Finite Difference Methods: Outline

- Solving ordinary and partial differential equations
- Finite difference methods (FDM) vs Finite Element Methods (FEM)
- Vibrating string problem
- Steady state heat distribution problem

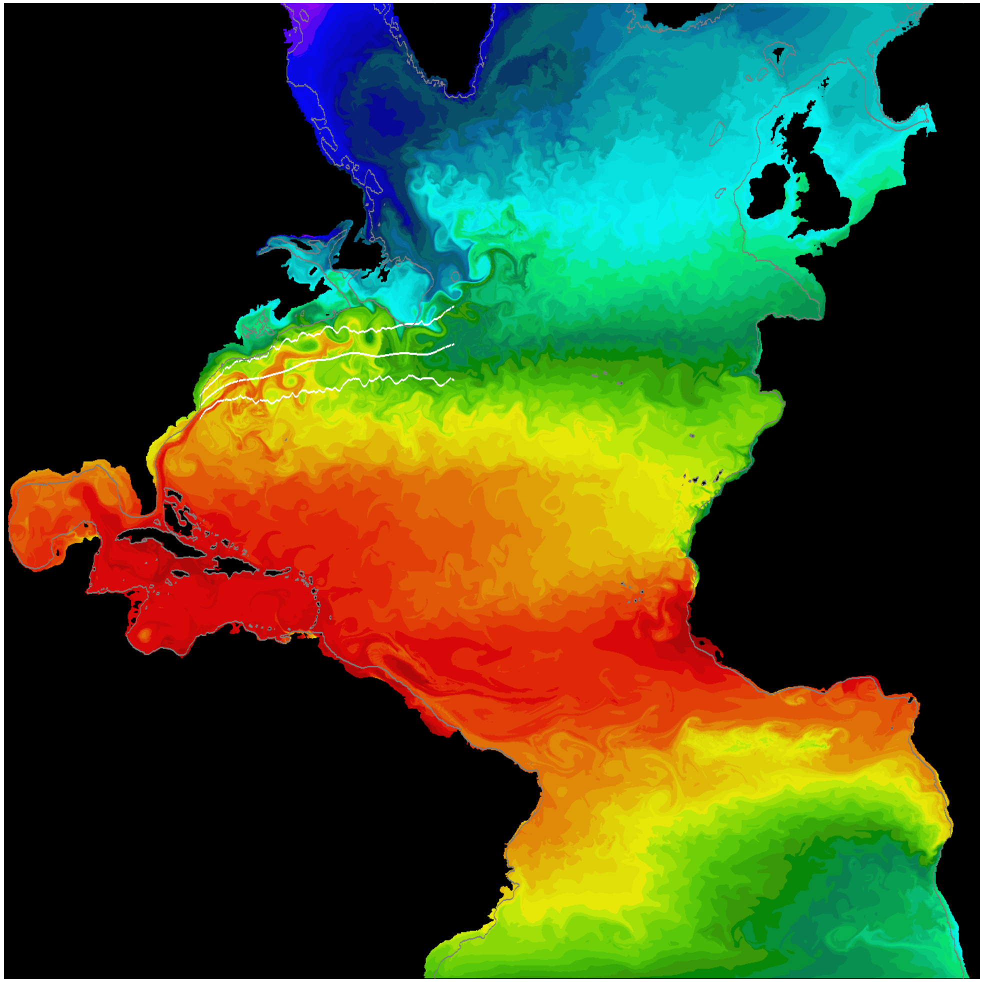
PDEs and Examples of Phenomena Modeled

- Ordinary differential equation: equation containing derivatives of a function of one variable
- Partial differential equation: equation containing derivatives of a function of two or more variables

Models:

- Air flow over an aircraft wing
- Blood circulation in human body
- Water circulation in an ocean
- Bridge deformations as its carries traffic
- Evolution of a thunderstorm
- Oscillations of a skyscraper hit by earthquake
- Strength of a toy
- Financial Markets

Model of Sea Surface Temperature in Atlantic Ocean



Courtesy MICOM group
at the Rosenstiel School
of Marine and Atmospheric
Science, University of Miami

Solving PDEs

- Finite element method
- Finite difference method (our focus)
  - Converts PDE into matrix equation
    - Linear system over discrete basis elements
  - Result is usually a sparse matrix
  - Matrix-based algorithms represent matrices explicitly
  - Matrix-free algorithms represent matrix values implicitly (our focus)

Class of Linear Second-order PDEs

- Linear second-order PDEs are of the form

$$Au_{xx} + 2Bu_{xy} + Cu_{yy} + Eu_x + Fu_y + Gu = H$$

where *A* - *H* are functions of *x* and *y* only

- Elliptic PDEs: $B^2 - AC < 0$

  (steady state heat equations)

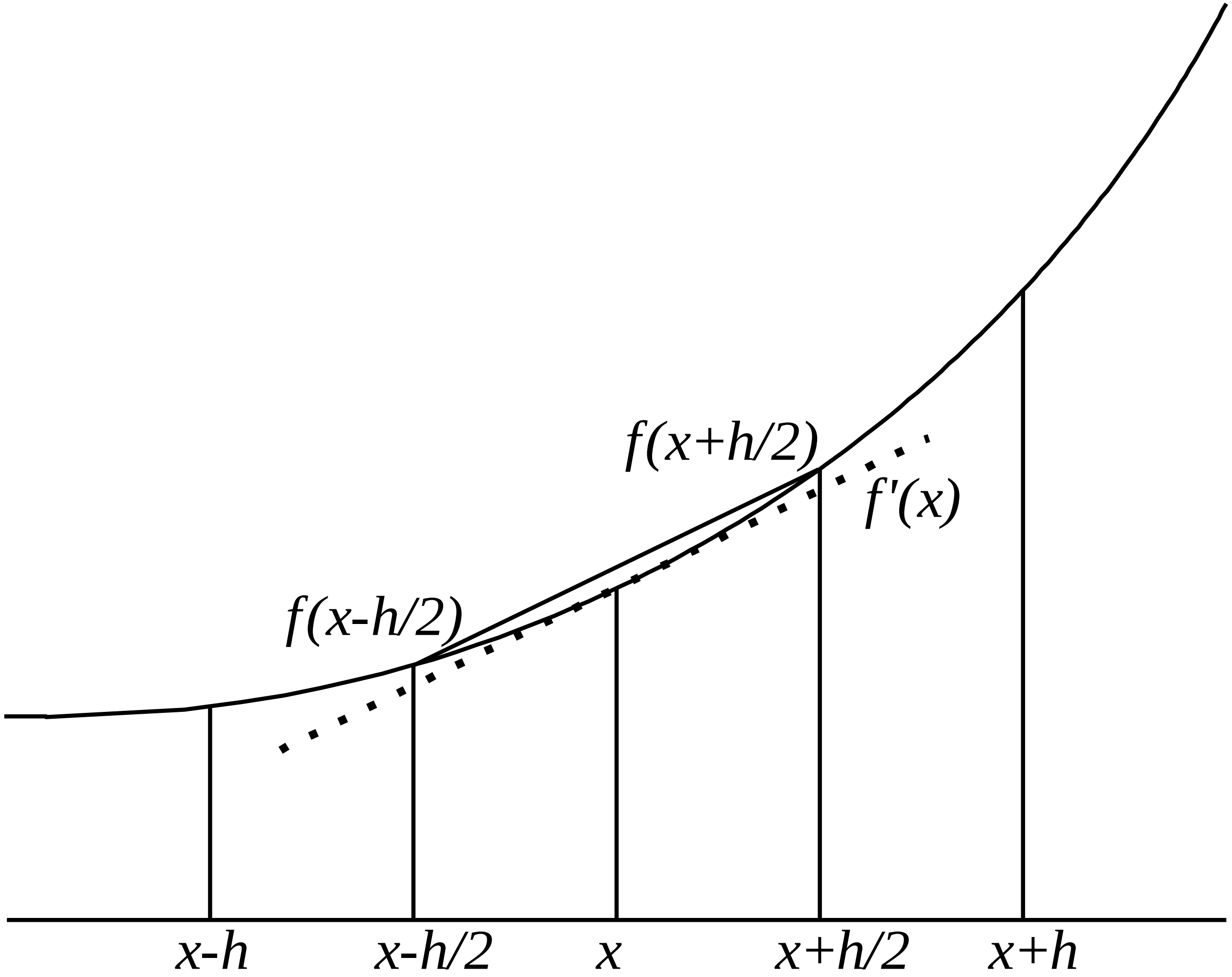
- Parabolic PDEs: $B^2 - AC = 0$

  (heat transfer equations)

- Hyperbolic PDEs: $B^2 - AC > 0$

  (wave equations)

Difference Quotients

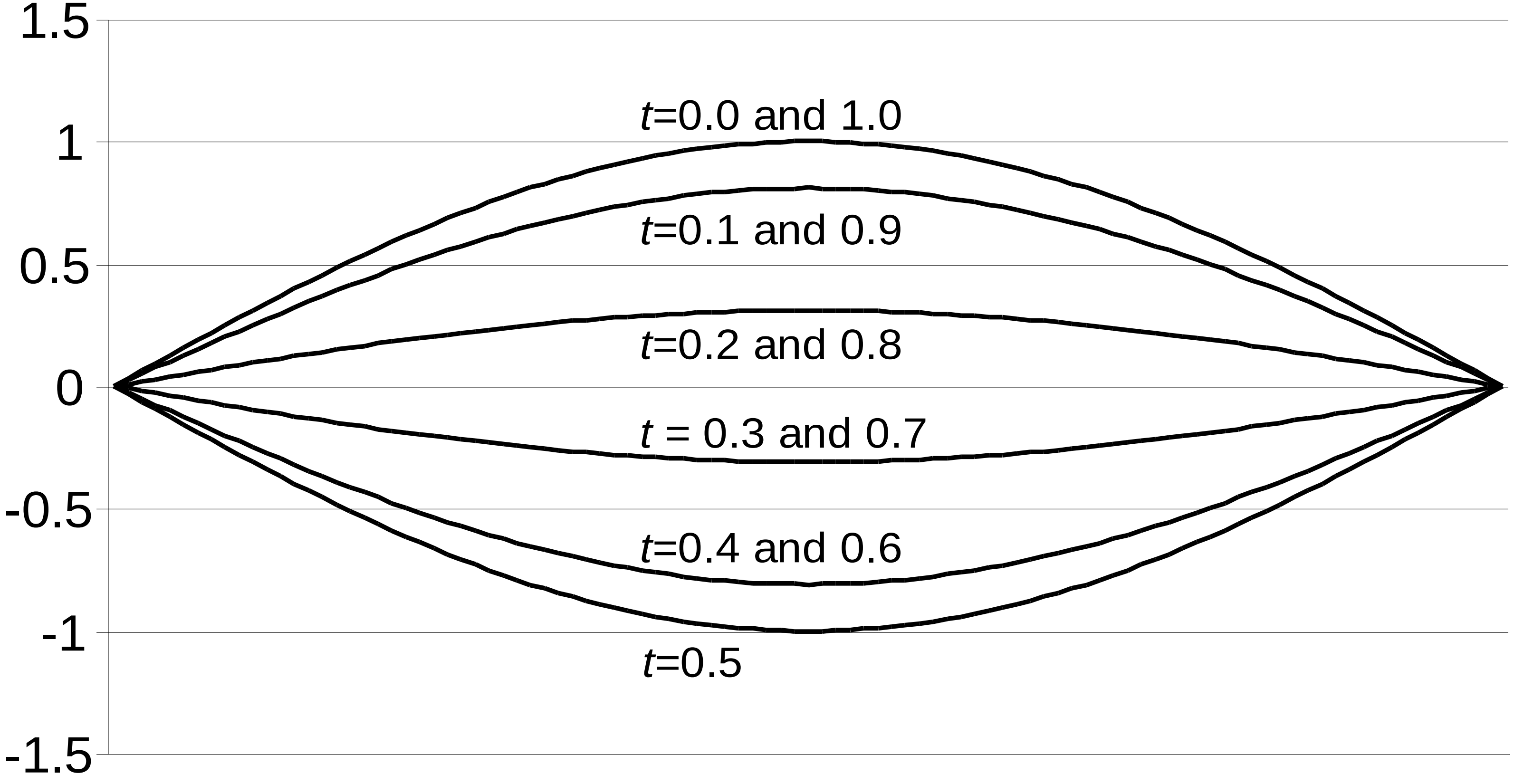


Formulas for 1st, 2d Derivatives

$$f'(x) \approx \frac{f(x+h/2) - f(x-h/2)}{h}$$

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Vibrating String Problem

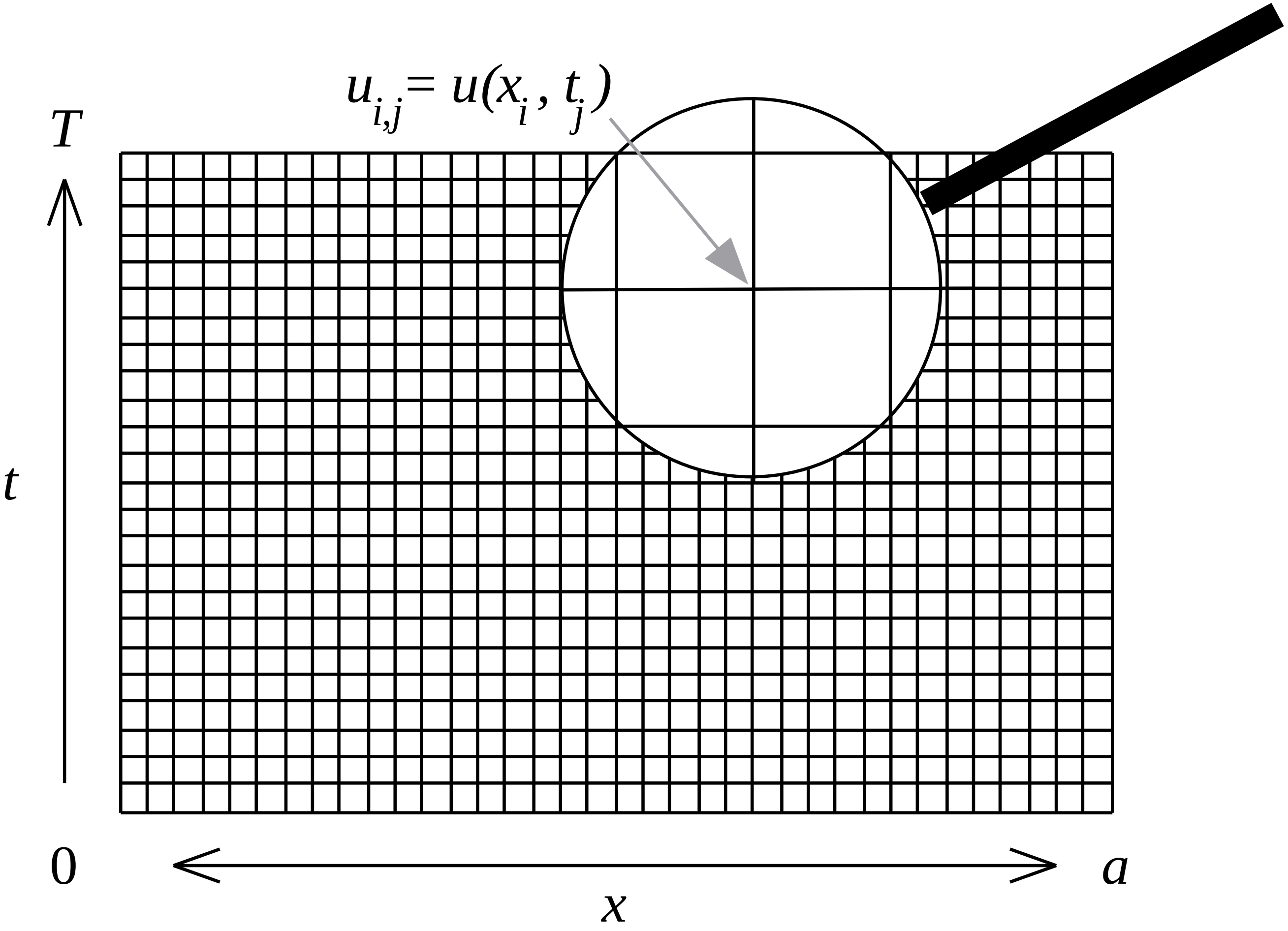


Vibrating string modeled by a hyperbolic PDE

Solution Stored in 2-D Array

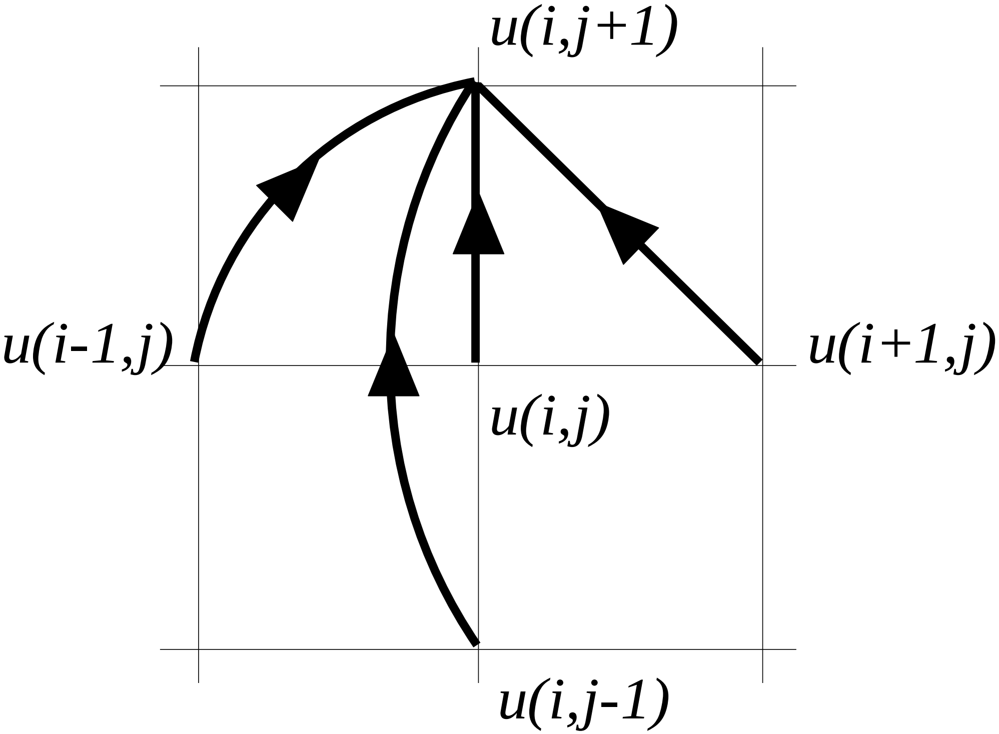
- Each row represents state of string at some point in time
- Each column shows how position of string at a particular point changes with time

Discrete Space, Time Intervals Lead to 2-D Array



Heart of Sequential C Program

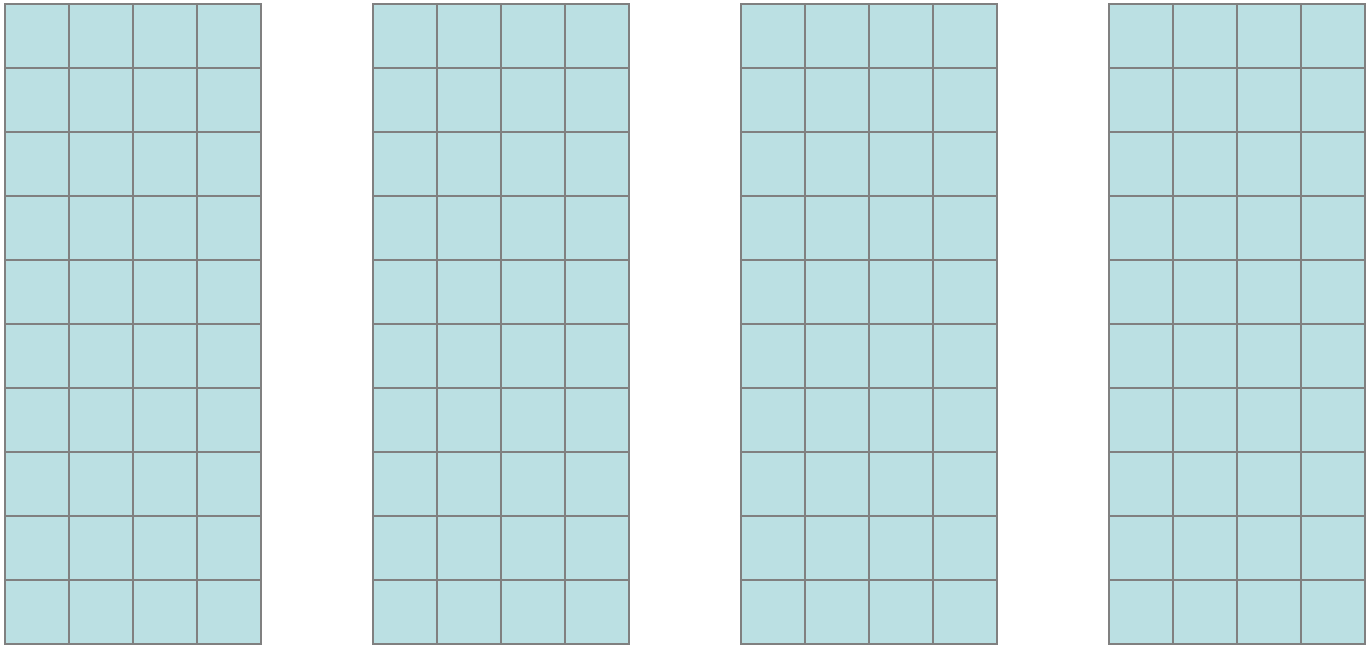
```
u[j+1][i] = 2.0*(1.0-L)*u[j][i] +
    L*(u[j][i+1] + u[j][i-1]) - u[j-1][i];
```



Parallel Program Design

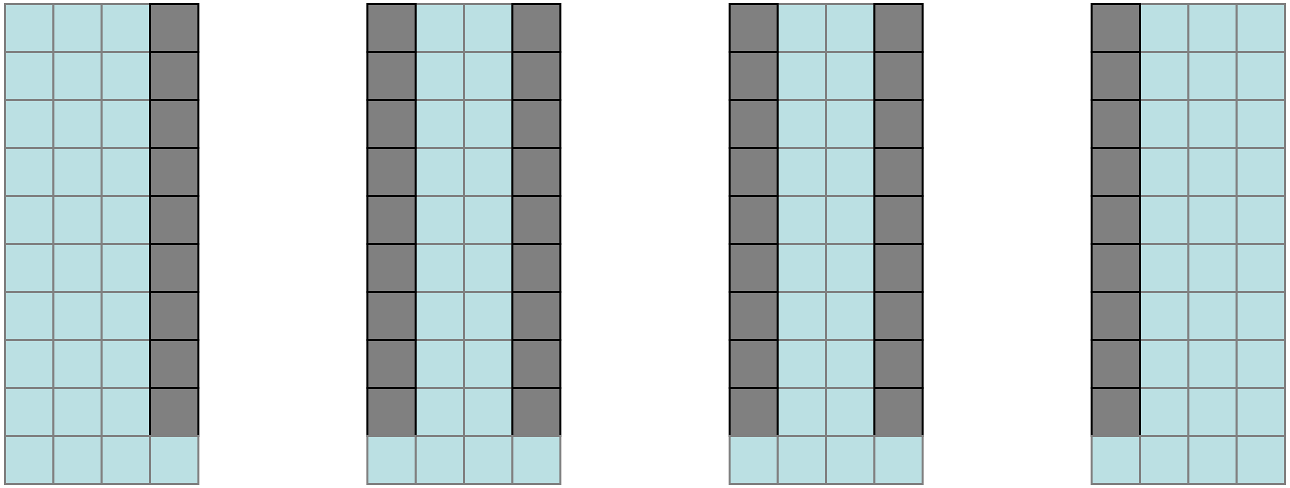
- Associate primitive task with each element of matrix
- Examine communication pattern
- Agglomerate tasks in same column
- Static number of identical tasks
- Regular communication pattern
- Strategy: agglomerate columns, assign one block of columns to each task

Result of Agglomeration and Mapping



Communication Still Needed

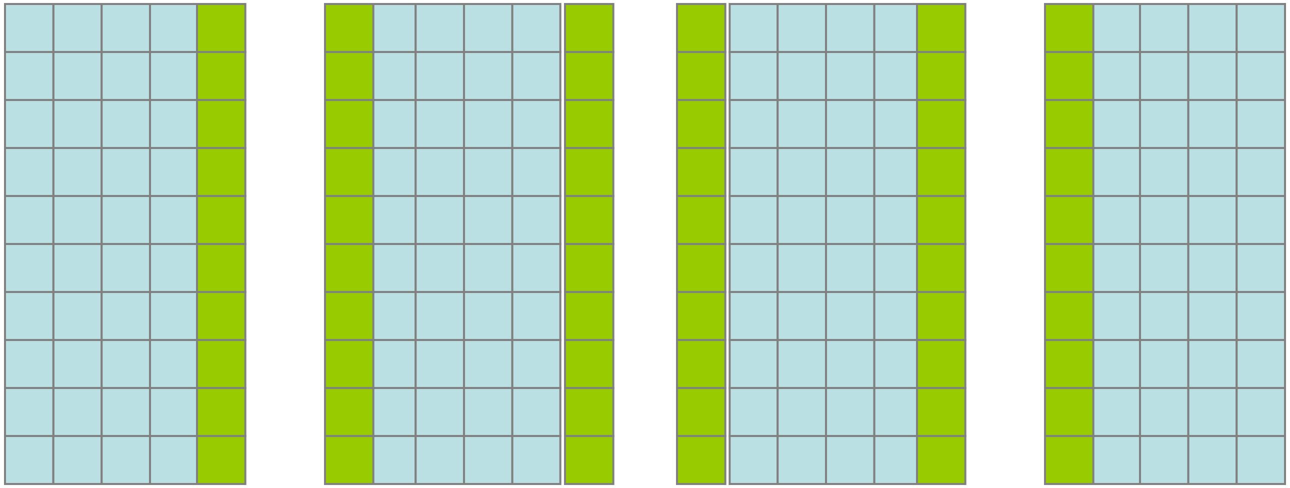
- Initial values (in lowest row) are computed without communication
- Values in black cells cannot be computed without access to values held by other tasks



Ghost Points

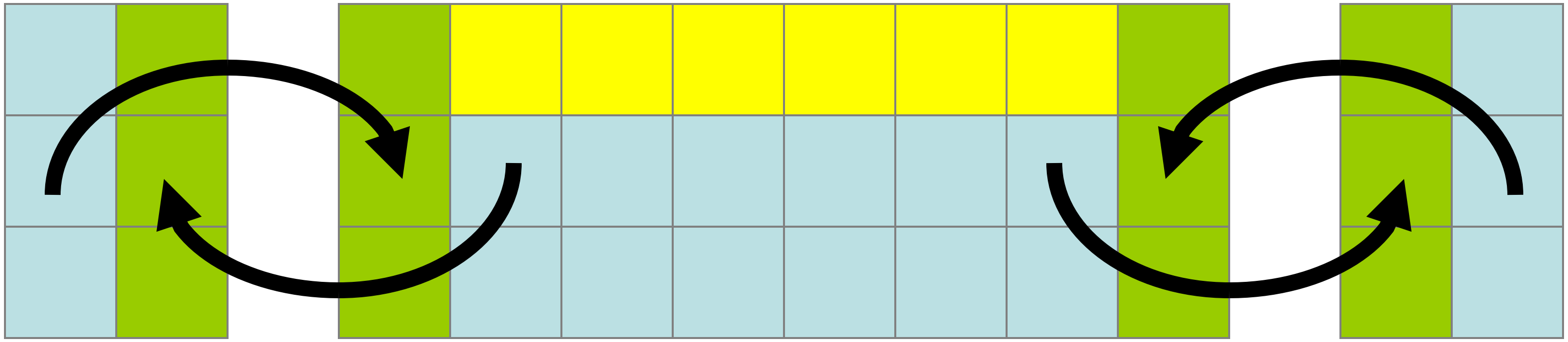
- Ghost points: memory locations used to store redundant copies of data held by neighboring processes
- Allocating ghost points as extra columns simplifies parallel algorithm by allowing same loop to update all cells

Matrices Augmented with Ghost Points



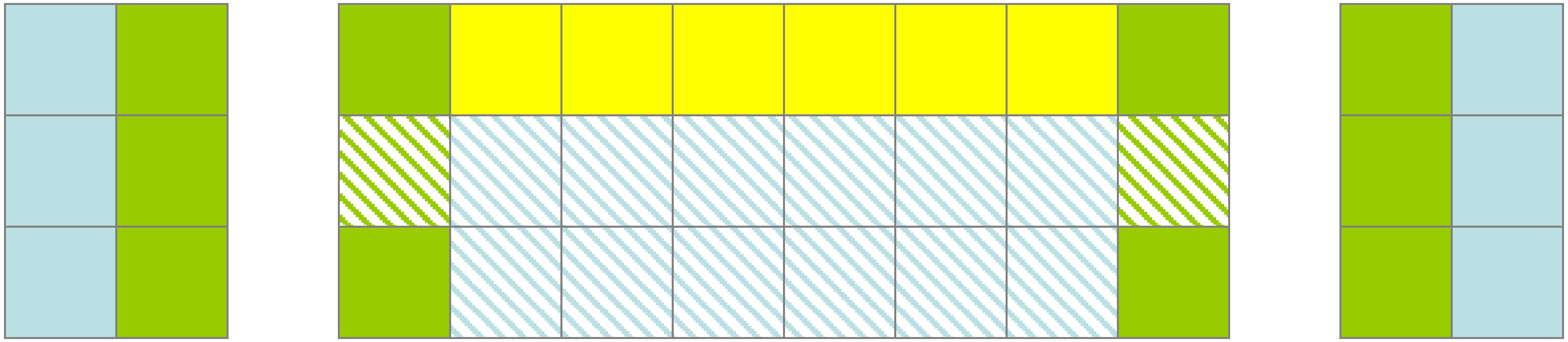
Green cells are the ghost points.

Communication in an Iteration



This iteration the process is responsible for computing the values of the yellow cells.

Computation in an Iteration



This iteration the process is responsible for computing the values of the yellow cells. The striped cells are the ones accessed as the yellow cell values are computed.

Complexity Analysis

- Computation time per element is constant, so sequential time complexity per iteration is $\Theta(n)$
- Elements divided evenly among processes, so parallel computational complexity per iteration is $\Theta(n / p)$
- During each iteration a process with an interior block sends two messages and receives two messages, so communication complexity per iteration is $\Theta(1)$

Isoefficiency Analysis

- Sequential time complexity: $\Theta(n)$
- Parallel overhead: $\Theta(p)$
- Isoefficiency relation:

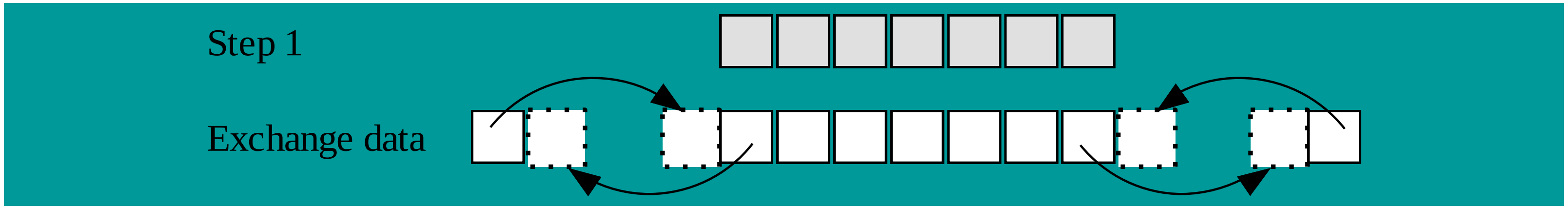
  $n \geq Cp$
- To maintain the same level of efficiency, *n* must increase at the same rate as *p*
- If $M(n) = n^2$, algorithm has poor scalability
- If matrix of 3 rows rather than *m* rows is used, $M(n) = n$ and system is perfectly scalable

Replicating Computations

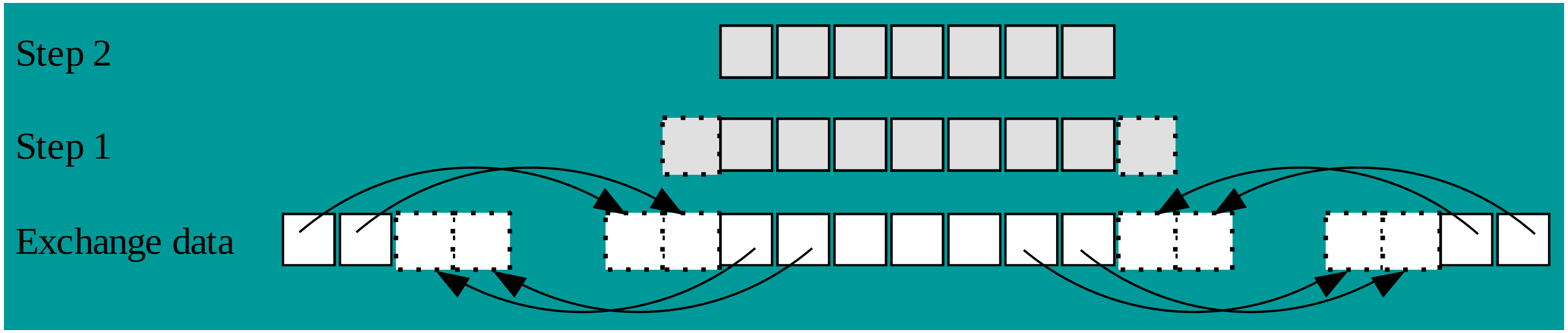
- If only one value transmitted, communication time dominated by message latency
- We can reduce number of communications by replicating computations
- If we send two values instead of one, we can advance simulation two time steps before another communication

Replicating Computations

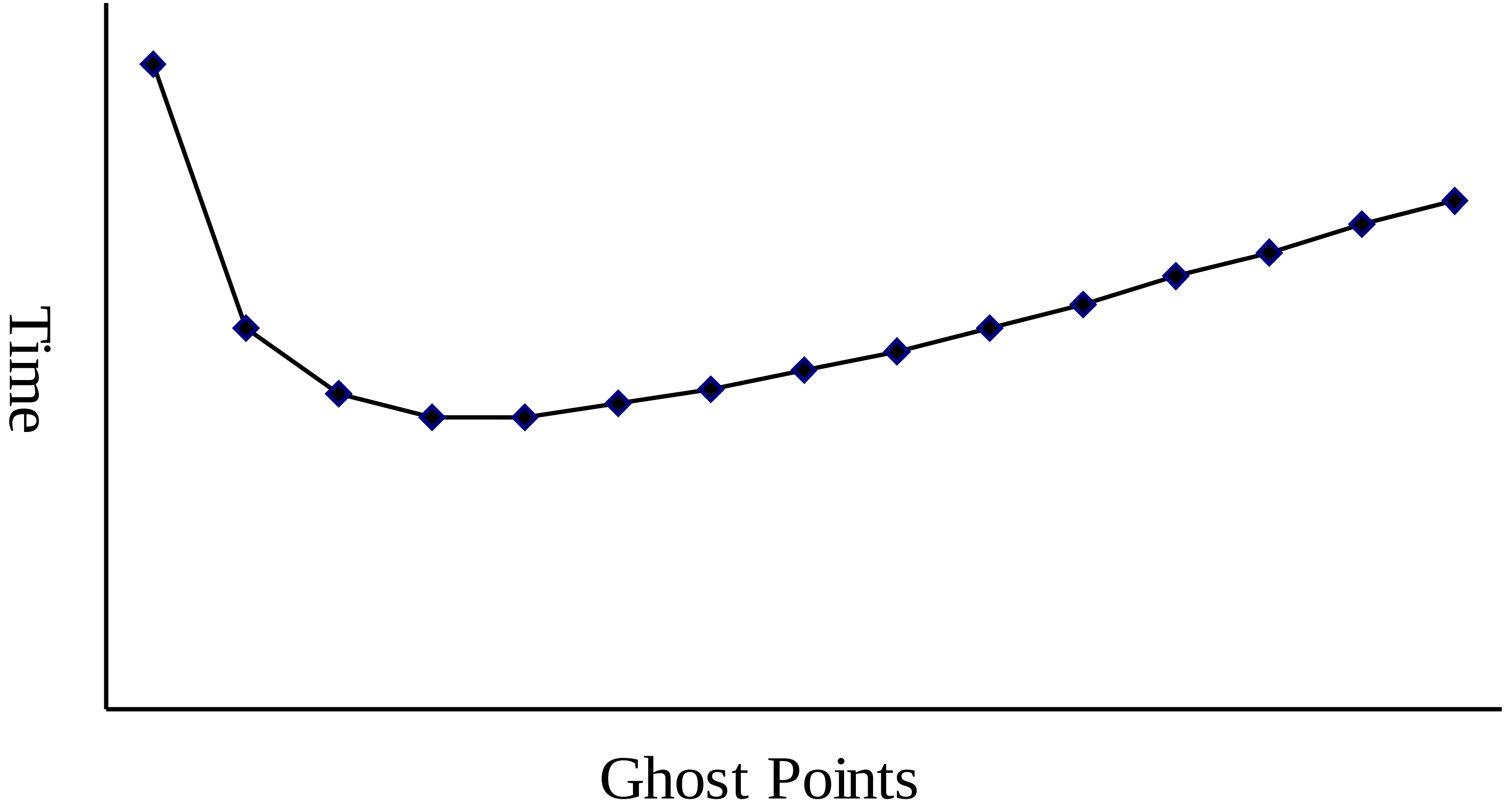
Without replication:



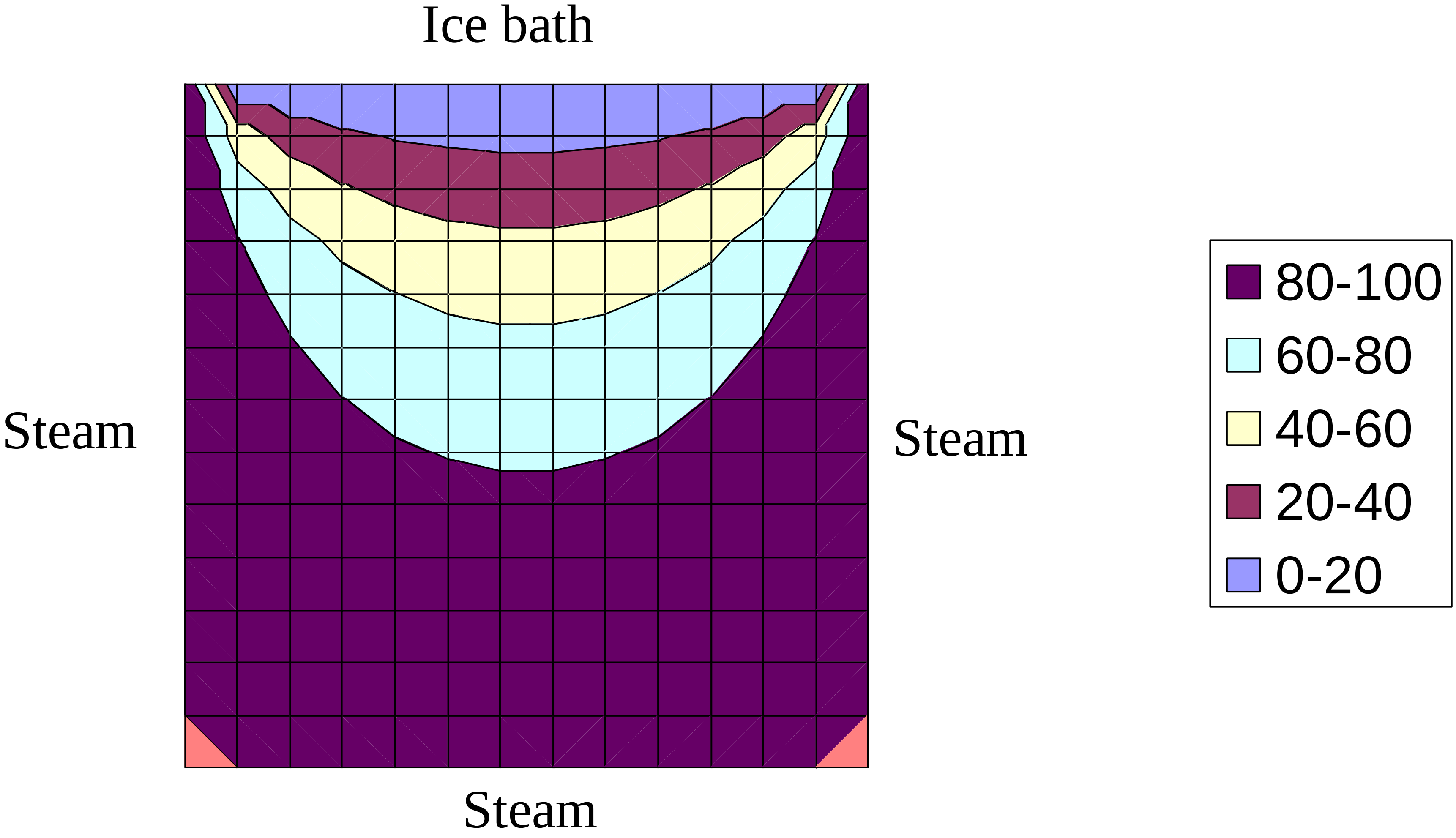
With replication:



Communication Time vs. Number of Ghost Points



Next Case: Steady State Heat Distribution Problem



Solving the Problem

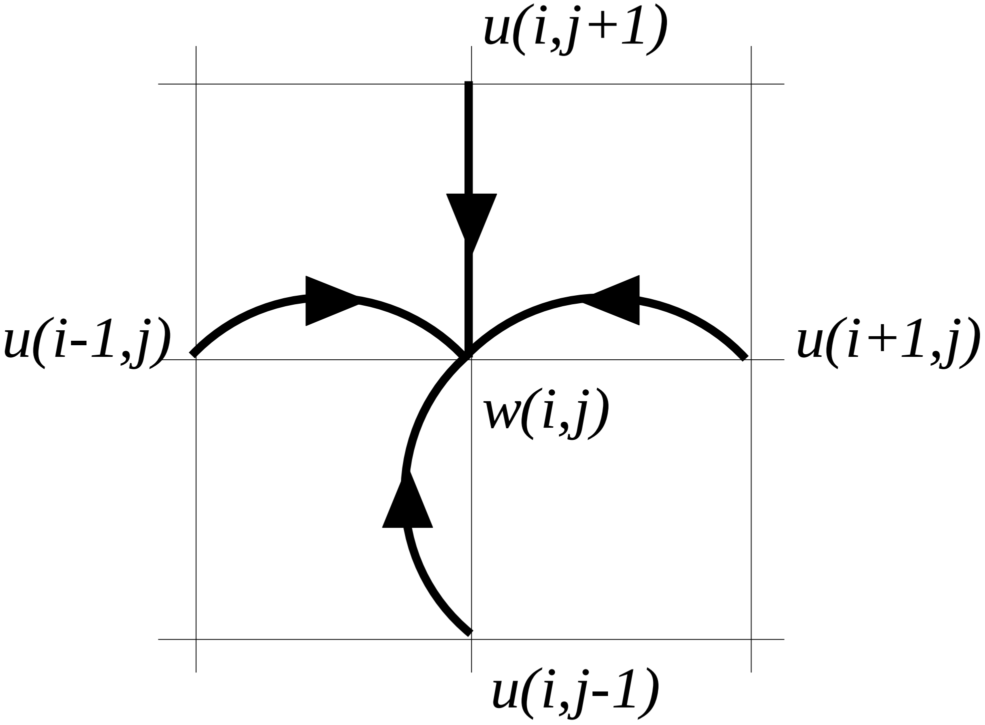
- Underlying PDE is the Poisson equation

$$u_{xx} + u_{yy} = f(x, y)$$

- This is an example of an elliptical PDE
- Will create a 2-D grid
- Each grid point represents value of state state solution at particular (*x*, *y*) location in plate

Heart of Sequential C Program

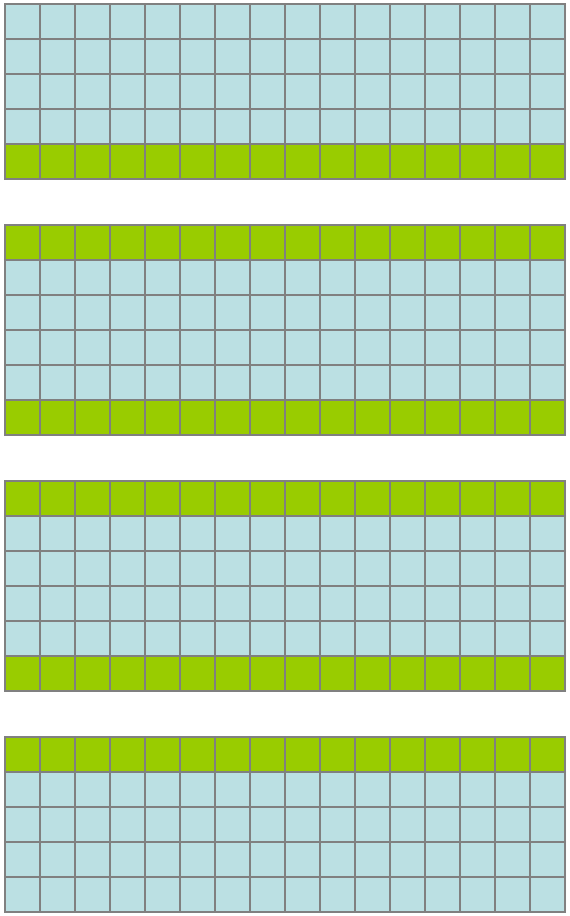
```
w[i][j] = (u[i-1][j] + u[i+1][j] +
                    u[i][j-1] + u[i][j+1]) / 4.0;
```



Parallel Algorithm 1

- Associate primitive task with each matrix element
- Agglomerate tasks in contiguous rows (rowwise block striped decomposition)
- Add rows of ghost points above and below rectangular region controlled by process

Example Decomposition



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)

Isoefficiency Analysis

- Sequential time complexity: $\Theta(n^2)$
- Parallel overhead: $\Theta(pn)$
- Isoefficiency relation:
  $n^2 \geq Cnp \Rightarrow n \geq Cp$

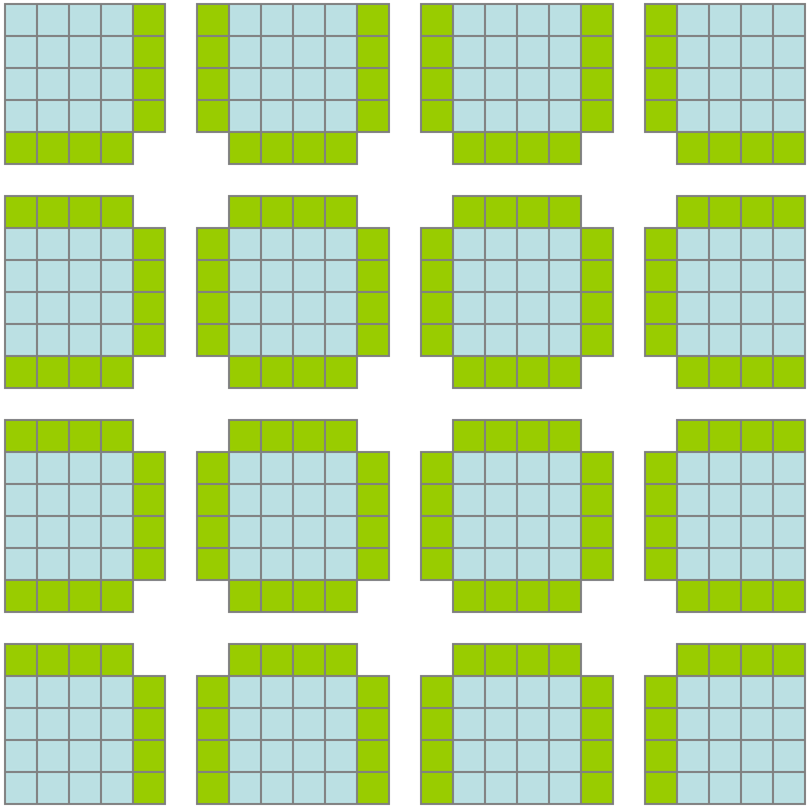
$$M(Cp)/p = C^2p^2/p = C^2p$$

- This implementation has poor scalability

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 of ghost points to all four sides of rectangular region controlled by 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 must be used when receiving these ghost point values
- Similarly, buffer must be used when sending column of values to a neighboring process

Complexity Analysis

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

Isoefficiency Analysis

- Sequential time complexity: $\Theta(n^2)$
- Parallel overhead: $\Theta(n \sqrt{p})$
- Isoefficiency relation: $n^2 \geq Cn \sqrt{p} \Rightarrow n \geq C \sqrt{p}$

$$M(C\sqrt{p})/p = C^2 p/p = C^2$$

- This system is perfectly scalable

Summary (1/4)

- PDEs used to model behavior of a wide variety of physical systems
- Realistic problems yield PDEs too difficult to solve analytically, so scientists solve them numerically
- Two most common numerical techniques for solving PDEs
  - finite element method
  - finite difference method

Summary (2/4)

- Finite difference methods
  - Matrix-based methods store matrix explicitly
  - Matrix-free implementations store matrix implicitly
- We have designed and analyzed parallel algorithms based on matrix-free implementations

Summary (4/4)

- Ghost points store copies of values held by other processes
- Explored increasing number of ghost points and replicating computation in order to reduce number of message exchanges
- Optimal number of ghost points depends on characteristics of parallel system