each of the two spatial dimensions into $N$ intervals of equal size such that $Nh = 1$, where $h$ is the step size. We also discretize the time dimension with uniform step size $k$ and choose $k = h^2/4$ so that the explicit Euler method is stable [1]. The discrete points on the regular grid are denoted by $x_i = ih, y_j = jh, t_n = nk$. Let $u_{i,j}^{(n)} = u(x_i, y_j, t_n)$ represent the discretization of $u$ and let $v_{i,j}^{(n)}$ denote our approximation of $u_{i,j}^{(n)}$ . Then using a finite difference scheme with the explicit Euler method as the time-stepping method, we can derive the following equation for $v_{i,j}^{(n+1)}$:

$$\begin{aligned} v_{i,j}^{(n+1)} &= v_{i,j}^{(n)} + \frac{k}{h^2}(v_{i-1,j}^{(n)} + v_{i,j-1}^{(n)} - 4v_{i,j}^{(n)} + v_{i,j+1}^{(n)} + v_{i+1,j}^{(n)}) \\ &= \alpha v_{i-1,j}^{(n)} + \alpha v_{i,j-1}^{(n)} + \beta v_{i,j}^{(n)} + \alpha v_{i,j+1}^{(n)} + \alpha v_{i+1,j}^{(n)}, \end{aligned}$$

where $\alpha = k/h^2$ and $\beta = 1 - 4\alpha$.

Note that $v_{i,j}^{(n+1)}$ depends on the value of $v^{(n)}$ at the point $(x_i, y_j)$ as well as at each of the four neighboring points $(x_i \pm h, y_j \pm h)$ . This update pattern is known as a 5-point stencil.

[1] Kai Velten, *Mathematical Modeling and Simulation: Introduction for Scientists and Engineers*, Wiley, 2009.

## Part 1. Algorithm 1 and Iso-efficiency analysis

Consider a 2D block distribution of the (x,y) grid points onto a $\sqrt{p}\times\sqrt{p}$ 2D Cartesian topology of processes. In order for a process that owns a block of points to advance its points one time step, it needs the values of neighboring grid points. Figure 1 illustrates an 8x8 block owned by a process and the neighboring grid points.

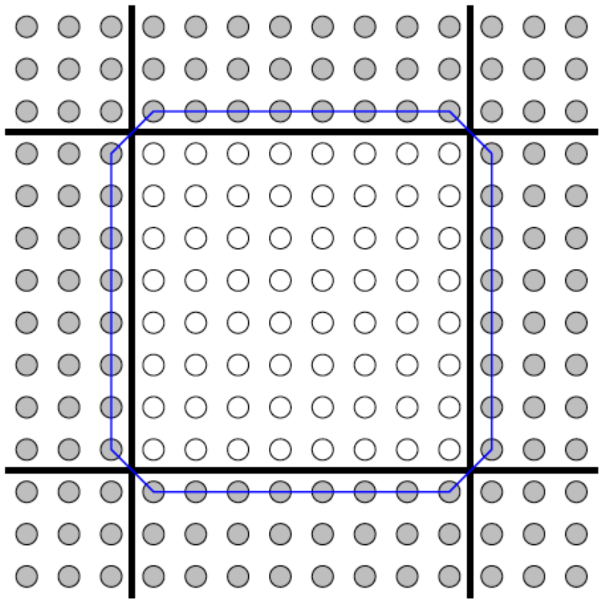

Figure 1. Block of grid points and one layer of neighboring grid points

Pseudocode for the basic algorithm 1 that we will implement is given below.

```
Algorithm 1
for n = 1, 2, … do
    Exchange border grid points with N, S, E, and W neighbors
    Advance the local grid points one time step
end for
```

1. Analyze Algorithm 1 theoretically with respect to the parallel execution time, speedup, and the asymptotic iso-efficiency function. Based on your analysis, what is the factor that most limits the scalability of the algorithm? (Note: Do not consider initialization and distribution of data in your analysis).

2. Implement Algorithm 1 using MPI. Use a 2D Cartesian topology for the process grid. Use MPI derived datatypes as appropriate.

Part 2. Performance Optimization

Optimization 1. In the basic algorithm 1, communication is not overlapped with computation. Note that it is possible to exchange the neighboring grid points while computing the new interior grid point values and then compute the border grid point values after the data have arrived. This overlap can be implemented by using asynchronous communication.

Optimization 2. It is possible to trade off communication and computation by performing some redundant computations. The idea is to exchange *r* layers of neighboring grid points so as to be able to advance the local block *r* time steps between communications. Figure 2 illustrates the neighboring points that need to be exchanged for *r* = 3.

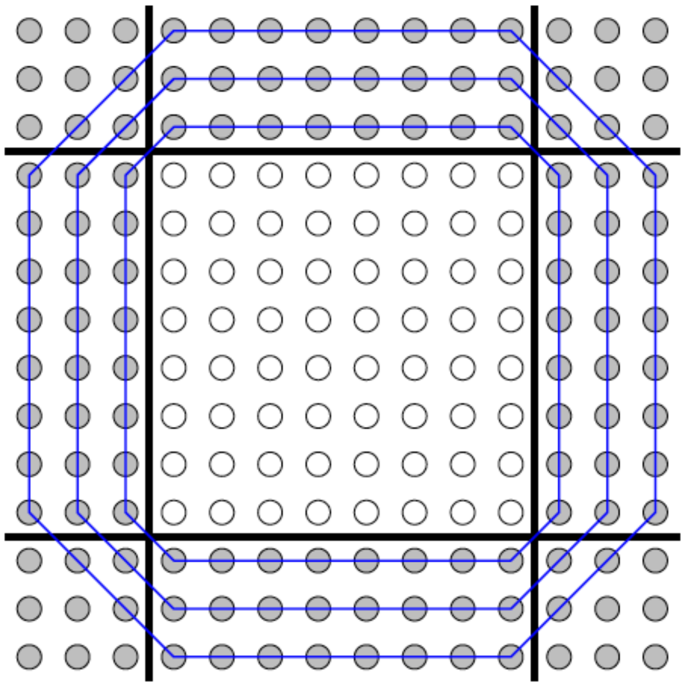
Figure 2. Block of grid points with 3 layers of neighboring grid points

3. Implement Optimization 1 as Algorithm 2.

4. Implement Optimization 2 as Algorithm 3.

5. Implement both optimizations together as Algorithm 4.

6. Evaluate the impact of the value of *r* on Algorithms 3 and 4.

7. Evaluate the scalability of Algorithms 1 through 4 by scaling the problem size according to the iso-efficiency function you found in part 1. In order to get reasonable results, you will need to start with a problem size that gives decent efficiency with a small number of processes. Be sure to measure parallel runtime in a valid manner.