Parallel Numerical Solution of the 2D Heat Equation

Design and Analysis of Parallel Algorithms

Due: Monday May 2nd at 12:00

1 Introduction

We consider developing a parallel numerical solver for the 2D heat equation

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

on the domain $\Omega = [0,1]^2$ (i.e., the unit square) with the initial conditions

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

and the boundary conditions

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

We begin by dividing each of the two spatial dimensions into N intervals of uniform size. In other words, we have $Nh = 1$, where h is the step size in space. We also discretize the time dimension with a uniform step size k, which we choose as

$$k = \frac{h^2}{4}$$

in order for the finite difference scheme to be stable. The discrete points on the grid are denoted by

$$x_i = ih, \qquad y_j = jh, \qquad t_n = nk.$$

By restricting the function u to the grid, we obtain a discretization of u:

$$u_{i,j}^{(n)} = u(x_i, y_j, t_n).$$

Let $v_{i,j}^{(n)}$ denote our approximation of $u_{i,j}^{(n)}$. If we use a finite difference scheme with explicit Euler as the time-stepping method, then we eventually arrive at the following formula 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 the constant α and β are defined as

$$\alpha = \frac{k}{h^2}, \qquad \beta = 1 - 4\frac{k}{h^2}.$$

The key point is that $v_{i,j}^{(k+1)}$ depends exclusively on the value of $v^{(k)}$ at the point (x_i, y_j) as well as each of the four neighbouring points $(x_i \pm h, y_j \pm h)$. The update formula above is also known as a 5-point stencil.

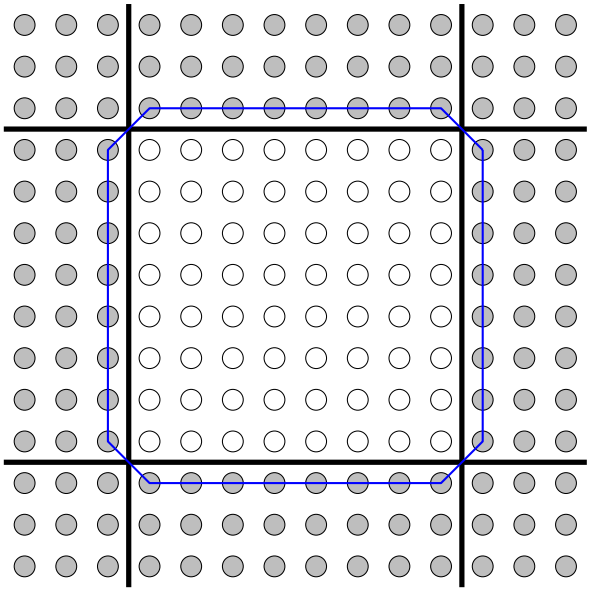

Figure 1: Illustration of a block of grid points and one level of neighbouring grid points.

Algorithm 1 Basic

1: **for** $k = 0, 1, \ldots$ **do**
2: Send the interior borders to the neighbouring processes.
3: Receive the exterior borders from the neighbouring processes.
4: Advance the local block one time step.
5: **end for**

2 Distributed-memory parallelization

We distribute the grid points onto a $p_x \times p_y$ process mesh with a 2D block distribution. In Figure 1, we illustrate an 8×8 block in a larger grid and three additional rows/columns on either side of the block. In order for the process that owns the central block to advance its points one time step, it must know the values of the neighbouring grid points Note that these are stored on the four immediate neighbours of the process. Algorithm 1 outlines a straightforward parallel algorithm.

One drawback with Algorithm 1 is that the communications on line 2–3 are not overlapped with computations. Therefore, the parallel execution time will include the full cost of the communication. It is possible to overlap the communication with computation by observing that the interior grid points in the local block can be computed before the grid points on the border, which means that we can communicate the border while advancing the interior grid points. The resulting algorithm is outlined in Algorithm 2.

Another drawback with Algorithm 1 is that we perform only $\theta(n^2/p)$ computations between each pair of communication phases. It is possible to increase the granularity of the computations by a factor of r by performing some redundant computations. The idea is to send in each communication phase all the points that we need to advance the local block r time steps. The three blue loops in Figure 2 illustrate the points needed for the case $r = 3$. Note three things. First, we communicate with eight instead of four neighbouring processes.

Algorithm 2 Overlapped

1: **for** $k = 0, 1, \ldots$ **do**
2: Asynchronously send the interior borders to the neighbouring processes.
3: Asynchronously receive the exterior borders from the neighbouring processes.
4: Advance the *interior* of the local block one time step.
5: Wait until the sends and receives complete.
6: Advance the *borders* of the local block one time step.
7: **end for**

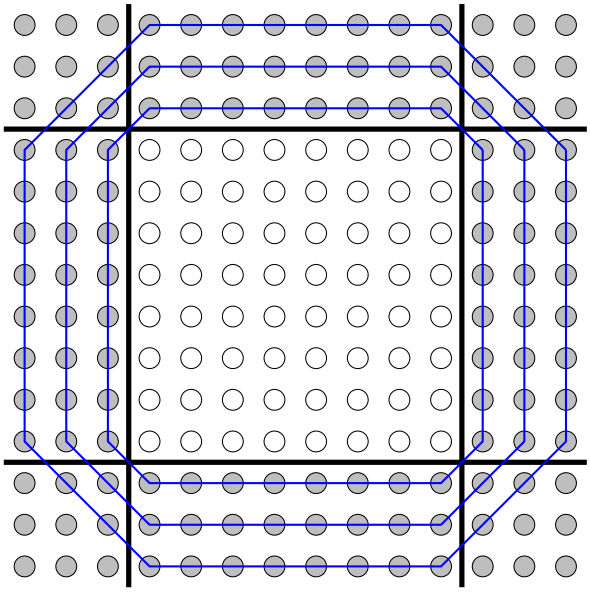

Figure 2: Illustration of a block of grid points and three levels of neighbouring grid points.

Second, we communicate slightly more grid points ($4 \cdot 3$ in Figure 2). Third, we perform redundant computations to advance the exterior borders locally. Algorithm 3 outlines the communication-avoiding Basic algorithm.

Algorithm 3 Communication-Avoiding Basic

1: **for** $k = 0, r, \ldots$ **do**
2: Send the r interior borders to the neighbouring processes.
3: Receive the r exterior borders from the neighbouring processes.
4: Advance the local block r time steps.
5: **end for**

Of course, we can also apply the same technique to increase the granularity in Algorithm 2. The resulting communication-avoiding Overlapped algorithm is outlined in Algorithm 4.

Let us briefly touch upon the issue of appropriate data structures. A good option is to store the local $m_y \times m_x$ block at the center of a matrix of size $(m_y + 2r) \times (m_x + 2r)$. Then we can store the exterior borders received from the neighbouring processes in the extra space created around the local block in the center of the matrix.

Algorithm 4 Communication-Avoiding Overlapped

1: **for** $k = 0, r, \ldots$ **do**
2: Asynchronously send the r interior borders to the neighbouring processes.
3: Asynchronously receive the r exterior borders from the neighbouring processes.
4: Advance the *interior* of the local block r time steps.
5: Wait until the sends and receives complete.
6: Advance the r *borders* of the local block r time steps.
7: **end for**

3 Instructions

The aim of this assignment is to practice MPI programming as well as analysis and evaluation of parallel algorithms. You are asked to do the following:

1. Analyze Algorithm 1 (Basic) theoretically with respect to the parallel execution time and the asymptotic isoefficiency function. Based on your analysis, what is the factor that limits the scalability (latency, bandwidth, or degree of concurrency)?

2. Implement Algorithms 1–4 using MPI.

3. Evaluate the impact of the parameter r on the performance of Algorithms 3–4.

4. Evaluate the weak scalability of Algorithms 1–4 by scaling the problem size according to the isoefficiency function you found above. In order to get reasonable results you must make sure to start from a problem size that is large enough to give decent efficiency on a small number of processes.

5. Evaluate the strong scalability of Algorithms 1–4 by keeping the problem size fixed and increasing the number of processes. Again, start from a problem size that is large enough to give decent efficiency on a small number of processes. See if you can increase the number of processes beyond the optimal number for the chosen problem size.