COMP5426
Parallel and Distributed Computing

Performance Analysis of Parallel Systems (Examples)

Jacobi Iteration

- Jacobi iteration is a numerical method used to solve Laplace partial differential equations, e.g., to determine the steady-state temperature on some domain when the temperature of its boundaries is held fixed.
- The method approaches a solution iteratively; in each iteration, the temperature of a point is computed to be the average of temperatures of its (four) neighbors, except that temperatures at the boundary are not changed.
- This iterative method is often referred to as relaxation method as an initial *guess* at the solution is allowed to slowly relax towards the true solution, reducing the errors as it does so.

Sequential Algorithm

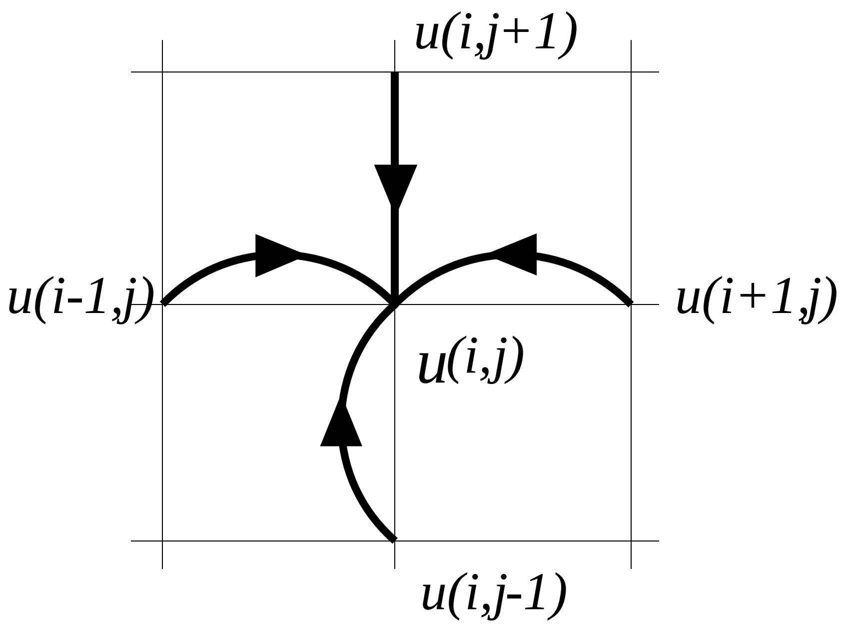
- To find the solution for a two-dimensional Laplace equation:

1. Initialise u_j to some initial *guess*.
2. Apply the boundary conditions.
3. For each internal mesh point set
   - $u_{ij}* = (u_{(i+1)j} + u_{(i-1)j} + u_{i(j+1)} + u_{i(j-1)})/4.$
4. Replace old solution U with new estimate U*.
5. If solution does not satisfy tolerance, repeat from step 2.

Heart of Sequential Algorithm

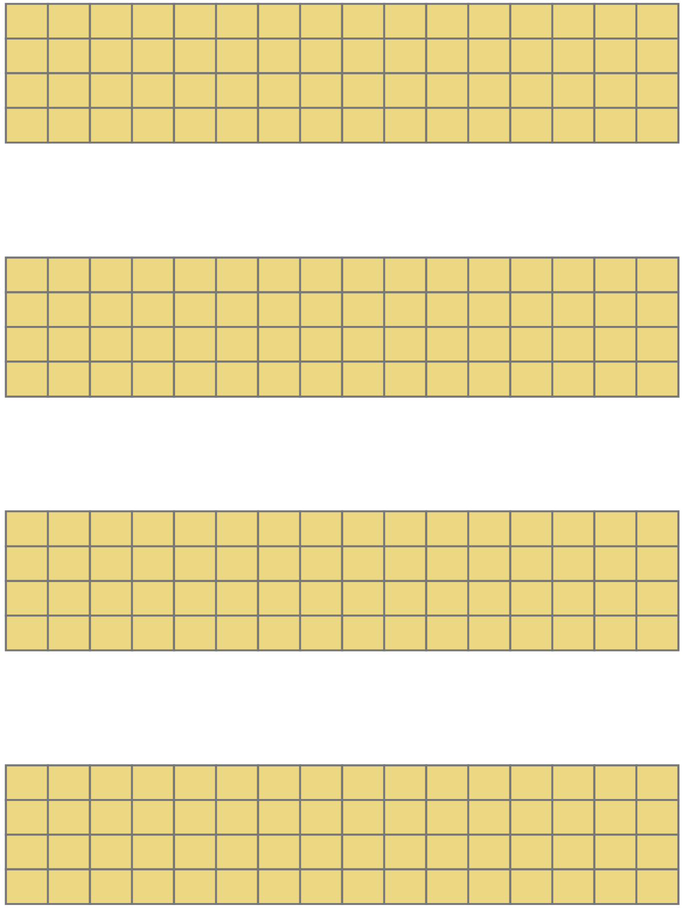
- Create a 2D grid and
- Each grid point represents value of state solution at particular (*i, j*) location in plate

u*(i,j) = (u(i-1,j) + u(i+1,j) + u(i,j-1) + u(i,j+1)) / 4.0;



Parallel Algorithm 1

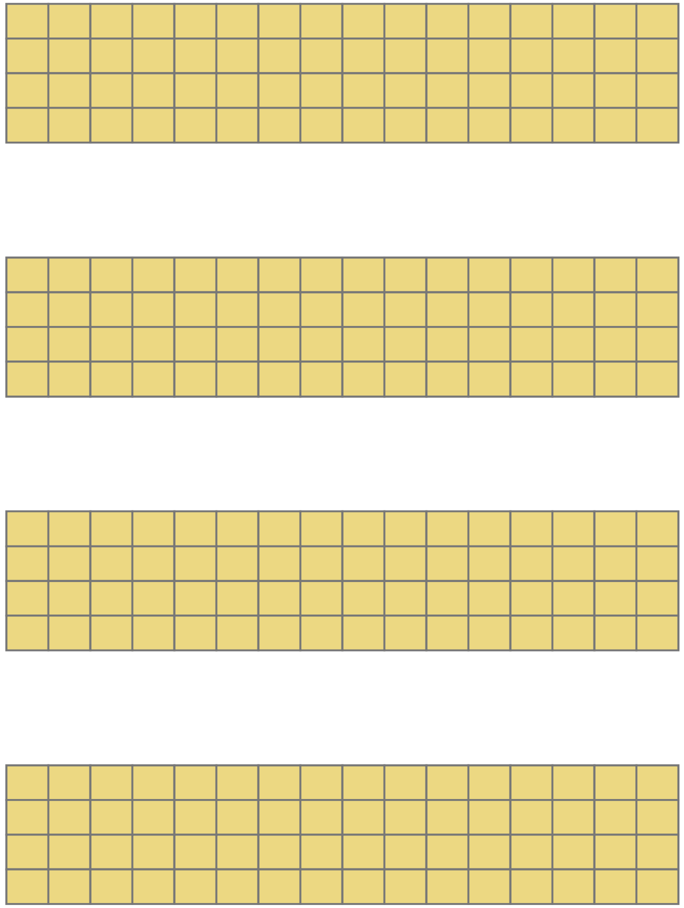
- Associate primitive task with each matrix element
- Agglomerate tasks in contiguous rows (rowwise block striped decomposition)



16×16 grid divided among 4 processors

Communication

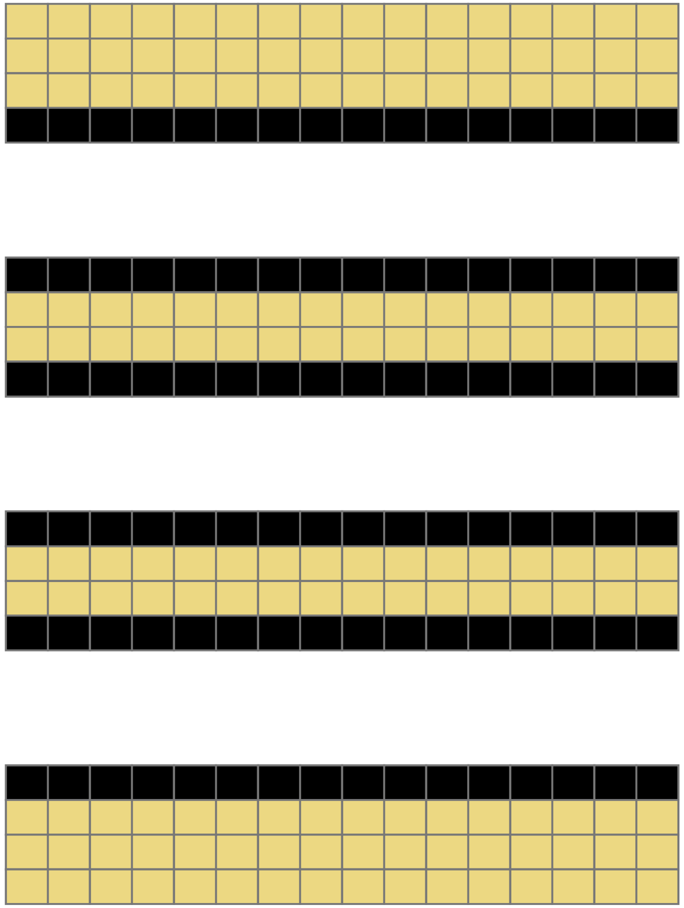
- Values in black cells cannot be computed without access to values held by other Associate primitive tasks.



16 × 16 grid divided among 4 processors

Communication

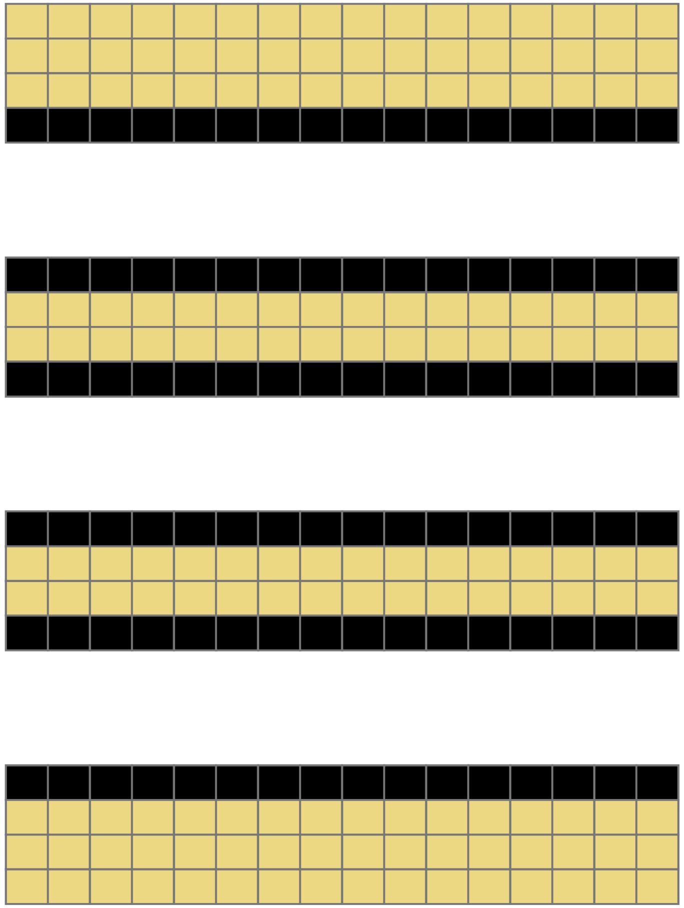
- Values in black cells cannot be computed without access to values held by other Associate primitive tasks.



16 × 16 grid divided among 4 processors

Ghost Points

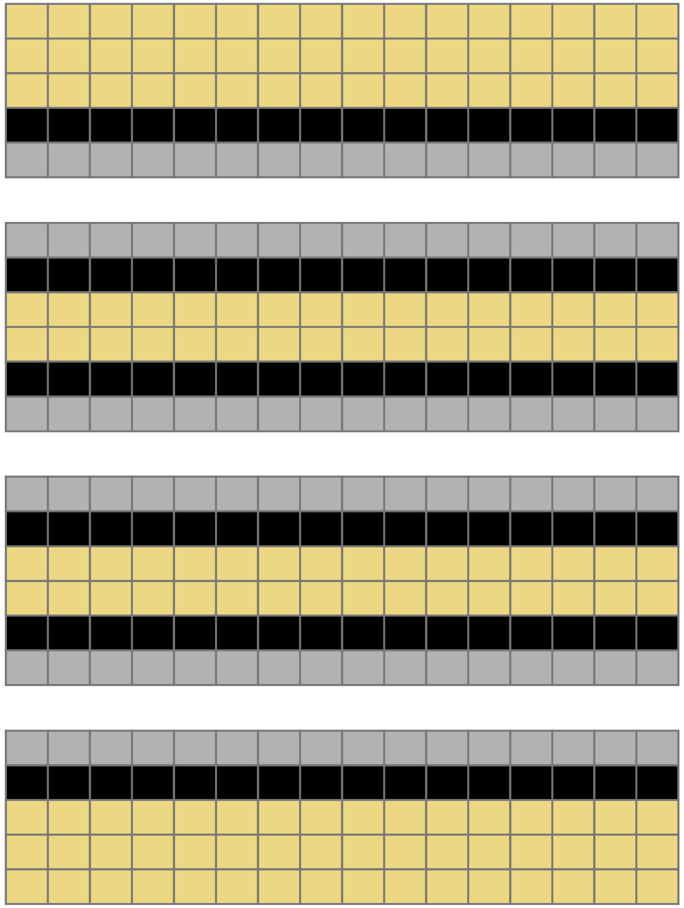
- Memory locations used to store redundant copies of data held by neighboring processes
- Allocating ghost points as extra columns simplifies parallel algorithm by allowing same loop to update all cells



16 × 16 grid divided among 4 processors

Ghost Points

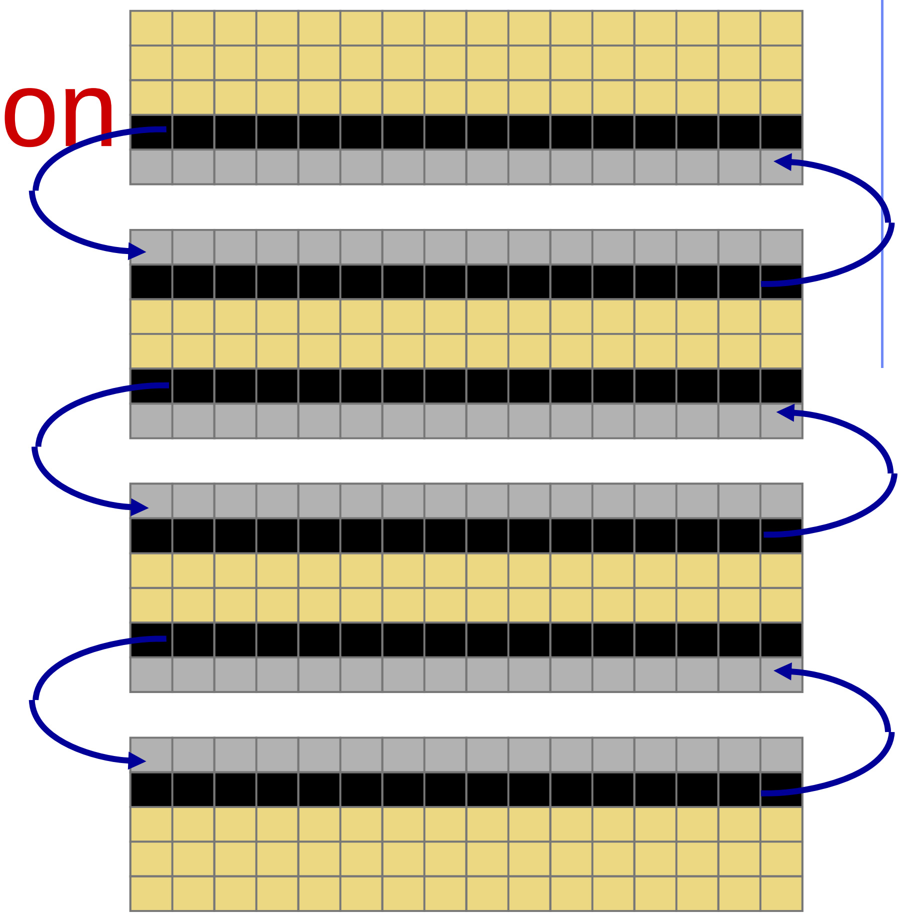
- Memory locations used to store redundant copies of data held by neighboring processes
- Allocating ghost points as extra columns simplifies parallel algorithm by allowing same loop to update all cells



16×16 grid divided among 4 processors

Comm in a Iteration

- At the beginning of each iteration, boundary rows are send/recv between left and right neighbors



16 × 16 grid divided among 4 processors

Complexity Analysis

- Sequential time complexity: $\Theta(n^2)$ each iteration
- Parallel computational complexity: $\Theta(n^2 / p)$ each iteration
- Parallel communication complexity: $\Theta(n)$ each iteration (two sends and two receives of n elements)

Complexity Analysis

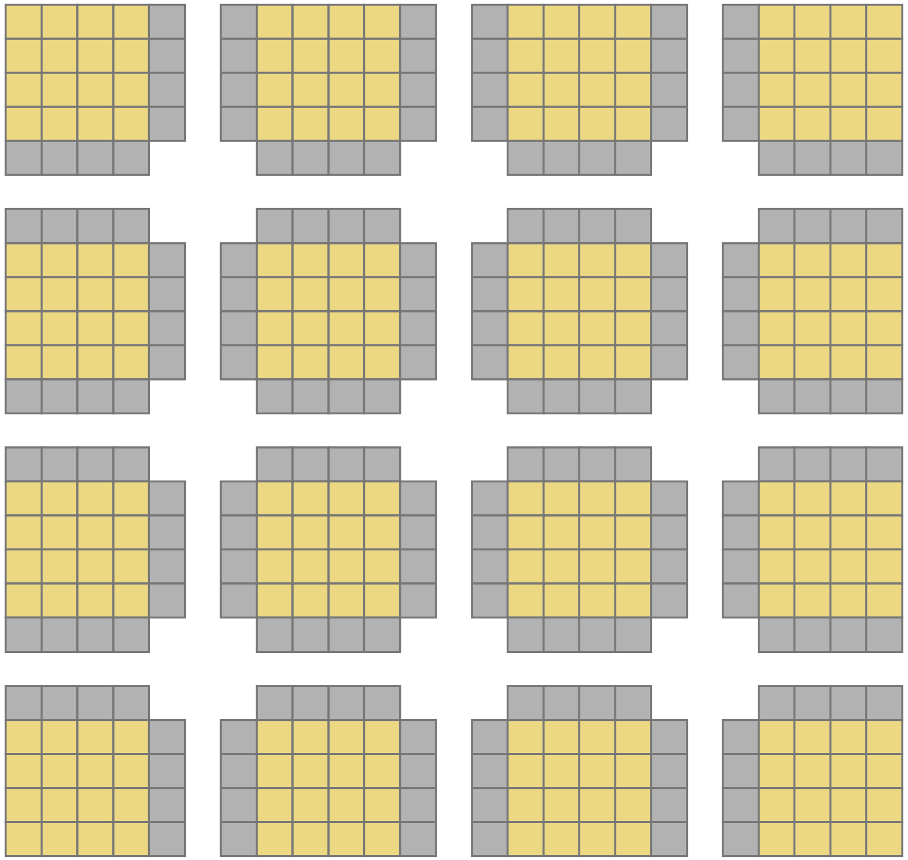
- Sequential time: $\Theta(n^2)$ (T_s)
- Parallel time: $\Theta(n^2/\text{p}+n)$ (T_p)
- Total parallel time: $\Theta(n^2+pn)$ ($p\,T_p$)
- Total overhead: $\Theta(pn)$ ($p\,T_p$ - T_s)
- Speedup: $\Theta(p\,/(1+p/\,n))$ (T_s / T_p)
- Efficiency: $\Theta(1/(1+\text{p}/\,n))$ (T_s / $p\,T_p$)
- Isoefficiency relation: ($T_s \geq C T_o$)

$$n^2 \geq Cnp \Rightarrow n \geq Cp$$

Parallel Algorithm 2

- Associate primitive task with each matrix element
- Agglomerate tasks into blocks that are as square as possible (checkerboard block decomposition)
- Add rows and columns of ghost points to all four sides of rectangular region controlled by each process

Example Decomposition



16 × 16 grid divided among 16 processors

Implementation Details

- Using ghost points around 2-D blocks requires extra copying steps
- Ghost points for left and right sides are not in contiguous memory locations
  - An auxiliary buffer may be used when receiving these ghost point values and similarly, buffer may be used when sending column of values to a neighboring process

Complexity Analysis

- Sequential time: $\Theta(n^2)$
- Parallel computational time: $\Theta(n^2 / p)$
- Parallel communication time: $\Theta(n / \sqrt{p})$ each iteration (four sends and four receives of $n / \sqrt{p}$ elements each)

Complexity Analysis

- Sequential time: $\Theta(n^2)$
- Parallel time: $\Theta(n^2/\text{p}+n/\sqrt{p}\ )$
- Total parallel time: $\Theta(n^2+\sqrt{p}n)$
- Total overhead: $\Theta(\sqrt{p}n)$
- Speedup: $\Theta(p\ /(1+\sqrt{p}\ /\ n))$
- Efficiency: $\Theta(1/(1+\sqrt{p}\ /\ n))$
- Isoefficiency relation:($T_S \geq CT_O$)
  $n^2 \geq Cn\sqrt{p} \Rightarrow n \geq C\sqrt{p}$

Questions

- Complexity analysis for
  - matrix multiplication
  - odd-even transposition sort