THE UNIVERSITY OF WARWICK

High Performance Computing

Course Notes

Coursework 2

Dr Ligang He

Problem Domain

Coursework taken from the field of Computational Fluid Dynamics (CFD)

- Fluid dynamics based on three fundamental principles: (i) mass is conserved; (ii) Newton's second law (acceleration of object dependent on net force acting on object and mass of object); (iii) energy is conserved
- These are usually expressed as partial differential equations, showing how velocity and pressure are related, etc (called governing equations).
- In the governing equations the coordinates and time are independent variables while velocity and pressure are dependent variables

Computational Fluid Dynamics is the science of finding a numerical solution to the governing equations of fluid flow, over the discretized space or time

Governing Equations

Navier–Stokes Equations
3 – dimensional – unsteady

Glenn Research Center

Coordinates: (x,y,z)
Velocity Components: (u,v,w)

Time : t
Density: ρ
Total Energy: Et

Pressure: p
Stress: τ

Heat Flux: q
Reynolds Number: Re
Prandtl Number: Pr

Continuity:

$$\frac{\partial \rho}{\partial t} + \frac{\partial(\rho u)}{\partial x} + \frac{\partial(\rho v)}{\partial y} + \frac{\partial(\rho w)}{\partial z} = 0$$

X – Momentum:

$$\frac{\partial(\rho u)}{\partial t} + \frac{\partial(\rho u^2)}{\partial x} + \frac{\partial(\rho uv)}{\partial y} + \frac{\partial(\rho uw)}{\partial z} = -\frac{\partial p}{\partial x} + \frac{1}{Re_r}\left[\frac{\partial \tau_{xx}}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{xz}}{\partial z}\right]$$

Y – Momentum:

$$\frac{\partial(\rho v)}{\partial t} + \frac{\partial(\rho uv)}{\partial x} + \frac{\partial(\rho v^2)}{\partial y} + \frac{\partial(\rho vw)}{\partial z} = -\frac{\partial p}{\partial y} + \frac{1}{Re_r}\left[\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \tau_{yy}}{\partial y} + \frac{\partial \tau_{yz}}{\partial z}\right]$$

Z – Momentum

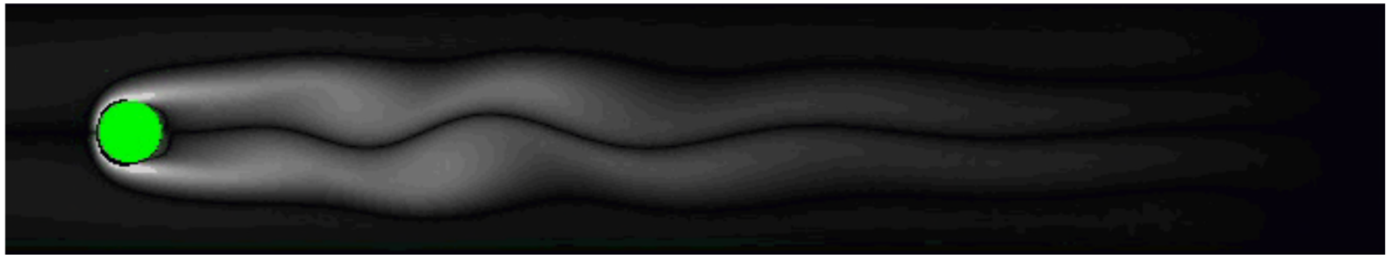
$$\frac{\partial(\rho w)}{\partial t} + \frac{\partial(\rho uw)}{\partial x} + \frac{\partial(\rho vw)}{\partial y} + \frac{\partial(\rho w^2)}{\partial z} = -\frac{\partial p}{\partial z} + \frac{1}{Re_r}\left[\frac{\partial \tau_{xz}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \tau_{zz}}{\partial z}\right]$$

Energy:

$$\frac{\partial(E_T)}{\partial t} + \frac{\partial(uE_T)}{\partial x} + \frac{\partial(vE_T)}{\partial y} + \frac{\partial(wE_T)}{\partial z} = -\frac{\partial(up)}{\partial x} - \frac{\partial(vp)}{\partial y} - \frac{\partial(wp)}{\partial z} - \frac{1}{Re_r Pr_r}\left[\frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z}\right]$$

$$+ \frac{1}{Re_r}\left[\frac{\partial}{\partial x}(u\,\tau_{xx} + v\,\tau_{xy} + w\,\tau_{xz}) + \frac{\partial}{\partial y}(u\,\tau_{xy} + v\,\tau_{yy} + w\,\tau_{yz}) + \frac{\partial}{\partial z}(u\,\tau_{xz} + v\,\tau_{yz} + w\,\tau_{zz})\right]$$

CFD code



The code in the coursework calculates the velocity and pressure of a 2D flow

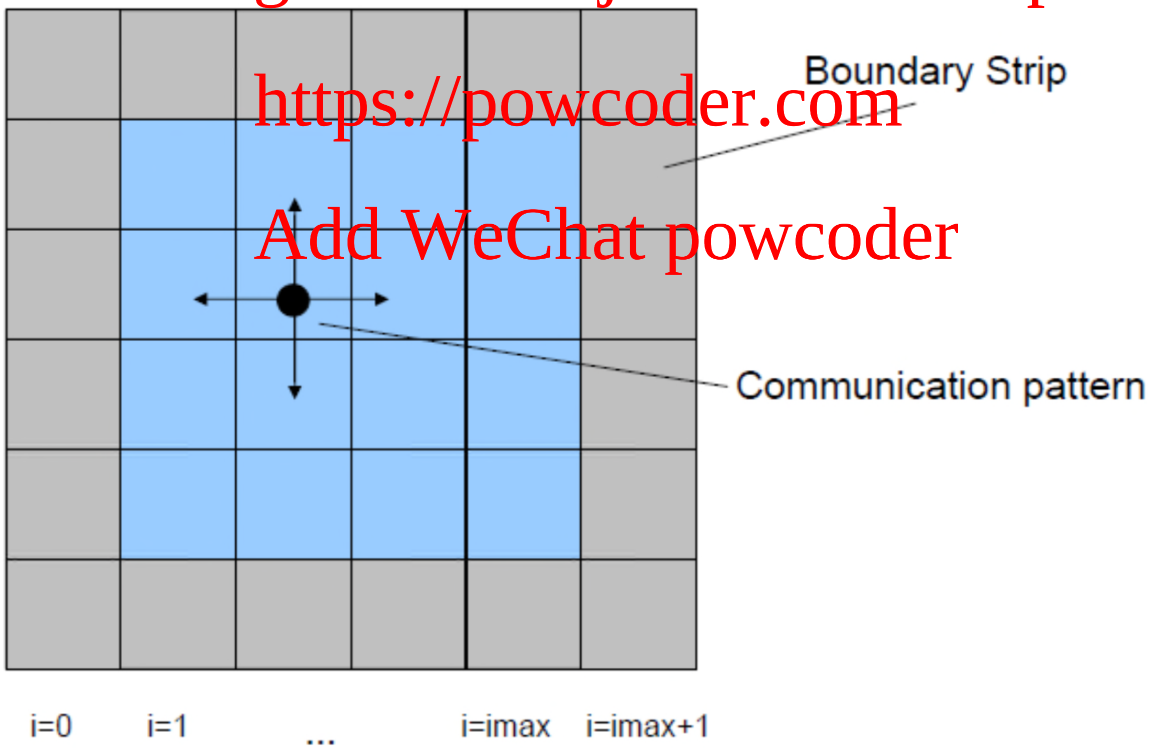
The code writes a binary file that contains solution values

Currently the code is sequential; the purpose of the coursework is to parallelize the code

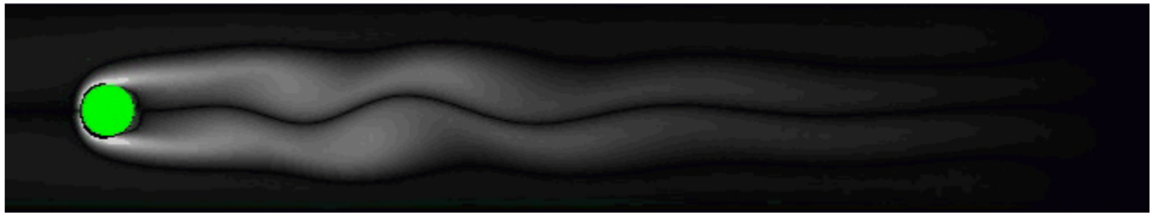
Data and stencil

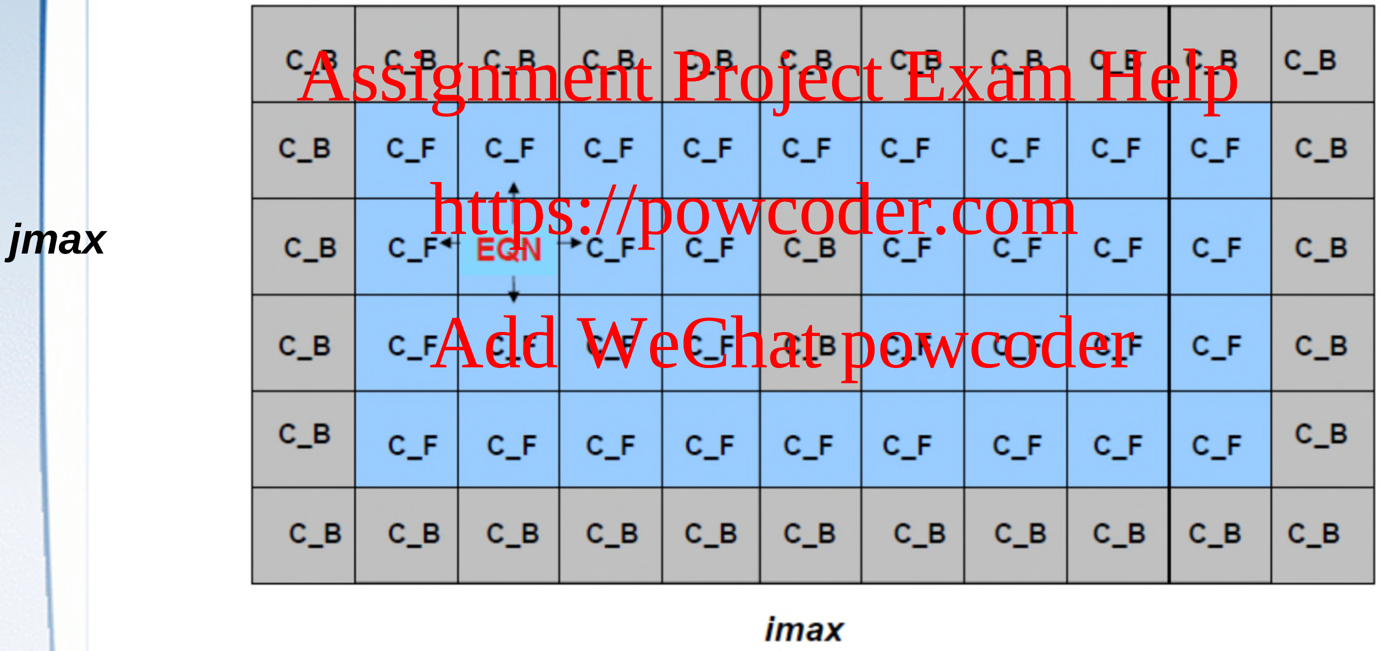
The area represented as a 2D Grid (discretize)

Calculate one point in each cell



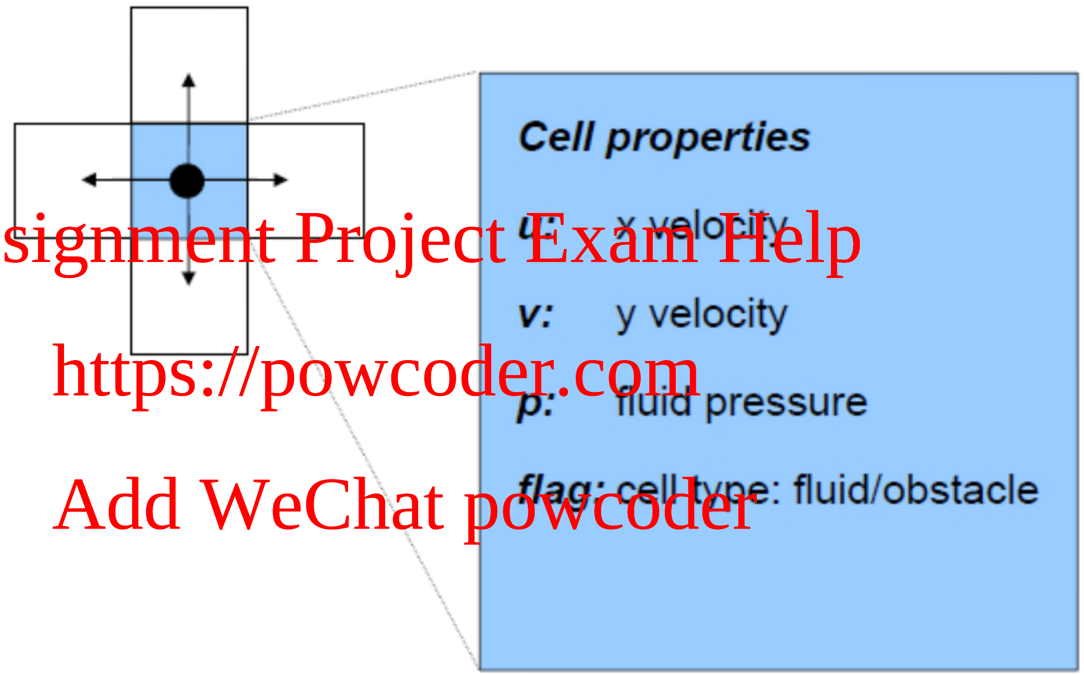
CFD code





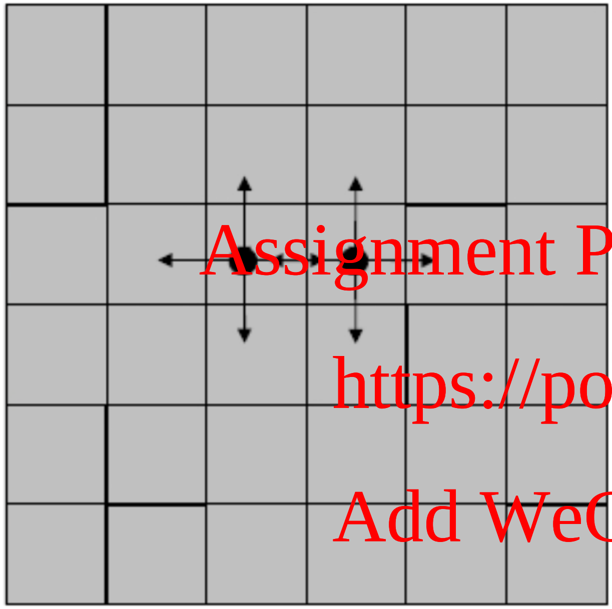
There are two types of cell: fluid (C_F) and obstacle (C_B)

Data and stencil



- **The properties of a cell in the grid**
- **Communication pattern: using a five-point stencil to calculate a point**

Numerical method for solving the governing equations



Initialize each cell value

Check if the solution satisfies the governing equations

If not, generate the new solution based on a stencil of current solutions from neighbouring cells

Iterations are advanced until the termination condition is met

General Steps for using the numerical method to solve the linear equations

- **Aim: solve** AΦ=B

- Step 1: Guess a initial solution Φ^0

- Step 2: Check if convergence is reached by checking the residual $B-A\Phi^i$<tolerance

- Step 3: For Φ^i (i>=0), $\Phi^{(i+1)}= f(\Phi^{(i)})$, go to Step 2

e.g., $f(\Phi^{(i)})=\Phi^{(i)}+(B-A\Phi^{(i)})$

- **Key: repeating iterative steps and each step generates a better approximation of the solution**

Successive Over-Relaxation(SOR)

- The SOR method can speed up convergence
- For a set of linear equations: AΦ=B

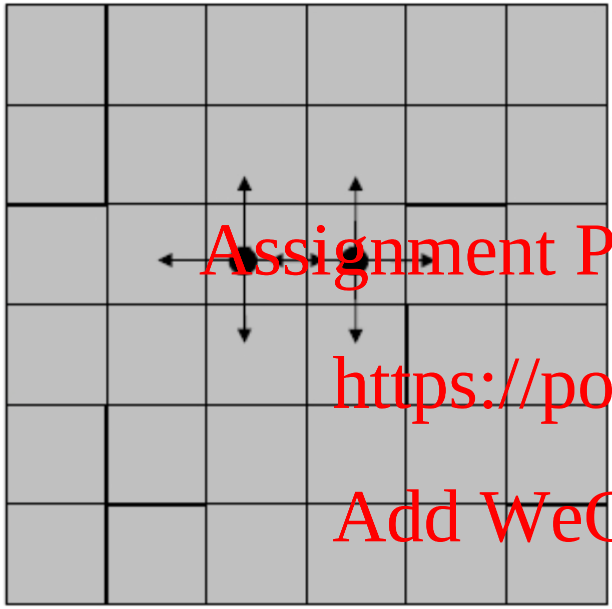
- let A=D+U+L, where D, L and U denote the diagonal, strictly lower triangular, and strictly upper triangular parts of A, respectively

- The successive over-relaxation (SOR) iteration is defined by the following

$$(D + \omega L)\phi^{(k+1)} = (-\omega U + (1 - \omega)D)\phi^{(k)} + \omega b. \qquad (*)$$

- Where values of ω > 1 are used to speedup convergence of a slow-converging process, while values of ω < 1 are often help establish convergence of diverging iterative process

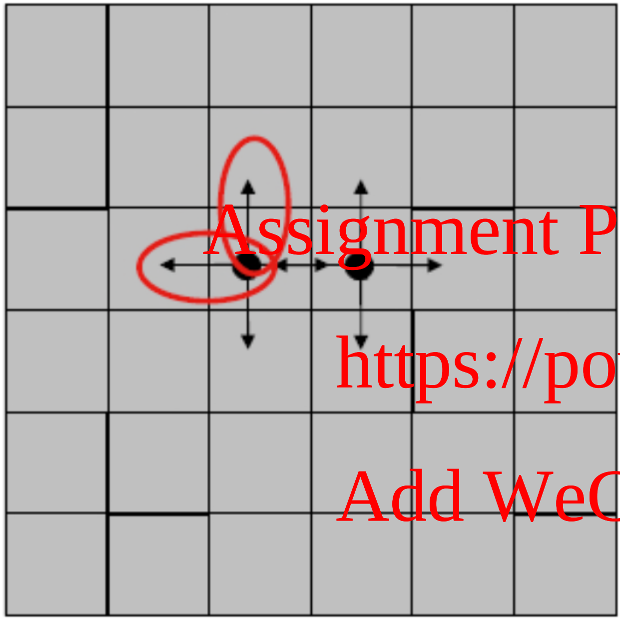
SOR



Successive Over-Relaxation (SOR) method is used to speed up convergence

$$(D + \omega L)\phi^{(k+1)} = (-\omega U + (1 - \omega)D)\phi^{(k)} + \omega b. \qquad (*)$$

SOR



- **SOR is originally written so that calculating p_new(i, j) depends on**
  - **1) a stencil of current solutions from neighbouring cells,**
  - **2) p_new(i-1, j),**
  - **3) p_new(i, j-1).**

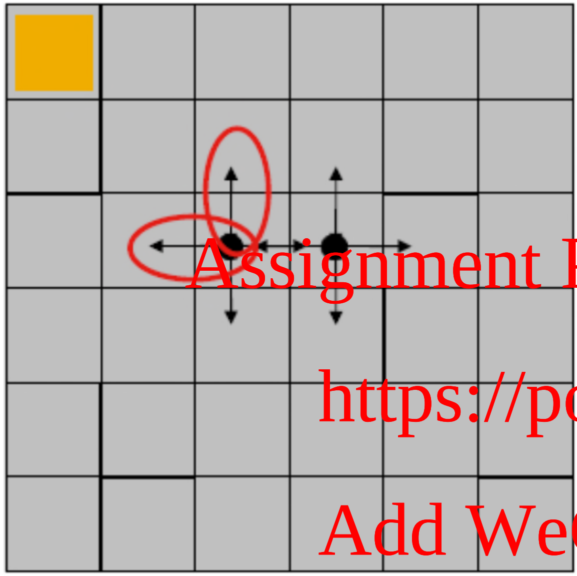
  for all i ≤ imax, j ≤ jmax, do
  $p_{ij}^{(k+1)} = f(P^{(k)}, p_{i-1,j}^{(k+1)}, p_{i,j-1}^{(k+1)})$

- **This introduces a chain of dependencies for each cell**

- **Fine on sequential machines: just calculate values from left to right, and top to bottom**

$P^{(k)}$ is the solution at the iteration k (P is a matrix [p_{ij}])

SOR



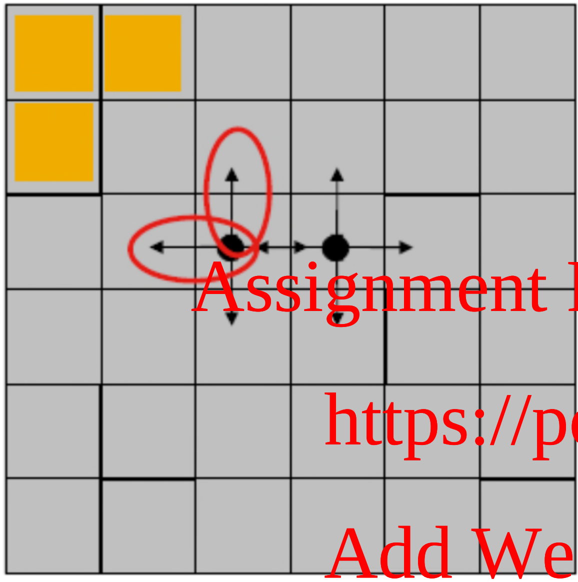
- **SOR is originally written so that calculating p_new(i, j) depends on**
  - **1) a stencil of current solutions from neighbouring cells,**
  - **2) p_new(i-1, j),**
  - **3) p_new(i, j-1).**

- **This introduces a chain of dependencies for each cell**

- **Fine on sequential machines: just calculate values from left to right, and top to bottom**

$P^{(k)}$ is the solution at the iteration k (P is a matrix [p_{ij}])

SOR



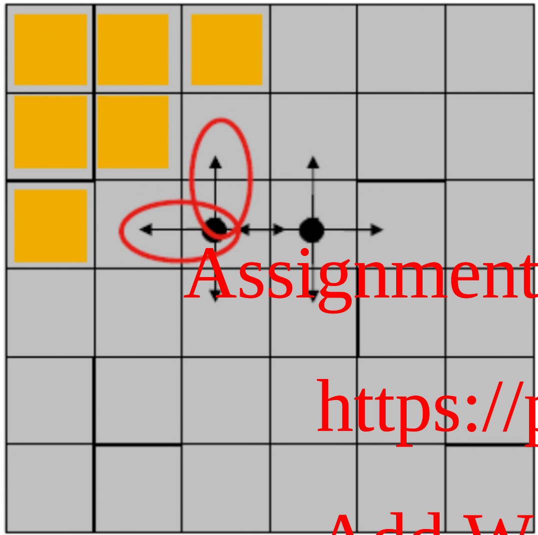
- **SOR is originally written so that calculating p_new(i, j) depends on**
  - **1) a stencil of current solutions from neighbouring cells,**
  - **2) p_new(i-1, j),**
  - **3) p_new(i, j-1).**

- **This introduces a chain of dependencies for each cell**

- **Fine on sequential machines: just calculate values from left to right, and top to bottom**

$P^{(k)}$ is the solution at the iteration k (P is a matrix [p_{ij}])

SOR



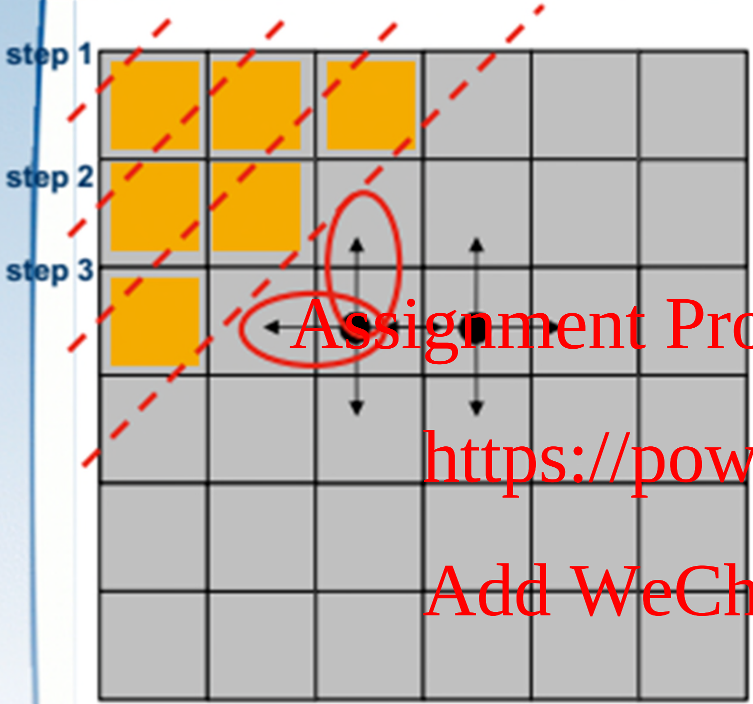
- **SOR is originally written so that calculating p_new(i, j) depends on**
  - **1) a stencil of current solutions from neighbouring cells,**
  - **2) p_new(i-1, j),**
  - **3) p_new(i, j-1).**

- **This introduces a chain of dependencies for each cell**

- **Fine on sequential machines: just calculate values from left to right, and top to bottom**

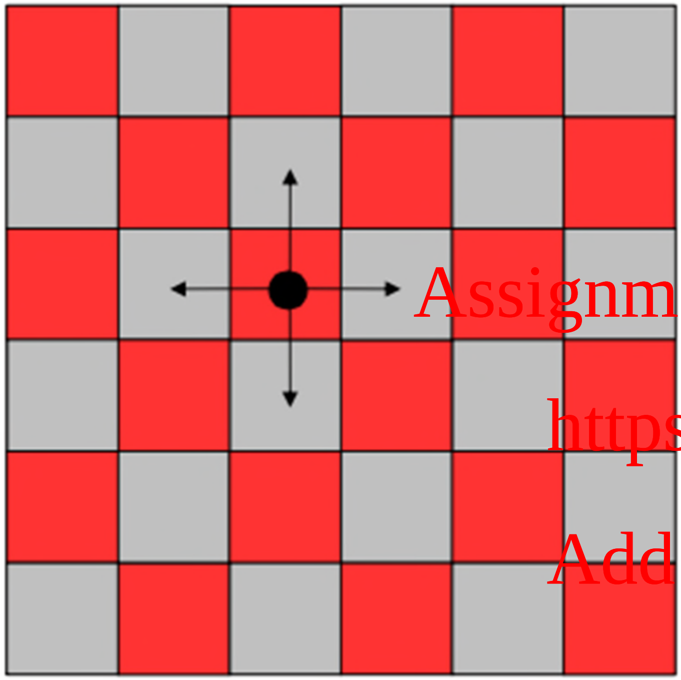
$P^{(k)}$ is the solution at the iteration k (P is a matrix [p_{ij}])

SOR



- This approach forms a “wave front”
- It does not make for a good parallel solution
- If we implement a parallel *wave-front* then the work is highly **unbalanced**
- The fundamental reason is that the number of cells to be computed varies in each step

Red/Black SOR



☺ Work can be balanced across processors
☺ Less communications

- Another solution that achieves the same result is *red/black ordering*
- Rewrite the SOR equations so the cells are divided into two populations.
- Red is calculated first (using **current black** values)

Red cells:

$$p_{ij}^{(k+1)}=f(p_{i-1,j}^{(k)}, p_{i,j-1}^{(k)}, p_{i+1,j}^{(k)}, p_{i,j+1}^{(k)})$$

- Then black is calculated using the just updated red values.

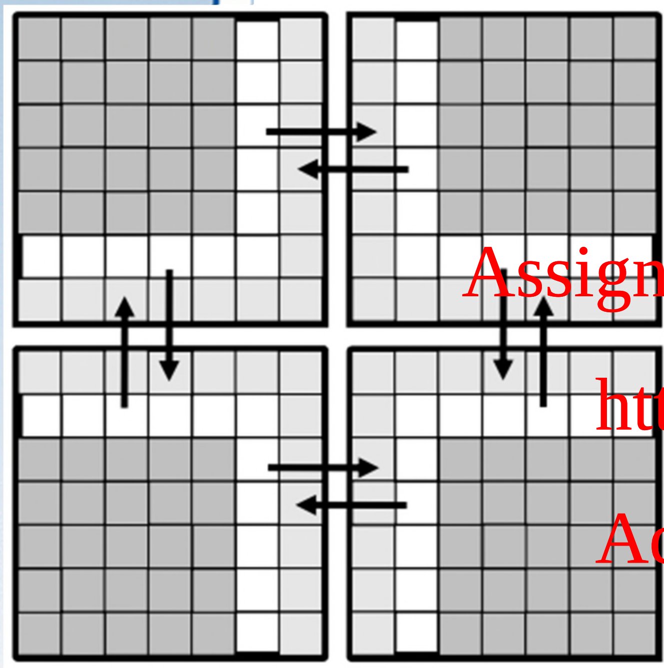
Black cells:

$$p_{ij}^{(k+1)}=f(p_{i-1,j}^{(k+1)}, p_{i,j-1}^{(k+1)}, p_{i+1,j}^{(k+1)}, p_{i,j+1}^{(k+1)})$$

Note: $p_{i-1,j}^{(k+1)}$, $p_{i,j-1}^{(k+1)}$, … are red cells

- Since the stencil for any cell is the cell itself and cells of opposite color, they can all be updated in parallel

Domain decomposition

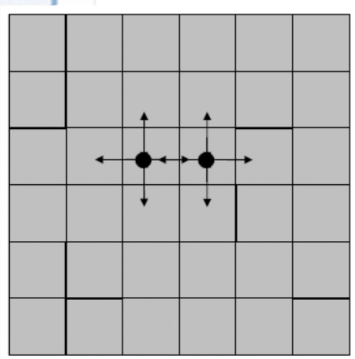


- Make each processor responsible for updating a block of cells

- At the end of each iteration it must send the cells at the edge of its domain to its neighbours, and receive a copy of the edge cells from its neighbours

  Why?

- You need to think whether you implement 1D or 2D decomposition



The main loop

for (t = 0.0; t< t_end; t = t + delt)
{

1.Calculate an approximate time-step size by seeing how much movement occurred in the last time-step. The discrete approximation is only stable when the maximum motion < 1 cell per time-step.

2.For each cell, compute a tentative new velocity *(f,g)* based on the previous *(u,v)* values. It takes as input the u, v and flag matrices, and updates the f and g matrices.

3.For each cell calculate the RHS of the pressure equation. This uses two *f* and *g* values. It takes as input the *f*, *g*, and flag matrices and updates the RHS matrix.

4.For the entire pressure matrix, use Red/Black SOR to solve the Poisson equation. This takes a large number iterations of the Red/Black process as shown in the slides:. It takes as input the current pressure matrix, flag matrix, and the RHS matrix and outputs a new pressure matrix

The main loop

5. For each cell, update the real *(u,v)* values based on the pressure matrix and the tentative velocity values *(f,g)*. It takes as input the pressure, *f, g* and flag matrices and updates the u and v matrices.

6. For each cell that is adjacent to an edge cell, the *(u,v)* values of the boundary cells are updated (by taking values from their neighbours). It takes as input the *u, v* and flag matrices and updates the *u* and *v* matrices

}

Note that it is not going to be worth parallelizing the whole program, just parts of it.

Assignment 2 – What to Do in General

- You will be given a serial program, called Karman
- You need to parallelize the Karman code and write a report
- 80% of the full mark comes from the parallelization using MPI
- 20% of the mark comes from the parallelization using both MPI and OpenMP (a hybrid approach), i.e., parallelizing the computations in one machine using OpenMP, while parallelizing the computations across machines using MPI

Assignment 2 – Detailed Requirements for Programming

- **Parallelize the Karman code, using a pure MPI approach (80% of marks) and a hybrid MPI-OpenMP approach (20% marks)**

- **Profiling which Karman functions are more time consuming**

- **Design your decomposition strategy, date exchange strategies between neighboring partitions, the MPI functions (e.g., collective operations)**

- **For a hybrid approach, design the parallelization strategy for OpenMP**

Assignment 2 – Detailed Requirements for Programming

- Benchmarking the execution time of your parallel program as you increase the number of processes

- If you develop a hybrid MPI-OpenMP implementation,
  - benchmark the runtime of OpenMP code
  - Compare the performance between OpenMP and MPI

- Ensure the simulation results are the same after the parallelization

- Your parallel program should contain the adequate comments as the evidence of good programming practice.

Assignment 2 – Requirements for Report

- **Profiling the functions to determine which functions are more time consuming; discuss the profiling results**

- **Discuss your MPI implementation, for example,**
  - your decomposition strategy and load balancing strategy;
  - your strategy of exchanging the boundary data between neighboring partitions;
  - collective operations you used;

- **Benchmark the change in execution times of your parallel program as you increase the number of processes; present the results in graph or table**

Assignment 2 – Requirements for Report

- **Analyze and discuss the performance of your parallel program**

- **If you develop the hybrid MPI-OpenMP implementation,**
  - **Discuss OpenMP implementation**
  - **Benchmark the runtime of the OpenMP codes and present the results**
  - **Discuss the performance comparison between the hybrid approach and the pure MPI approach**

- **Writing skills: pay attention to spelling, punctuation and grammar**

Assignment 2 – Submission

- **Submit a compressed folder (.zip) to Tabula; the folder should contain**
  - **your parallel code (the directory structure used in the project should not change)**
  - **your report (pdf format and place your report in the top level directory in the folder)**
- **Deadline: 12pm, March 14th, 2018**