Technische Universität München
Institut für Informatik
Ao Mo-Hellenbrand
Alexander Pöppl

Summer 2016
Worksheet 1

Future Trends in Computing Lab Course

Worksheet 1

Computation Fluid Dynamics with Navier-Stokes Equations and Parallelization

Deadline: May 9th, 2016, 2:00 PM

1 Navier-Stokes Equations

Non-stationary (incompressible) viscous fluid flow is described by the Navier-Stokes equations. For simplicity, we restrict our considerations to the two-dimensional case and carry out our analysis in Cartesian coordinates. The quantities to be computed are

- u, the fluid velocity in x–direction,
- v, the fluid velocity in y–direction,
- p the pressure.

The Navier-Stokes equations consist of two *momentum equations*

$$\frac{\partial u}{\partial t} + \frac{\partial (u^2)}{\partial x} + \frac{\partial (uv)}{\partial y} = -\frac{\partial p}{\partial x} + \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) + g_x, \tag{1}$$

$$\frac{\partial v}{\partial t} + \frac{\partial (uv)}{\partial x} + \frac{\partial (v^2)}{\partial y} = -\frac{\partial p}{\partial y} + \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) + g_y \tag{2}$$

and the *continuity equation*

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0, \tag{3}$$

where g_x and g_y denote external forces such as gravity. The dimensionless quantity Re is called *Reynolds number*; it characterizes the fluid flow and defines whether the flow is laminar or turbulent.

In order to solve this system of partial differential equations, both initial and boundary conditions are required.

Initial conditions: At the initial time $t = 0$, initial conditions $u = u_0(x, y)$ and $v = v_0(x, y)$ are prescribed throughout the entire computational domain. The velocity field needs to satisfy the continuity equation (3).

Boundary conditions: Besides, supplementary conditions holding at all four boundaries of the region at all times are required, yielding an *initial-boundary value problem.*

In order to formulate the boundary conditions, let us denote the velocity component perpendicular to the boundary (in normal direction) as $w_\perp$ and the component parallel to the boundary (in tangential direction) as $w_\parallel$. The derivatives in the normal direction will be denoted as $\partial w_\perp/\partial \mathbf{n}$ and $\partial w_\parallel/\partial \mathbf{n}$, respectively.

Then, for the vertical boundary, we have

$$w_\perp = u, \qquad w_\parallel = v, \qquad \frac{\partial w_\perp}{\partial \mathbf{n}} = \frac{\partial u}{\partial x}, \qquad \frac{\partial w_\parallel}{\partial \mathbf{n}} = \frac{\partial v}{\partial x},$$

and for the horizontal boundary

$$w_\perp = v, \qquad w_\parallel = u, \qquad \frac{\partial w_\perp}{\partial \mathbf{n}} = \frac{\partial v}{\partial y}, \qquad \frac{\partial w_\parallel}{\partial \mathbf{n}} = \frac{\partial u}{\partial y}.$$

For the points (x, y) on the rigid boundary $\Gamma := \partial\Omega$, one can formulate the following boundary conditions:

1. No-slip condition: $w_\perp(x, y) = 0$, $w_\parallel(x, y) = 0$.
   (the fluid velocity vanishes at the boundary)
2. Free-slip condition: $w_\perp(x, y) = 0$, $\partial w_\parallel(x, y)/\partial \mathbf{n} = 0$.
   (there is no friction at the boundary)
3. Inflow condition: $w_\perp(x, y) = w_\perp^0$, $w_\parallel(x, y) = w_\parallel^0$.
   (the velocity components $w_\perp^0, w_\parallel^0$ are prescribed)
4. Outflow condition: $\partial w_\perp(x, y)/\partial \mathbf{n} = 0$, $\partial w_\parallel(x, y)/\partial \mathbf{n} = 0$.

If only the velocities and not their normal derivatives are given at the boundaries, then the surface (line) integral over the velocities normal to the boundaries should vanish, i.e.

$$\int_\Gamma \begin{pmatrix} u \\ v \end{pmatrix} \cdot \mathbf{n} ds = 0.$$

This is required in order to satisfy the continuity equation (3).

2 Domain Discretization: the Grid

We use a finite difference method based on a *staggered Cartesian grid*, that is the unknown variables u, v, and p are located at different positions on our grid. The reference grid subdivides the whole region into cells characterized by index (i, j) corresponding to the rectangle $[(i-1)\,\delta x, i\,\delta x] \times [(j-1)\,\delta y, j\,\delta y]$. The pressure p is associated to the cell's midpoint, the velocity u to the midpoint of vertical cell edges, and the velocity v to the midpoints of the horizontal cell edges; see Fig. 1 for the assignment of the index (i, j) to the values of u and v.

As a consequence, the boundary of our rectangular domain does not contain values of all unknown functions (vertical boundary lines contain only u-values, whereas level lines only contain v-values; p-values are never located on boundary lines). For this reason, one more boundary strip of grid cells is introduced (see Figure 2), so that the boundary conditions may be satisfied by averaging the neighboring grid points on either side (see below).

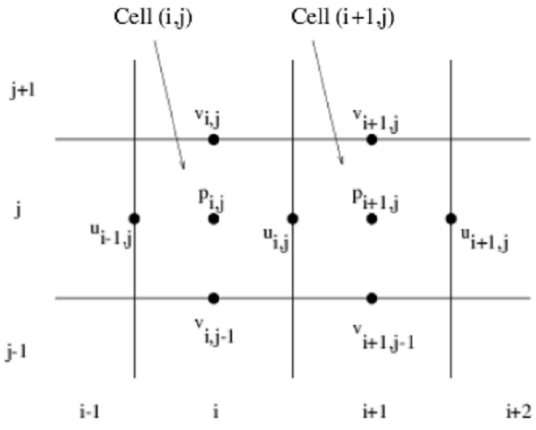

Figure 1: Staggered Grid for u, v, and p.

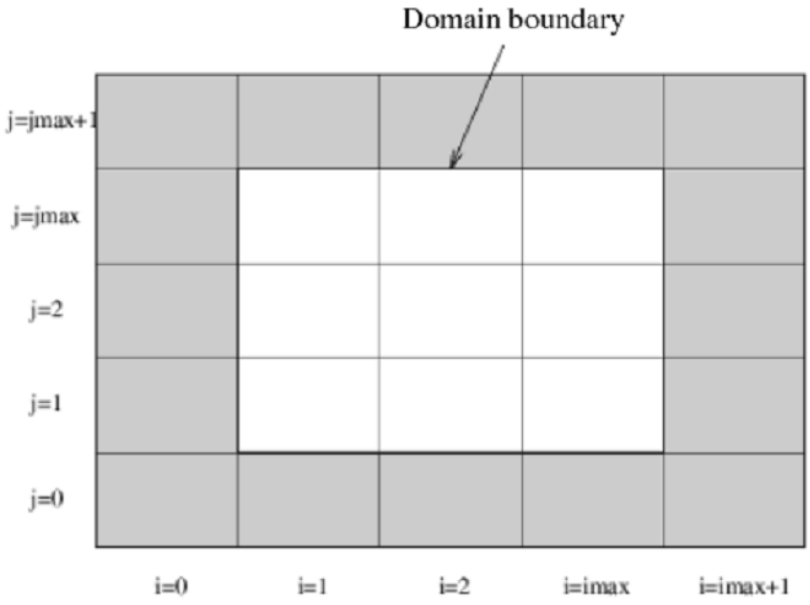

Figure 2: Domain with boundary cells.

3 The Algorithm

We will skip all details on the spatial and time discretization treatments in this handout, and jump directly to the main algorithm:

```
Setup the domain geometry
Set t := 0, n := 0
Assign initial values to u, v, p
Setup boundaries to u, v, p

While (t < t_end)
   1. Compute time step size δt of the current iteration
   2. Compute F^(n) and G^(n), and
        update the non-fluid cells in domain to F, G
   3. Compute pressure p^(n+1): solver the pressure equation using a
SOR solver...
        Set it := 0
        While it < itmax and res > ε
          Perform a SOR iteration according to (4) using the
            provided function and retrieve the residual res
          it := it + 1
        End While
        update the non-fluid cells in domain to p
   4. Compute u^(n+1) and v^(n+1)
   5. Increment simulation time: t := t + δt, n := n + 1
   6. Output of u, v, p values for visualization, if necessary
End Whild
```

4 Task 1: Implementation of serial SOR soler

You are given a class named **NSSim**, in which all constants/variables are setup and methods are implemented except for the SOR solver, which is needed to solve the pressure equation described in the above algorithm in step 3. Your goal is to implement this solver, based on the following ...

The SOR (Successive Over-Relaxation) method, where the iteration step is given by the following loop over all cells:

$$
\begin{array}{l}
it = 1, \ldots, itmax \\
\quad j = 1, \ldots, jmax \\
\quad\quad i = 1, \ldots, imax
\end{array}
$$

$$
\begin{aligned}
p_{i,j}^{it+1} \;:=\; & (1-\omega)\, p_{i,j}^{it} + \\
& \frac{\omega}{2(\frac{1}{(\delta x)^2} + \frac{1}{(\delta y)^2})} \left(\frac{p_{i+1,j}^{it} + p_{i-1,j}^{it+1}}{(\delta x)^2} + \frac{p_{i,j+1}^{it} + p_{i,j-1}^{it+1}}{(\delta y)^2} - rs_{i,j} \right)
\end{aligned} \tag{4}
$$

where $rs_{i,j}$ is given by

$$
rs_{i,j} = \frac{1}{\delta t} \left(\frac{F_{i,j}^{(n)} - F_{i-1,j}^{(n)}}{\delta x} + \frac{G_{i,j}^{(n)} - G_{i,j-1}^{(n)}}{\delta y} \right), \tag{5}
$$

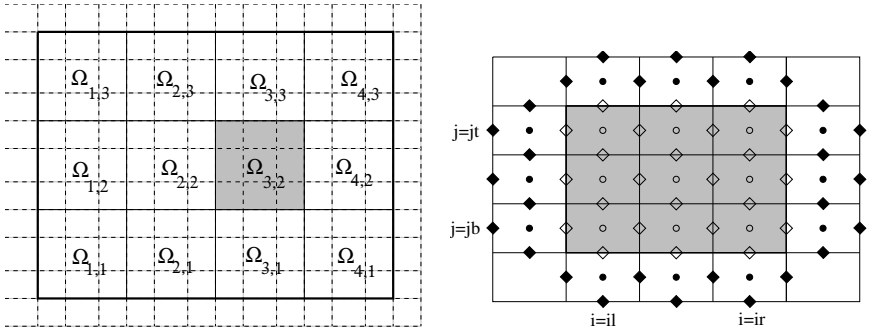

Figure 3: **left:** Partitioning of the domain Ω (`iproc`=4, `jproc`=3) **right:** Unknowns and boundary values for the subdomain $\Omega_{3,2}$

and ω is a parameter (relaxation factor), which must be chosen from the interval $]0, 2[$ (often the value $\omega = 1.7$ is used). For $\omega = 1$, the method becomes identical to Gauss-Seidel. The iteration process should either stop once the maximum number of iterations, $itmax$, is reached or when the absolute residual drops below a predefined tolerance value ϵ. As a starting value for the iteration process to calculate the pressure $p^{(n+1)}$, the pressure values of the previous time step n are used.

$$res := \left(\sum_{i=1}^{imax} \sum_{j=1}^{jmax} \left(\frac{p_{i+1,j} - 2p_{i,j} + p_{i-1,j}}{(\delta x)^2} + \frac{p_{i,j+1} - 2p_{i,j} + p_{i,j-1}}{(\delta y)^2} - rs_{i,j} \right)^2 / (imax \cdot jmax) \right)^{1/2} \tag{6}$$

As a result of Task 1, you should have a fully functional **NSSim** class, which runs the simulation with any given resolution (lowest resolution: $N_x = 27, N_y = 15$). A Main method is included, in which **NSSim** is instantiated and the public simulation driver $run(N_x, N_y)$ is called.

5 Task 2: Parallelization of the Navier-Stokes simulation code

Let's turn to the parallelization of the sequential code. We begin by dividing the base domain Ω for the flow calculation along the grid lines into i_{proc} parts in the x direction and j_{proc} parts in the y direction, so that we get $i_{proc} \cdot j_{proc}$ rectangular subdomains $\Omega_{ip,jp}$, $ip = 1, \ldots, i_{proc}$, $jp = 1, \ldots, j_{proc}$. This decomposition is represented in Figure 3 where the grid lines of the staggered grid are shown as dashed lines and the subdomain boundaries as solid lines. In each process, the pressure and velocity values in the interior of the associated subdomain are calculated. To save an additional communication step, velocities lying exactly on a subdomain boundary are computed by both processes involved.

Thus, a process treating a subdomain $[(il-1)\,\delta x, ir\,\delta x] \times [(jb-1)\,\delta y, jt\,\delta y]$ requires the values shown in Figure 3. The values lying on points marked with white symbols must be computed by the process itself, whereas the values in the black points only serve as boundary data to determine the values in the interior and on the subdomain boundary. These boundary values are – unless they fall outside the total domain Ω – calculated by the neighboring processes. The boundary values must therefore

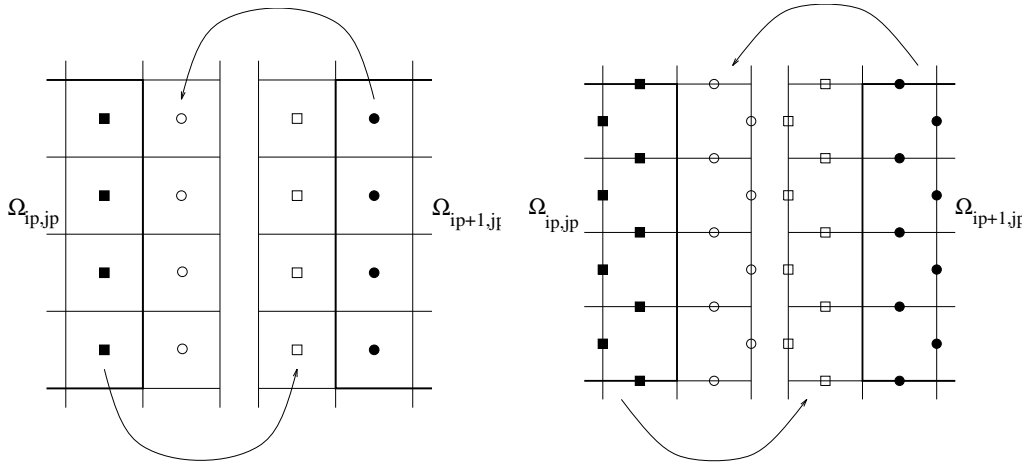

Figure 4: **left:** Exchange of pressure values **right:** Exchange of velocity values

be sent, in an appropriate form and with a proper frequency, by the neighboring processes. The following communication steps have to be implemented:

- After each SOR step in the pressure iteration, the pressure values located in the boundary strips must be exchanged as represented in Figure 4, such that the computation can proceed in the subdomains with the up-to-date boundary values. To avoid network overloading with too many simultaneous data bursts, the data exchange might be performed in four steps – to the left, to the right, up, and down. All relevant data should be sent to the respective neighbor as block of data.
- Following the data exchange of one SOR smoothing step, the termination criterion has to be evaluated: Therefore, each node computes its contribution to the global residual and, afterwards, sends the partial sum to a single process (master process). This process then decides whether to terminate the pressure iteration (tolerance ε achieved or the maximum number of iterations `itmax` exceeded) and broadcasts a message to all other processes saying whether to perform the next iteration step or to terminate the pressure iteration.
- At the end of the pressure iteration the current valid pressure values are contained in the boundary strips, so the velocity values on the subdomain boundaries can be updated in `calculate_uv` without any new communication. However, following `calculate_uv`, the updated velocity values must be exchanged as depicted in Figure 4 to enable the calculation of F and G in the next time step.
- Finally, the new time step `dt` must be determined. To find it, the maximum absolute values of `U` und `V` must be first determined in `calculate_dt` for each subdomain, and these local maxima should be sent to the master process. The latter determines the global maxima and disseminates them over all processes. After that, the CFL condition (see Worksheet 1, (13)) returns the same value of time step `dt` on each partial process. This procedure is, in principle, analogous to that of residual computation.

We may summarize all the steps in the following algorithm.

```
Setup the domain geometry
Set t := 0, n := 0
Assign initial values to u, v, p
Setup boundaries to u, v, p

While (t < t_end)
   1. Compute F^(n) and G^(n), and
       update the non-fluid cells in domain to F, G
   2. Compute pressure p^(n+1): solver the pressure equation using a
SOR solver...
       Set it := 0
       While it < itmax and res > ε
         Perform a SOR iteration according to (4) using the
         Exchange the pressure values in the ghost layer strips
         Compute the partial residual sum and send it to the master
process
         The master computes the global residual norm and
broadcasts it to all processes.
         it := it + 1
       End While
       update the non-fluid cells in domain to p
   3. Compute u^(n+1) and v^(n+1)
   4. Exchange the velocity values in the ghost layer strips
   5. Each processor compute local maximum of u^(n+1) and v^(n+1) and
send to master
       Master computes the global maximum of of u^(n+1) and v^(n+1) and
broadcast them
       Each process computes the global δt for the NEXT iteration
       6. Increment simulation time: t := t + δt, n := n + 1
   7. Output of u, v, p values for visualization, if necessary
End Whild
```

Your task is to implement the above distributed-memory parallelization scheme, with local shared-memory parallelization features to boost the performance. Measure and compare the execution time of your sequential code and your final parallelized version.

Determine the number of floating point operations executed in one execution of the SOR solver. This number may depend on the number of iterations the SOR solver needs to converge. Measure the time spent executing the solver as well. How many flops/sec does this translate to? Compare it with the peak floating point performance of your machine.