Uppsala University
Department of Information Technology
Scientific Computing

Programming of Parallel Computers
2016-02-22

Compulsory Assignment 3

Shared memory parallelization using OpenMP

Prepared by Anton Artemov, anton.artemov@it.uu.se.

1 Problem to solve

The main idea of this assignment is to solve a 2-dimensional PDE in parallel using OpenMP. As a model problem we will use a simple Poisson equation with homogeneous Neumann boundary conditions on a rectangular domain:

$$\begin{cases} \Delta u(x,y) = f(x,y), \ (x,y) \in \Omega = [0,a] \times [0,b], \\ \frac{\partial u(x,y)}{\partial \vec{n}} = 0, \ (x,y) \in \partial\Omega. \end{cases} \tag{1}$$

The important theoretical results about this problem are

- the equation can only have a solution for $\int\limits_{\Omega} f(x,y)d(x,y) = 0$;
- if u is a solution, then obviously $\tilde{u} = u + c$ is also a solution for any constant c.

2 Discretization and SOR

For discretization we assume that the domain is covered with a regular grid G which consists of $i_{max}+2$ by $j_{max}+2$ cells (including boundary layer) with meshsizes $\delta_x = a/i_{max}$ and $\delta_y = b/j_{max}$ (which might be different):

$$G = \{((i-0.5)\delta_x, (j-0.5)\delta_y \ ) \mid 0 \le i \le i_{max+1}, \ 0 \le j \le j_{max+1}\}.$$

Figure 1 shows the structure of such grid.

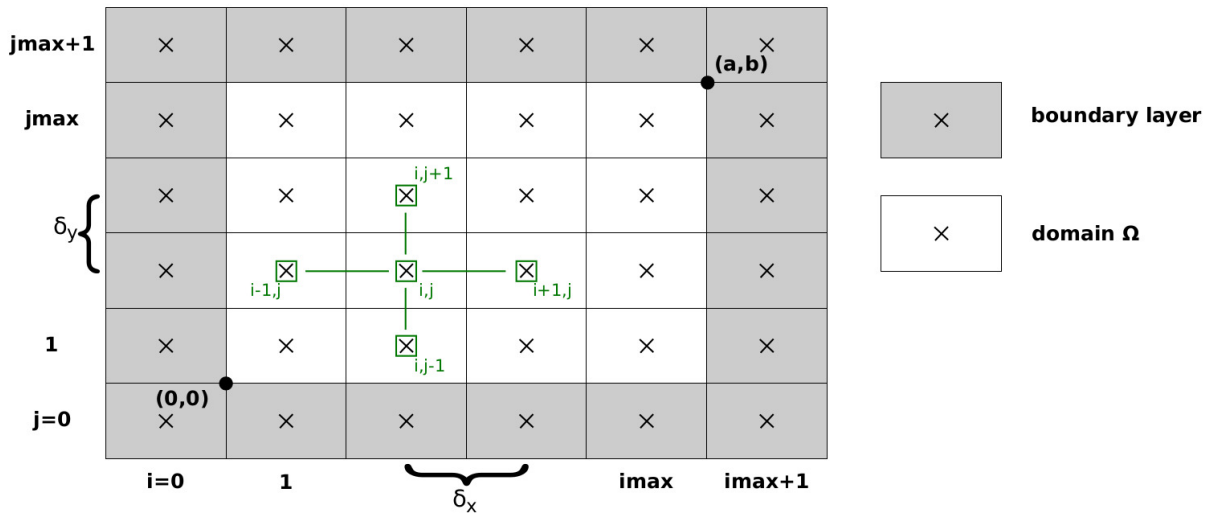

Figure 1: An example of a regular grid G, where the grid points are located at the cell centers.

Let us denote u^h a grid function, which is a discrete analog of function u and that satisfies equations (1) at every grid point in the domain. Then the following approximation of the Laplacian is used:

$$(\Delta u)_{i,j} \approx \frac{u_{i+1,j} - 2u_{i,j} + u_{i-1,j}}{\delta_x^2} + \frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{\delta_y^2} =: (\Delta^h u)_{i,j}. \tag{2}$$

As one can see, in order to evaluate the laplacian at a point, one has to know values at four neighbour points. This approximation is also known as a 5-point stencil.

At every inner point of the domain the grid function has to satisfy $\Delta^h u^h = f^h$, where $f^h_{i,j} = f(x_i, y_j)$. That gives one equation for each inner point. To handle boundaries, we rewrite the conditions from (1) as follows:

$$\frac{\partial u}{\partial \vec{n}} = \nabla u \cdot \vec{n} = \frac{\partial u}{\partial x} n_x + \frac{\partial u}{\partial y} n_y = 0.$$

Then we approximate the partial derivatives:

$$\left(\frac{\partial u}{\partial x}\right)_{i,j} \approx \frac{u_{i,j} - u_{i-1,j}}{\delta_x}, \text{ or } \left(\frac{\partial u}{\partial x}\right)_{i,j} \approx \frac{u_{i+1,j} - u_{i,j}}{\delta_x}$$

depending at which boundary the computations are done. The same applies for y-direction. For points at the right boundary, for instance, $\vec{n} = (1, 0)$, thus $\frac{\partial u}{\partial \vec{n}} = \frac{\partial u}{\partial x}$. In the discrete form we obtain the following simple relation:

$$\frac{u_{i_{max}+1,j} - u_{i_{max},j}}{\delta_x} = 0 \;\Rightarrow\; u_{i_{max}+1,j} = u_{i_{max},j}, \tag{3}$$

while for the left one we get $u_{0,j} = u_{1,j}$.

So now one can eliminate boundary conditions by inserting relations of kind (3) to the discrete equation for all the point which have boundary cells as neighbours. In this case the 5-point stencil becomes 4-point close to boundary and 3-point in the corner cells. In turn, we obtain a linear system $Au^h = f^h$, where u^h has $i_{max} \cdot j_{max}$ elements lying within the domain and A is a matrix which approximates the Laplacian.

There are two important theoretical results regarding this linear system:

- the linear system does only have a solution if the sum over the elements of the right-hand side f^h is 0. This guarantees that vector f^h is in the range of A. This corresponds to the condition $\int\limits_\Omega f(x, y)d(x, y) = 0$;
- the matrix A is singular, hence the solution is not unique. This, in turn, corresponds to the fact the if u is a solution, then $u + c$ is also a solution for any constant c.

One can iteratively solve the discrete analog of system (1) starting from an initial guess $u^{(0)}$:

1. Copy the the values of inner points touching the boundary to the boundary layer. By this, the boundary conditions are satisfied.

2. Perform an iteration of Successive Over-Relaxation on the inner points:

$$u^{new}_{i,j} = (1 - \omega)u^{old}_{i,j} + \omega \left(\frac{2}{\delta_x^2} + \frac{2}{\delta_y^2}\right)^{-1} \left[\left(\frac{u^{type}_{i+1,j} + u^{type}_{i-1,j}}{\delta_x^2} + \frac{u^{type}_{i,j+1} + u^{type}_{i,j-1}}{\delta_y^2}\right) - f^h_{i,j}\right], \tag{4}$$

   where $type = old$ or new depending on whether the point has been already updated or not.

3. Satisfy the boundary conditions by copying the values of inner points touching the boundary to the boundary layer, exactly as in 1.

4. Compute the residual of the discrete equation

$$\left(r^h\right)_{i,j} = f^h_{i,j} - \left(\Delta^h u^{new}\right)_{i,j}$$

and its L^2-norm:

$$\|r^h\|_2 := \sqrt{\frac{1}{i_{max} \cdot j_{max}} \sum_{i=1}^{i_{max}} \sum_{j=1}^{j_{max}} \left(r^h_{i,j}\right)^2}. \tag{5}$$

5. If the computed norm of the residual is not small enough, go to step 1.

For most problems the convergence rate depends heavily on the choice of the relaxation parameter ω. The most common choice is $\omega \in [1.7, 1.9]$, while $\omega = 1$ will transform (4) into Gauss-Seidel method. Note that sometimes it might take too much time to achieve the desired norm of residual, and you can add a second stopping criterion like the maximum number of iterations reached.

Since the solution is computed up to a constant, it might be a good idea to normalize it around some point, zero for example. It can be done after every k-th iteration. To do normalization, one computes the average value of the array and subtracts it from every element. The normalization is to be done before computing the residual.

3 Parallelization

The SOR iterations (4) can be parallelized within shared memory model. For this, so-called red-black colouring of the cells is used to decouple the iteration into two independent operations. An example can be found at Figure 2.

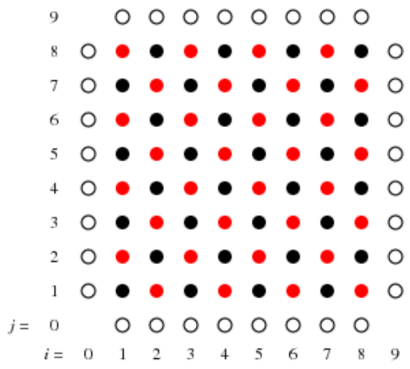

Figure 2: An example of red-black colouring in 2D.

One can notice that updating any red point using 5-point stencil will require only access to its black neighbours, while update of any black point needs only its red neighbours. This can be exploited for parallelization. The straightforward solution is to update several points of the same colour simultaneously with several OpenMP threads. Figure 3 illustrates how it can be done.

4 Tasks

1. Implement the concept of 2D array with boundary layer which was described in section 2). Note that index i follows x-axis, and j follows y-axis. Implement a function for displaying the array. For testing purposes, initialize a 10-by-10 array (not including boundaries), fill the elements corresponding to inner domain points with 1, boundary layer with 0, set the element (3,4) to 5 and include the output of your print function to the report. Do not forget to add the comments to the source code.

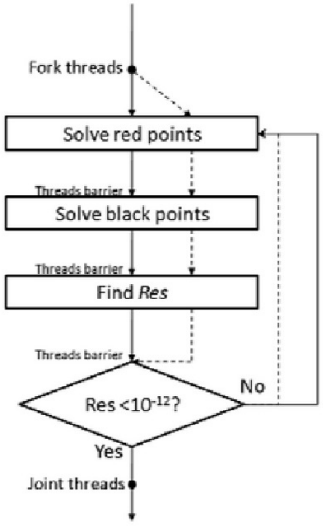

Figure 3: An example of OpenMP multithreading for SOR iteration.

2. Implement the SOR algorithm in serial way, test it on a unit square, with random initial guess and $f(x, y) = \sin(2\pi x)$. Do not forget to add the comments to the source code. Test the method with different grid sizes, do some tests for $i_{max} \neq j_{max}$. Plot the solution in Matlab or similar program and add to the report. After each iteration, print out the current iteration number and the current residual norm. In the end, print out the total number of iterations. Do several tests with varying ω. How does it influence the results?

3. Parallelize the serial code you wrote in the previous task with OpenMP. Use red-black colouring. Make sure that it produces correct results. Test the code with different number of threads on a cluster. Do tests for high enough resolution of the grid. Plot the speedup and explain its behaviour.

4. **Challenge (not compulsory).** Implement a hybrid version of SOR algorithm, which combines MPI and OpenMP techniques. In this case the domain is partitioned into smaller subdomains, which are handled by distinct nodes. Within the node, computations are done using OpenMP. You will have to handle the inter-node data exchange and find an appropriate pattern for that. Explain what you did, test the code with different number of processes/threads, plot the speedups and include them to the report.

5 Report

Present your experiments with relevant figures, tables, reflections and explaining text, i.e., submit a short informal report including your source codes in one PDF file. Submit also the source codes as c-files so that we can compile and run them if necessary.

The deadline is 2016-03-11 23:59.