CS410: Parallel Computing

Spring 2024

Nikhil Hegde
Milind Chabbi

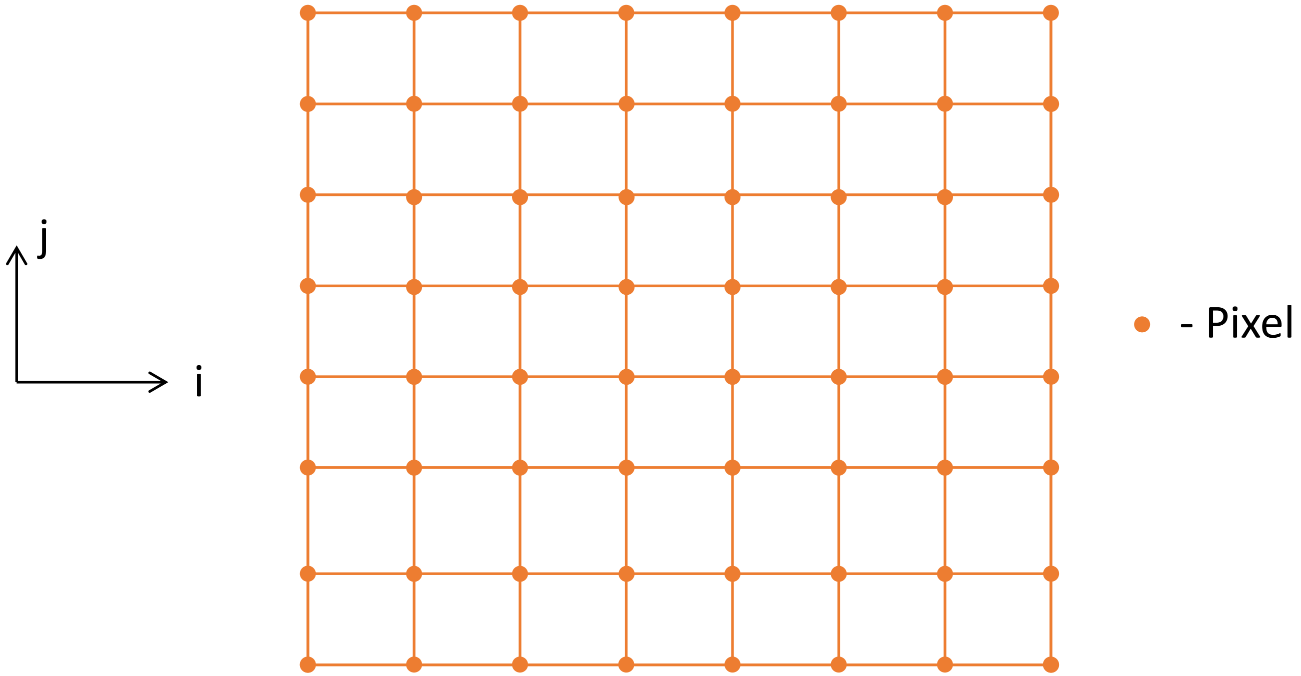
Parallel Algorithms and Applications

Embarrassingly Parallel Applications

- Application where:
  - A number of (almost) independent tasks
    - No or very little communication between tasks
  - Each task can be executed on a node
- Master-worker approach could be used
- Examples
  - Image Processing: e.g. blurring, scaling, rotation etc.
  - Computer Graphics: e.g. ray tracing
  - Monte Carlo method: e.g. estimation of pi
  - …

Interesting reads:
http://graphics.pixar.com/library/CurlyHairA/paper.pdf,
https://www.fxguide.com/fxfeatured/brave-new-hair/

Image Processing



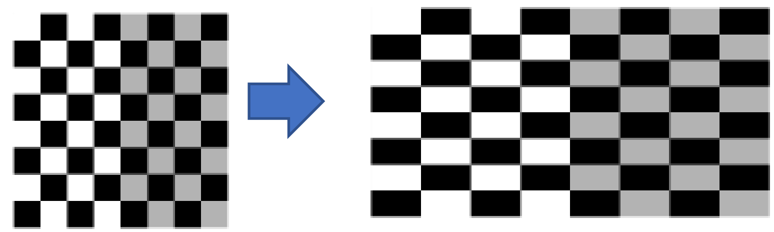
8 x 8 pixel array

Pixel

- 8-bits – 256 colors possible.
- 24-bits – More than 16 million colors possible
- Voxel – 3 dimensional image

Simple Image Processing

- Scaling
  - Scale the image by a factor λ_x in x – direction
    $x' = x \,.\, \lambda_x$
    $y' = y$
  - Scaling Matrix R: $\begin{bmatrix} \lambda_x & 0 \\ 0 & 1 \end{bmatrix}$



https://en.wikipedia.org/wiki/Digital_image_processing

usually 3D: $\begin{bmatrix} \lambda_x & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and $\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ *(3rd dim is a constant, usually 1)*

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R \times \begin{bmatrix} x \\ y \end{bmatrix}$$

- The computation is done for all pixels.
  - Notice that computation at each pixel does not depend on any other data other than the pixel value

Simple Image Processing

- Shifting an object

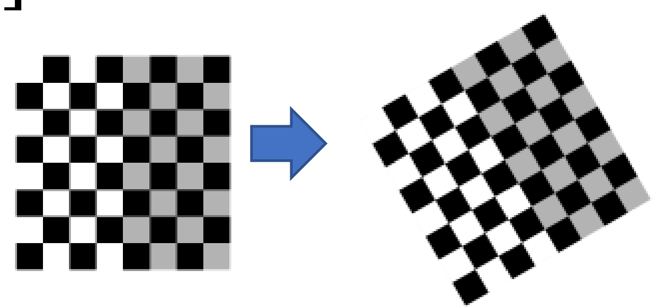
  $x' = x + \Delta x \qquad y' = y + \Delta y$

- Rotation of an object

  $x' = x \cos\theta - y \sin\theta \qquad y' = x \sin\theta + y \cos\theta$

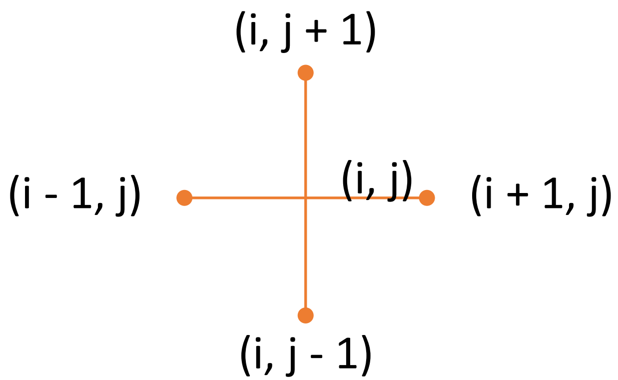
Rotation Matrix R: $\begin{bmatrix} cos\theta & -sin\theta & 0 \\ sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = R \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



https://en.wikipedia.org/wiki/Digital_image_processing

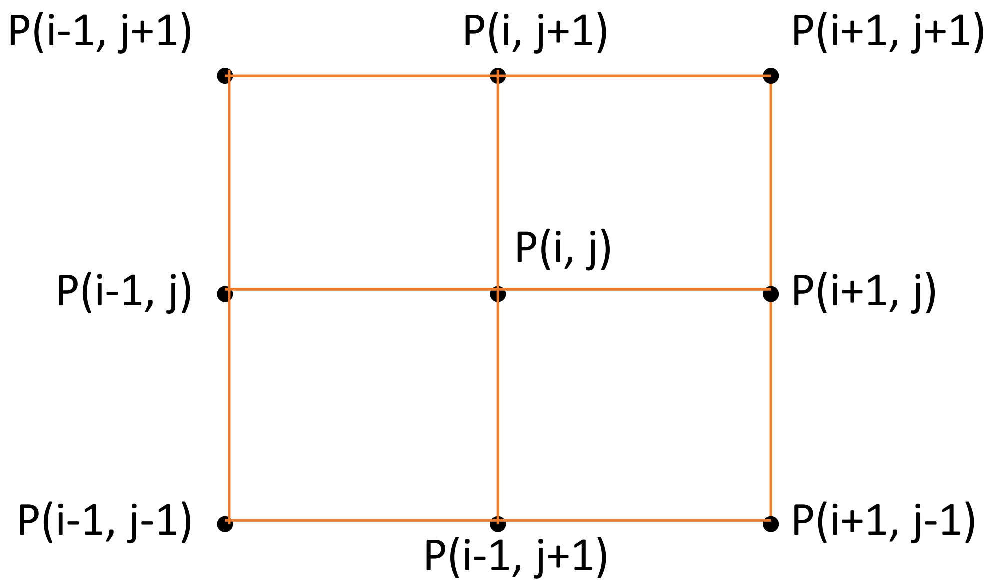
Simple Image Processing



$$x(i, j) = f\{\, x(i, j), x(i, j + 1), x(i, j - 1), x(i + 1, j), x(i - 1, j) \,\}$$

Example: Average values of the neighbouring pixels

Simple Image Processing



P'(i,j) = f { P(i,j), P(i+1, j+1), P(i+1, j), P(i+1, j-1), p(i-1, j), P(i-1, j-1),
P(i-1, j), P(i-1, j+1), P(i, j+1) }

P'(i,j) – new value of the pixel (i,j)

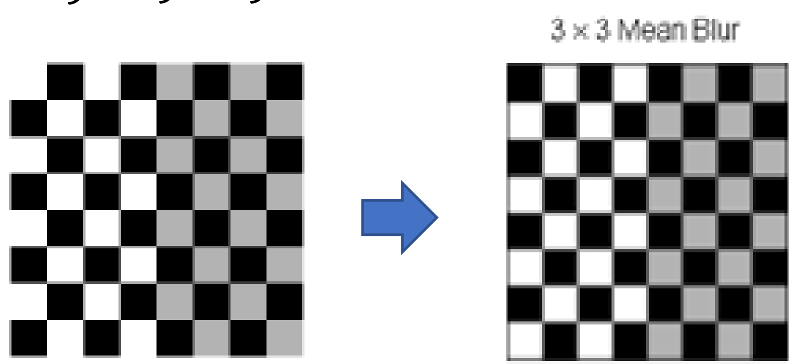
Simple Image Processing

- Lowpass (e.g. usage: blurring)

Mask / Kernel R: $\frac{1}{9}\begin{bmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{bmatrix}$

$P'_{ij} = 1/9 \{P_{ij} + P_{i+1j} + P_{i-1j} + P_{ij-1} + P_{i-1j-1} + P_{i+1j-1} + P_{ij+1} + P_{i-1j+1} + P_{i+1j+1}\}$

P'_{ij} – new value of the pixel (i,j)



https://en.wikipedia.org/wiki/Digital_image_processing

Highpass Kernel: Mask / Kernel R: $\frac{1}{9}\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

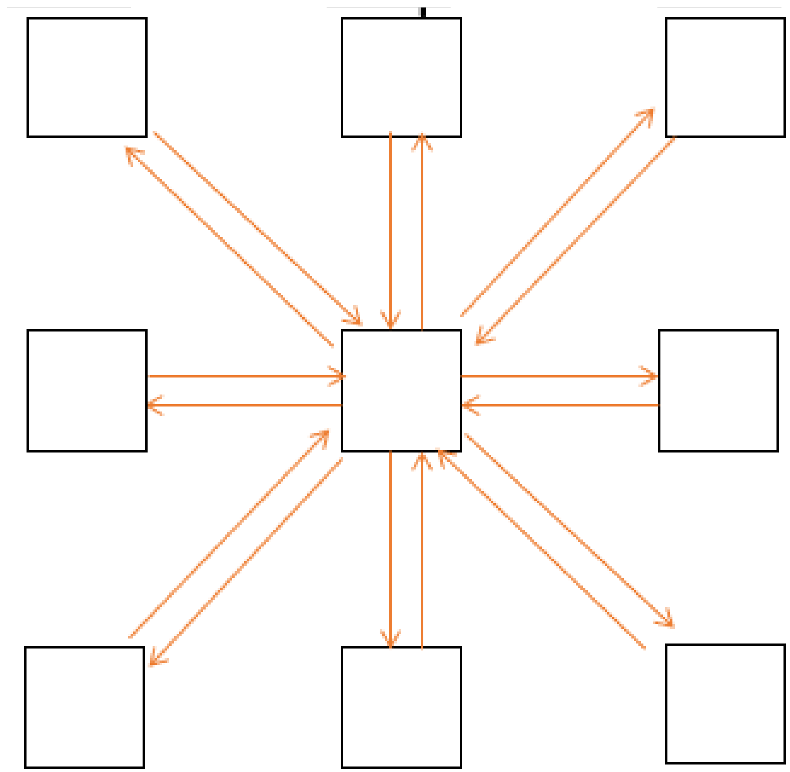
Simple Image Processing: Parallelizing

- Static Task Assignment
  - Divide the image region into fixed number of partitions
  - Assign each region to a distinct process / thread / processor
  - Different pixels may require different amount computation / iterations (depending on the application)
  - Number of iterations required – not known *a priori*
  - Unbalanced load for different processes
  - Performance – not very good

Dynamic Task Assignment

- Dynamic (i.e. runtime) allocation of tasks work pool – computations of different regions

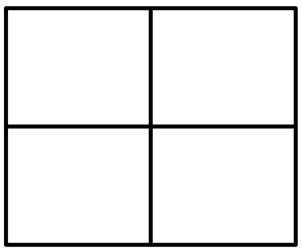
Simple Image Processing: Parallelizing



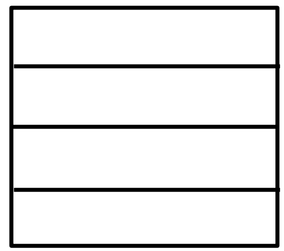
Assume: one Pixel Processing per Processor and neighbour pixel data is required in an application considered

Inter-processor Communication Requirements ?

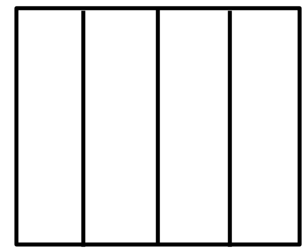
Data Decomposition



Square regions



Horizontal strips



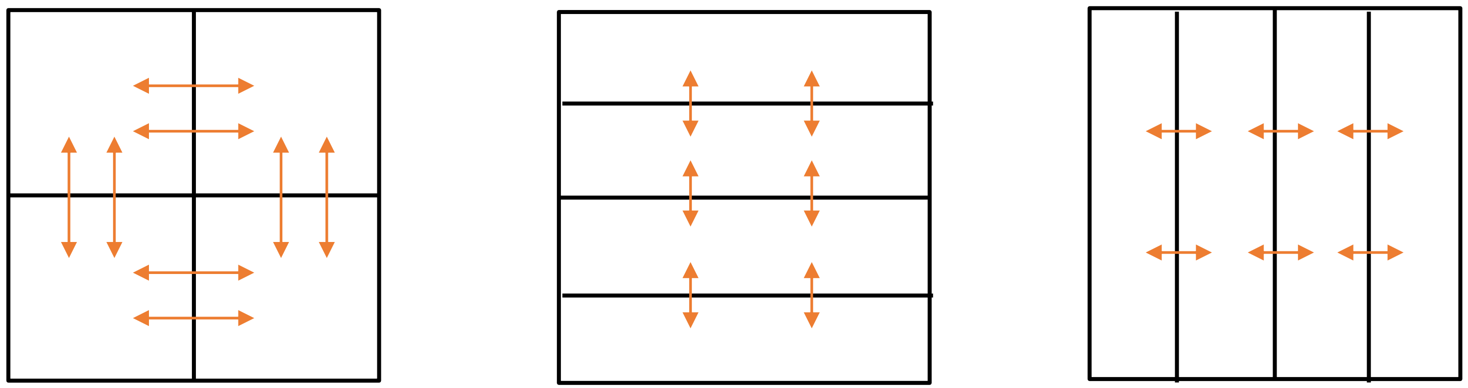
Vertical strips

Assume: region-wise partitioning of the pixel grid and neighbour pixel data is required in the application considered

Inter-processor Communication Requirements ?

Data Decomposition & Inter-processor Communication

- 4 Processors



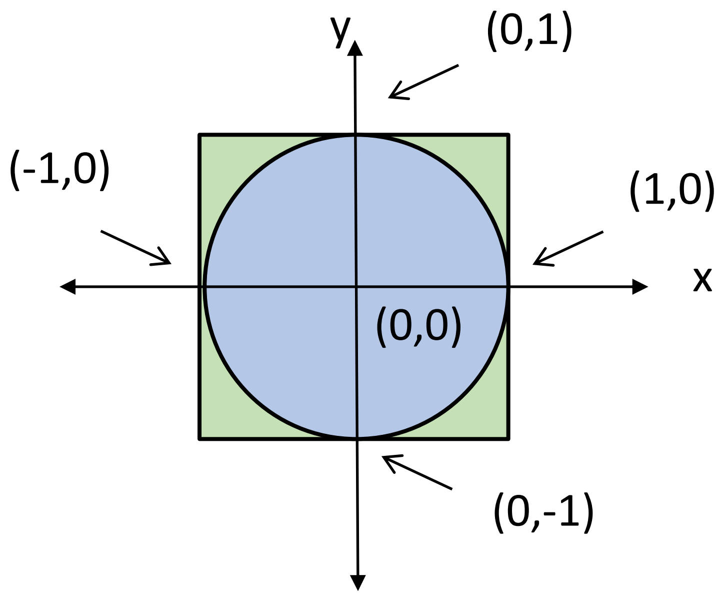
Computations/processor $\propto$(Area of the partition)
Communications/processor $\propto(\sum \text{length of the partitions adjacent to other partitions})$

Monte Carlo Methods

- Computations based on pseudo-random or quasi-random numbers

  Example: For C programs, use rand() for generating integer
  pseudo-random numbers; first use srand() to seed the
  rand() function.
- Applications: radiation transport, Monte Carlo simulation in communications, solution of partial differential equations (PDEs)
- Each process (processor) requires a pseudo-random (or quasi-random) number generator
- Obtain an estimate of the solution
- Estimate converges to the solution as the number of trials are increased

Monte Carlo Method



<u>Circle</u>

radius r = 1

Area of circle = $\pi r^2 = \pi$

Area of square = 2 x 2 = 4

$$\frac{Area\ of\ circle}{Area\ of\ square} = \frac{\pi}{4}$$

π = 4 . (area of circle) / (area of square)

Monte Carlo Method: Estimate of π

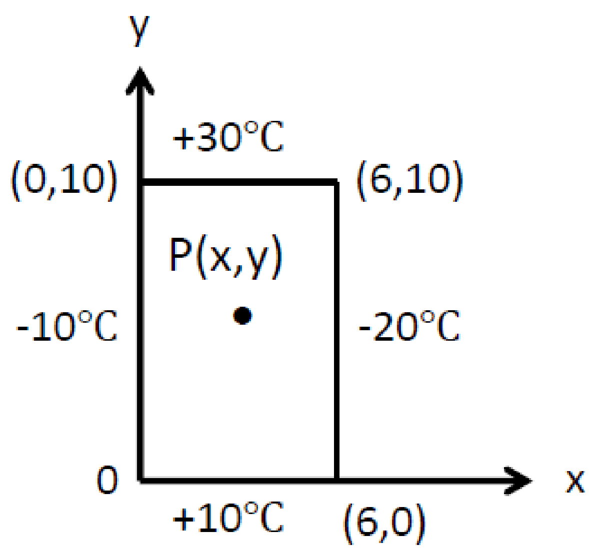
- Choose points randomly within the square
  (x,y): choose x and y coordinates randomly for a point
- Estimate of $\pi = \frac{4(\text{Number of points in circle})}{(\text{Number of points in square})}$

- Estimate of π → true value of π as
  number points → ∞
- Computations for all points are independent → can be done in parallel
- How to generate pseudo-random numbers in parallel?

- OpenMP Program (Demo)

Application: Laplace Equation

- Second order PDE : $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$



- Example
  - Metal plate of size 6 cm x 10 cm
  - Each edge (boundary) is held at a constant temperature
  - Find temperatures of points within the plate
  - Steady-state solution

Application: Laplace Equation

- We begin by writing difference equation for approximating the PDE
- Discretize the region (create a mesh of grid points)
- Compute the temperature at each grid point

Laplace Equation – Numerical Solution

1. Approximate the derivatives of $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$ = 0 using central differences

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{\big(u(x+\delta x, y) - 2u(x,y) + u(x-\delta x, y)\big)}{(\delta x)^2}$$

$$\frac{\partial^2 u}{\partial y^2} \approx \frac{\big(u(x, y+\delta y) - 2u(x,y) + u(x, y-\delta y)\big)}{(\delta y)^2}$$

Where, δx and δy are step sizes along x and y direction resp.

Laplace Equation – Numerical Solution

- Substituting in $\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$:

$$\frac{\big(u(x+\delta x, y) - 2u(x,y) + u(x-\delta x, y)\big)}{(\delta x)^2}$$

+

$$\frac{\big(u(x, y+\delta y) - 2u(x,y) + u(x, y-\delta y)\big)}{(\delta y)^2}$$

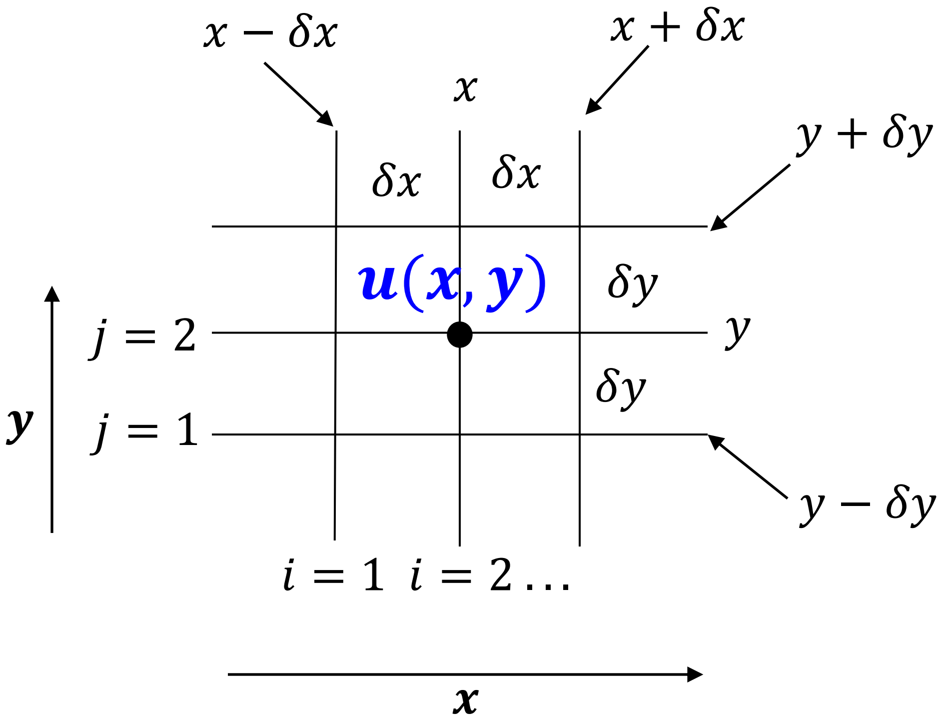
=

$$\frac{\big(u(x+\delta x, y) + \ u(x, y+\delta y) - 4u(x,y) + u(x-\delta x, y) + \ u(x, y-\delta y)\big)}{(h)^2}$$

$= 0$

Laplace Equation – Numerical Solution

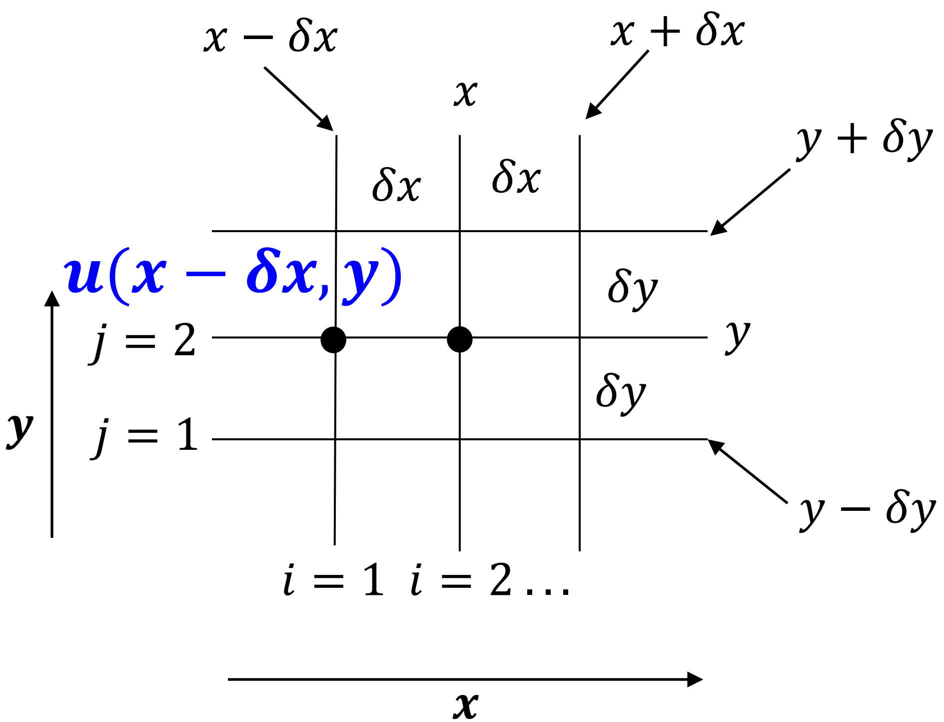
- Representing $u(x, y)$



Notation: $\mathbf{u_{i,j}}$

Laplace Equation – Numerical Solution

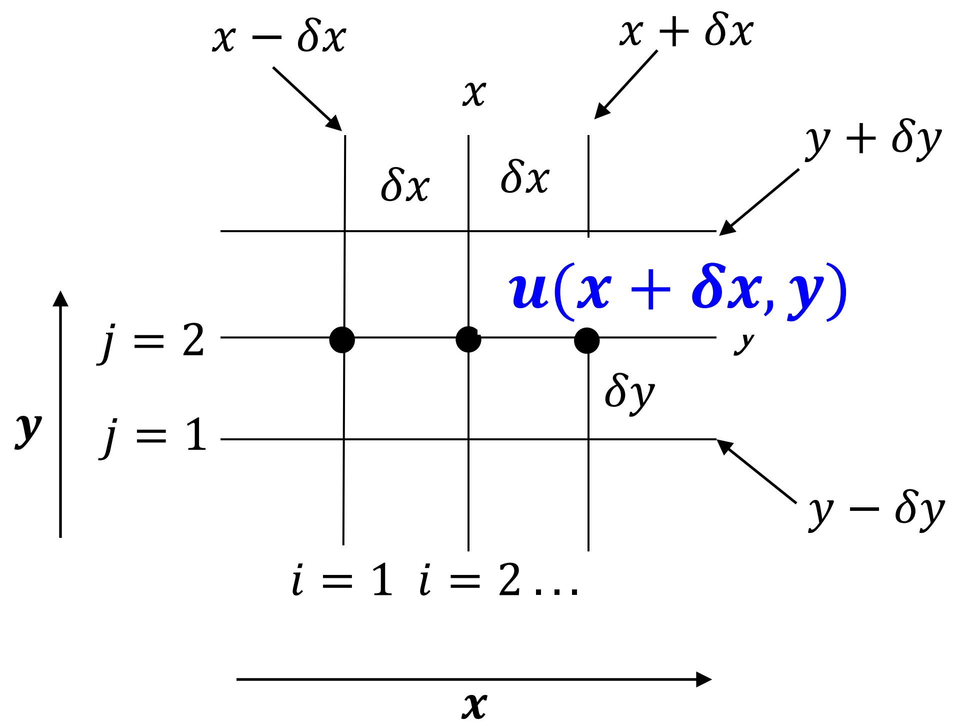
- Representing $u(x - \delta x, y)$



Notation: $\mathbf{u_{i-1,j}}$

Laplace Equation – Numerical Solution

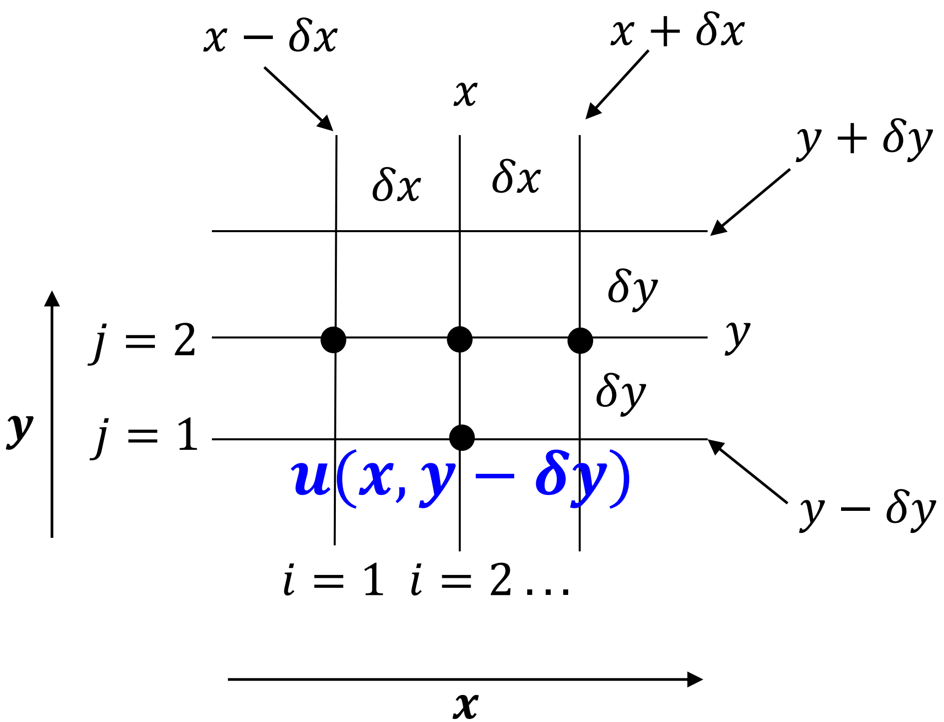
- Representing $u(x + \delta x, y)$



Notation: $\mathbf{u_{i+1,j}}$

Laplace Equation – Numerical Solution

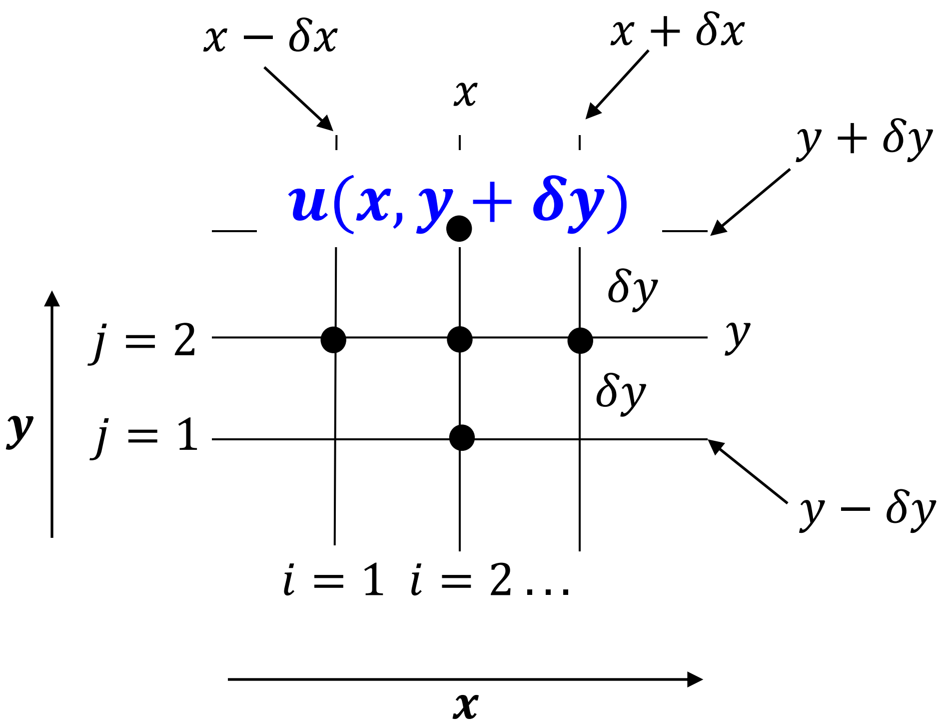
- Representing $u(x, y - \delta y)$



Notation: $\mathbf{u_{i,j-1}}$

Laplace Equation – Numerical Solution

- Representing $u(x, y + \delta y)$



Notation: $\mathbf{u_{i,j+1}}$

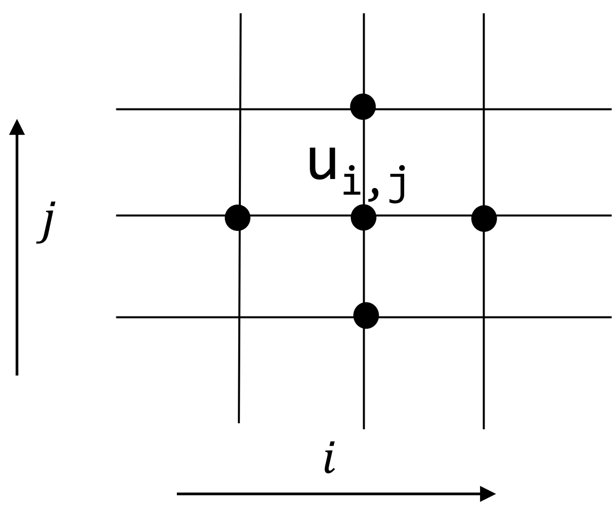
Laplace Equation – Numerical Solution

- Rewriting using notation:

$$\frac{\big(u(x+\delta x, y) + u(x, y+\delta y) - 4u(x,y) + u(x-\delta x, y) + u(x, y-\delta y)\big)}{(h)^2}$$

$$= 0$$

$$\frac{\mathbf{u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}}}{\mathbf{h^2}} = 0$$



called 5-point stencil

Laplace Equation – Numerical Solution

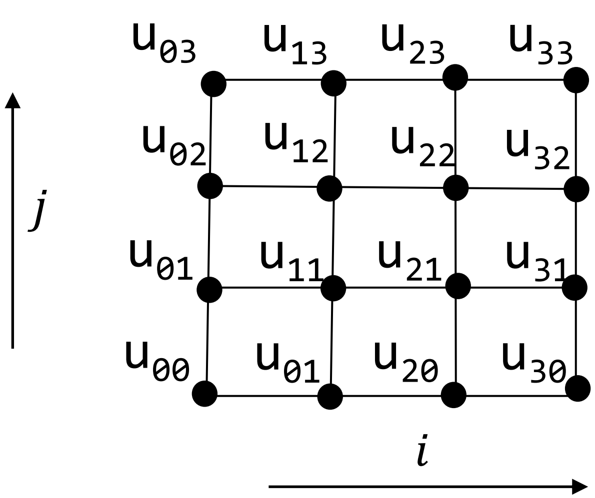
- Consider the *boundary-value* problem:

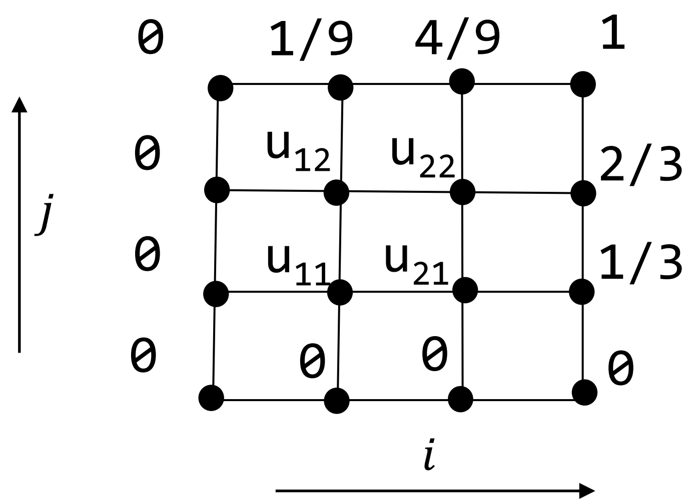
$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \text{ in the square } 0 < x < 1, 0 < y < 1$$

$u = x^2 y$ on the boundary, $h = 1/3$

$$\frac{u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1}}{h^2} = 0$$

Equation 1





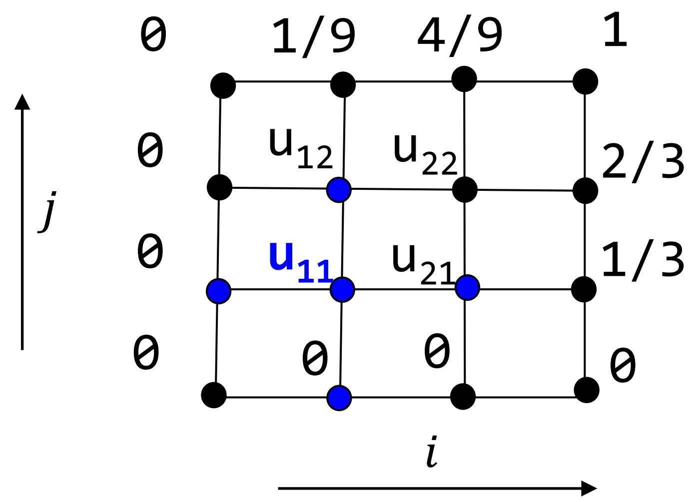
Laplace Equation – Numerical Solution

- Substituting for i,j in equation 1 and computing u_{11}

$$(u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1} = 0)$$

$$u_{21} + u_{12} - 4u_{11} + u_{01} + u_{10} = 0$$

$$u_{21} + u_{12} - 4u_{11} + 0 + 0 = 0$$



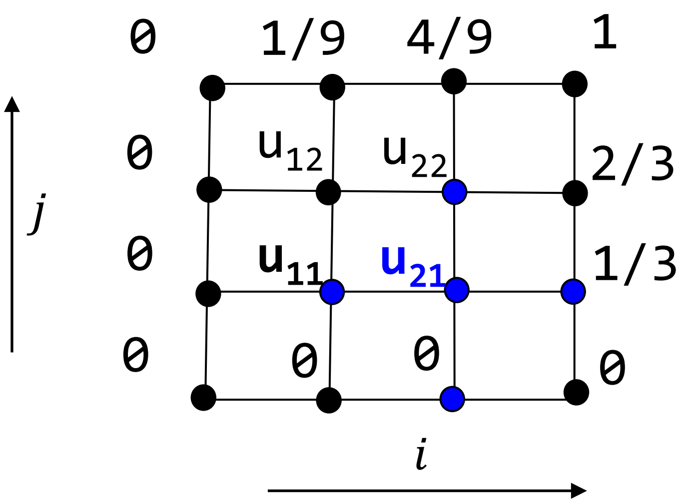
Laplace Equation – Numerical Solution

- Computing u_{21}

$(u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1} = 0)$

$$u_{31} + u_{22} - 4u_{21} + u_{11} + u_{20} = 0$$

$$1/3 + u_{22} - 4u_{21} + U_{11} + 0 = 0$$



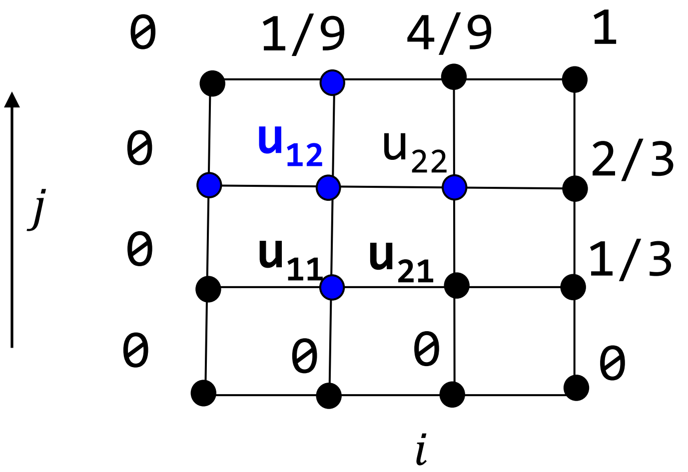
Laplace Equation – Numerical Solution

- Computing u_{12}

$(u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1} = 0)$

$u_{22} + u_{13} - 4u_{12} + u_{02} + u_{11} = 0$

$u_{22} + 1/9 - 4u_{12} + 0 + u_{11} = 0$



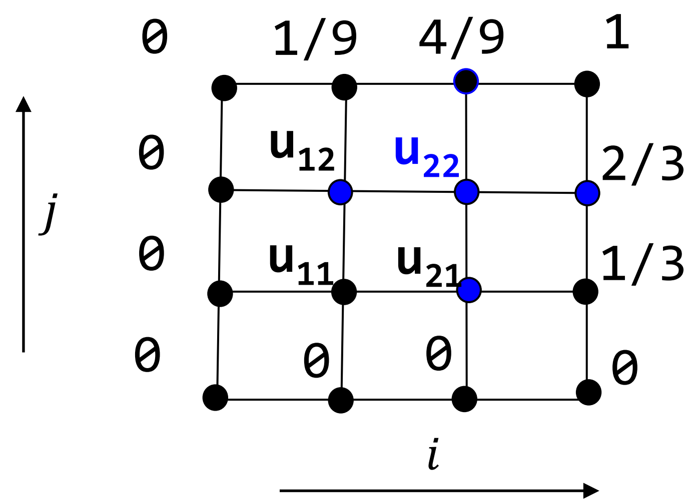
Laplace Equation – Numerical Solution

- Computing u_{22}

$(u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1} = 0)$

$$u_{32} + u_{23} - 4u_{22} + u_{12} + u_{21} = 0$$

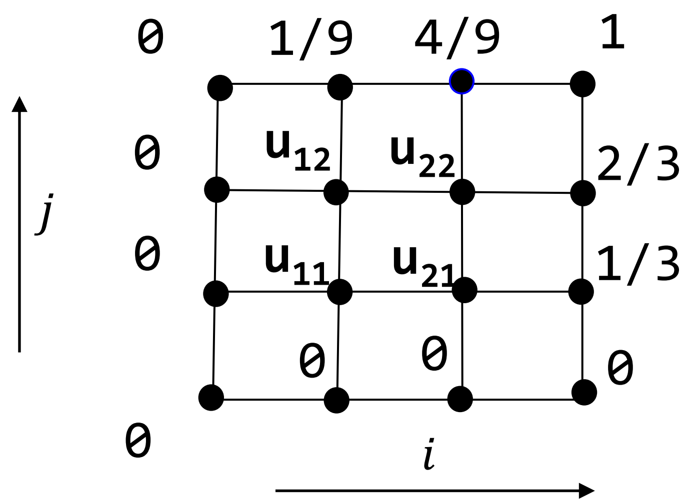
$$2/3 + 4/9 - 4u_{22} + u_{12} + u_{21} = 0$$



Laplace Equation – Numerical Solution

- System of Equations

$$(u_{i+1,j} + u_{i,j+1} - 4u_{i,j} + u_{i-1,j} + u_{i,j-1} = 0)$$



Right Top Center Left Bottom

$$u_{21} + u_{12} - 4u_{11} + 0 + 0 = 0$$

$$1/3 + u_{22} - 4u_{21} + u_{11} + 0 = 0$$

$$u_{22} + 1/9 - 4u_{12} + 0 + u_{11} = 0$$

$$2/3 + 4/9 - 4u_{22} + u_{12} + u_{21} = 0$$

Laplace Equation – Numerical Solution

- Computing System of Equations:

$$u_{21} + u_{12} - 4u_{11} + 0 + 0 = 0$$

$$1/3 + u_{22} - 4u_{21} + u_{11} + 0 = 0$$

$$u_{22} + 1/9 - 4u_{12} + 0 + u_{11} = 0$$

$$2/3 + 4/9 - 4u_{22} + u_{12} + u_{21} = 0$$

$$\begin{pmatrix} -4 & 1 & 1 & 0 \\ 1 & -4 & 0 & 1 \\ 1 & 0 & -4 & 1 \\ 0 & 1 & 1 & -4 \end{pmatrix} \begin{pmatrix} u_{11} \\ u_{21} \\ u_{12} \\ u_{22} \end{pmatrix} = \begin{pmatrix} 0 \\ -1/3 \\ -1/9 \\ -10/9 \end{pmatrix}$$

Ax=B

A x = B

Matrix A has only coefficients

	1	
1	**-4**	**1**
	1	

Computing Stencil – Iterative Methods

- Jacobi and Gauss-Seidel
  - Start with an initial guess for the unknowns u^0_{ij}
  - Improve the guess u^1_{ij}
  - Iterate: derive the new guess, u^{n+1}_{ij} , from old guess u^n_{ij}

Background – Jacobi Iteration

- **Goal:** find solution to system of equations represented by $AX=B$
- **Approach:** find sequence of approximations X^0 X^1 X^2 . . . X^n, which gradually approach X.
  - X^0 is called initial guess, X^i's called *iterates*
- **Method:**
  - Split A into $A=L+D+U$ e.g.

$$\begin{pmatrix} -4 & 1 & 1 & 0 \\ 1 & -4 & 0 & 1 \\ 1 & 0 & -4 & 1 \\ 0 & 1 & 1 & -4 \end{pmatrix} = \underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}}_{L} + \underbrace{\begin{pmatrix} -4 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & -4 \end{pmatrix}}_{D} + \underbrace{\begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}}_{U}$$

Background – Jacobi Iteration

- **Compute:** $AX=B$ is $(L+D+U)X=B$

  $\Rightarrow DX = -(L+U)X+B$

  $\Rightarrow DX^{(k+1)} = -(L+U)X^{k}+B$ **(iterate step)**

  $\Rightarrow X^{(k+1)} = D^{-1}(-(L+U)X^{k}) + D^{-1}B$

  (As long as D has no zeros in the diagonal $X^{(k+1)}$ is obtained)

- E.g. $\begin{pmatrix} -4 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & -4 \end{pmatrix} \begin{pmatrix} u_{11} \\ u_{21} \\ u_{12} \\ u_{22} \end{pmatrix}^{\mathbf{1}} = -\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} u_{11} \\ u_{21} \\ u_{12} \\ u_{22} \end{pmatrix}^{\mathbf{0}} + \begin{pmatrix} 0 \\ -1/3 \\ -1/9 \\ -10/9 \end{pmatrix},$

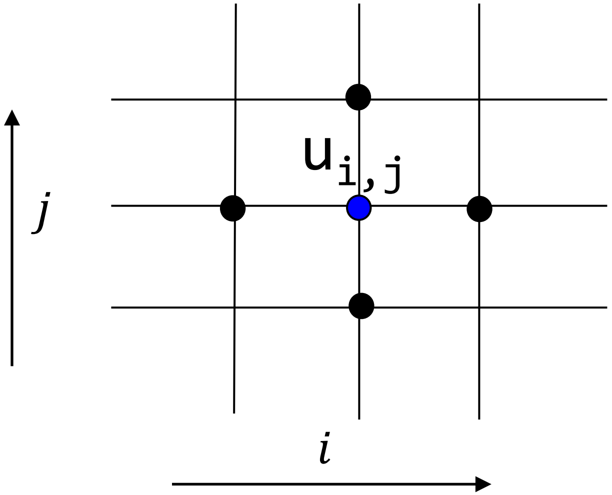
u_{ij} 's value in $(\mathbf{1})^{st}$ iteration is computed based on u_{ij} values computed in $(\mathbf{0})^{th}$ iteration

Background – Jacobi Iteration

- E.g. $\begin{pmatrix} -4 & 0 & 0 & 0 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & -4 & 0 \\ 0 & 0 & 0 & -4 \end{pmatrix} \begin{pmatrix} u_{11} \\ u_{21} \\ u_{12} \\ u_{22} \end{pmatrix}^{k+1} = -\begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} u_{11} \\ u_{21} \\ u_{12} \\ u_{22} \end{pmatrix}^{k} + \begin{pmatrix} 0 \\ -1/3 \\ -1/9 \\ -10/9 \end{pmatrix}$,

u_{ij} 's value in $(\mathbf{k+1})^{st}$ iteration is computed based on u_{ij} values computed in $(\mathbf{k})^{th}$ iteration

- Center's value is updated. Why?



5-point stencil

Computing Stencil

- Jacobi Solution Approach:
  - Approximate the *value of the center* with *old values* of (left, right, top, bottom)

Computing Stencil

- $\boldsymbol{u_{right} + u_{top} - 4u_{center} + u_{left} + u_{bottom} = 0}$

  $\Rightarrow \boldsymbol{u_{center} = 1/4(u_{right} + u_{top} + u_{left} + u_{bottom})}$

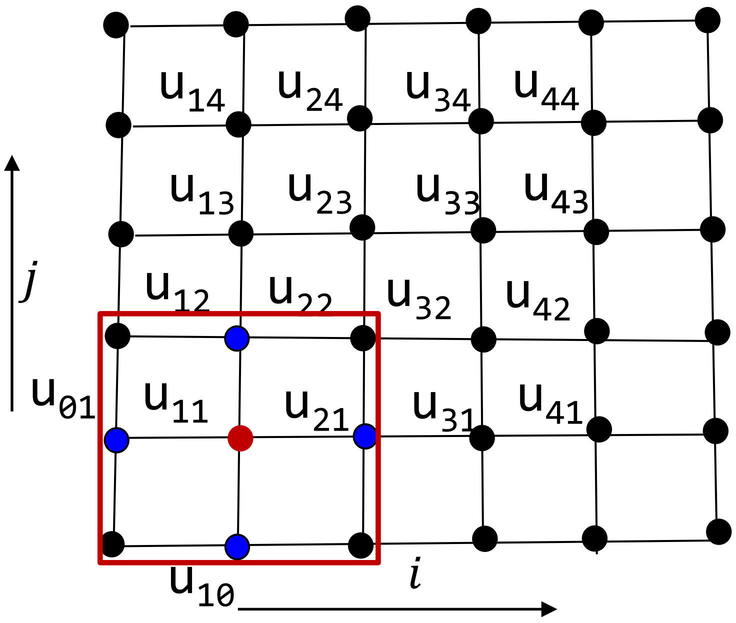
- Applying Jacobi Iteration:

$$\boldsymbol{u_{center}^{(k+1)} = 1/4(u_{right}^{(k)} + u_{top}^{(k)} + u_{left}^{(k)} + u_{bottom}^{(k)})}$$

Computing Stencil

- Example: applying Jacobi Iteration for 6x6 grid:

$$u_{center}^{(k+1)} = 1/4(u_{right}^{(k)} + u_{top}^{(k)} + u_{left}^{(k)} + u_{bottom}^{(k)})$$



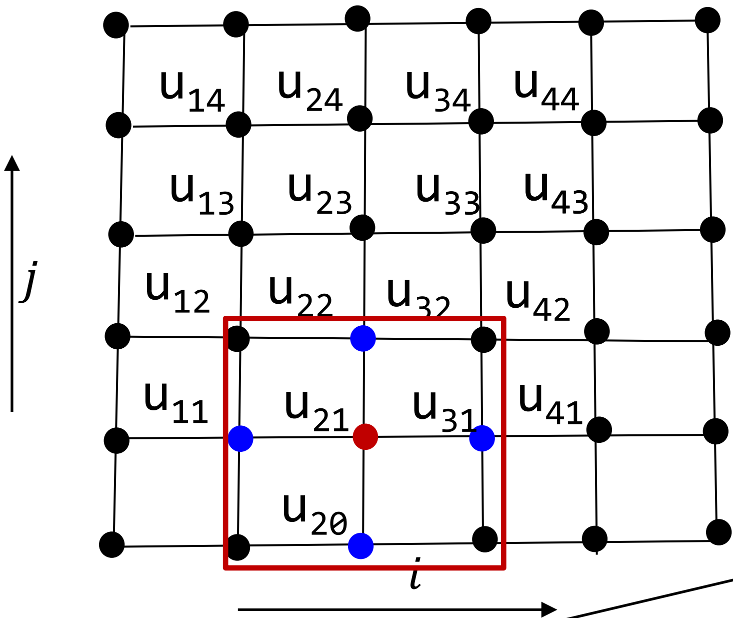
<u>Iteration 1</u>

1) Compute u_{11} using initial guess for u_{12} and u_{21}. u_{01} and u_{10} are known from boundary conditions

Computing Stencil

- Example: applying Jacobi Iteration:

$$\boldsymbol{u_{center}^{(k+1)} = 1/4(u_{right}^{(k)} + u_{top}^{(k)} + u_{left}^{(k)} + u_{bottom}^{(k)})}$$



<u>Iteration 1</u>

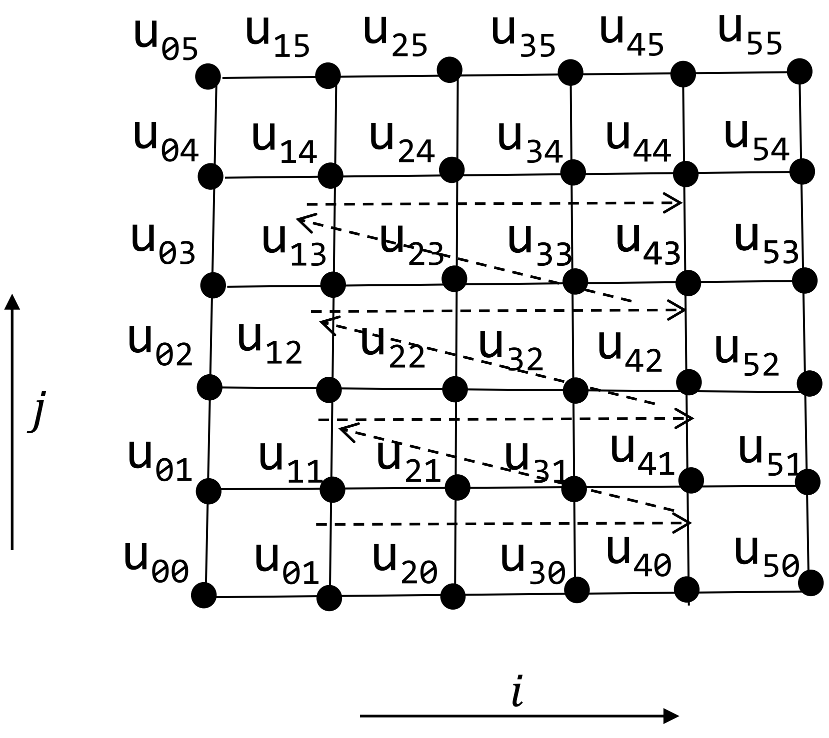
1) Compute u_{11} using initial guess for u_{12} and u_{21}. u_{01} and u_{10} are known from boundary conditions

2) Compute u_{21} using initial guess for u_{11}, u_{31}, and u_{22}. u_{20} are known from boundary conditions

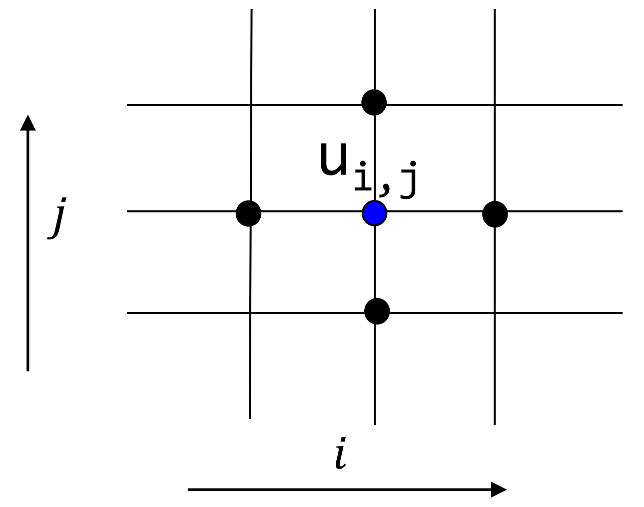
In 2), note that the initial guess for u_{11} is used even though u_{11} was updated just before in 1)

Computing Stencil

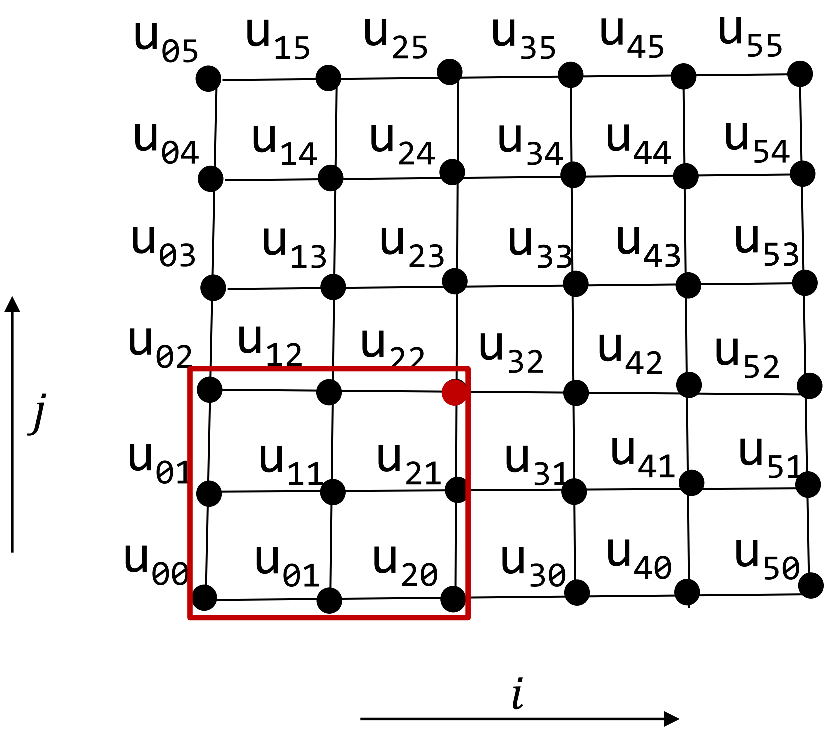
- In every iteration, suppose we follow the computing order as shown (dashed):



In any iteration, what are all the points of a 5-point stencil already updated while computing u_{ij} ?

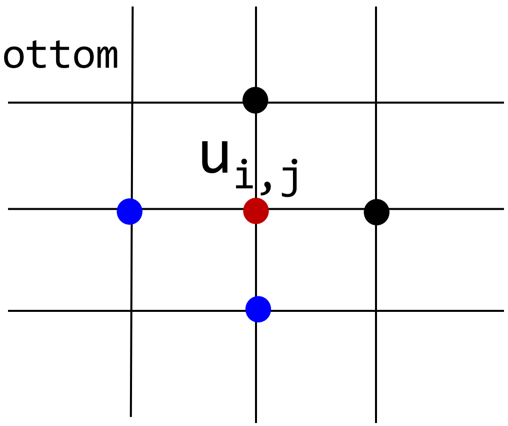


Computing Stencil



What are the points that are already computed at $\mathtt{u_{i,j}}$*?*

$\mathtt{u_{left}}$, $\mathtt{u_{bottom}}$



Parallel Implementation

- Using new values as soon as they are computed

- Cannot be parallelized
- Solution
  - Use all old values to find new values for all internal mesh points in parallel
  - Red-black ordering
  - Wavefront ordering

Parallel Implementation

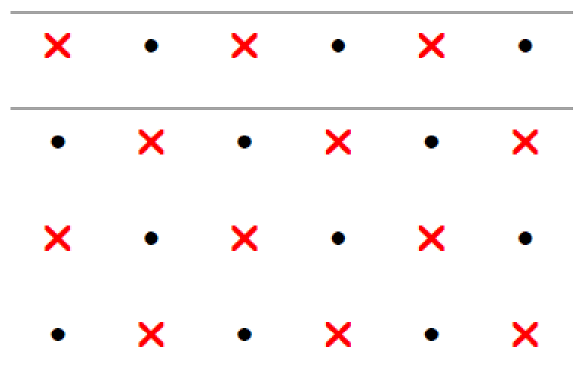
- Assuming 1 grid point per process and Jacobi (solution)

- Each process sends and receives 4 messages (except the processes adjacent to a boundary)

Total number of iterations	= m
Total number of messages per process	= m (4 + 4)
	= 8 m send/receive messages
Total number of messages for a corner process	= 4 m
Total number of messages for a process adjacent to only one boundary point	= 6 m

Laplace's Equation: Finite Difference Method – Red-Black Ordering

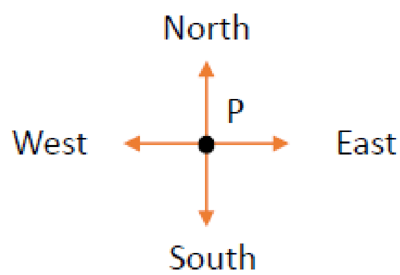
× - Red ● - Black

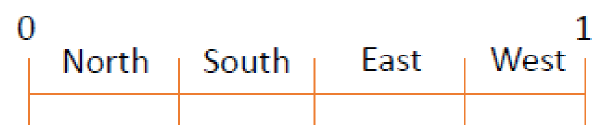


Step1: Compute Red points
Step2: Compute Black points
Step3: If convergence not reached, go back to step1

Laplace's Equation: Monte Carlo Method

- Discretize the region, mesh size h = ¼
- Random walk
  - Start at the point where solution is desired
  - Generate a random number between (0,1) with uniform distribution





  - Decide the direction of the next step based on the value of random number
  - Continue the random walk until arrive at a boundary point. Note that boundary value b_i for i-th random walk

Laplace's Equation: Monte Carlo Method

- Carry out N random walks, until the estimate of solution sufficiently converges

$$\text{Estimate of the solution} = \frac{1}{N}\sum_{i=1}^{N} b_i$$

- Estimate $\rightarrow$ solution, as $N \rightarrow \infty$

Monte Carlo Method: Laplace’s Equation

- Each random walk is independent of the other random walks
  - All random walks can be carried out in parallel
  - Each process (processor) would require a pseudo-random number generator
- The estimate of the solution converges to the approximate solution based on the grid size (h)
- The solution can be obtained only at a particular point inside the region
- Number of steps in a random walk- random variable
- Dynamic task allocation preferable when number of processors < N

Laplace's Equation: Monte Carlo Solution

Reference:

Bhavsar, V.C. and J.R. Isaac, "Design and Analysis of Parallel Monte Carlo Algorithms"
SIAM J. Statistical and Scientific Computing,
Vol. 8, No. 1, s73-s95, Jan. 1987.