The Temperature Problem

An area has known temperatures along each of its edges.

Find the temperature distribution within.

Divide area into fine mesh of points, $h_{i,j}$.

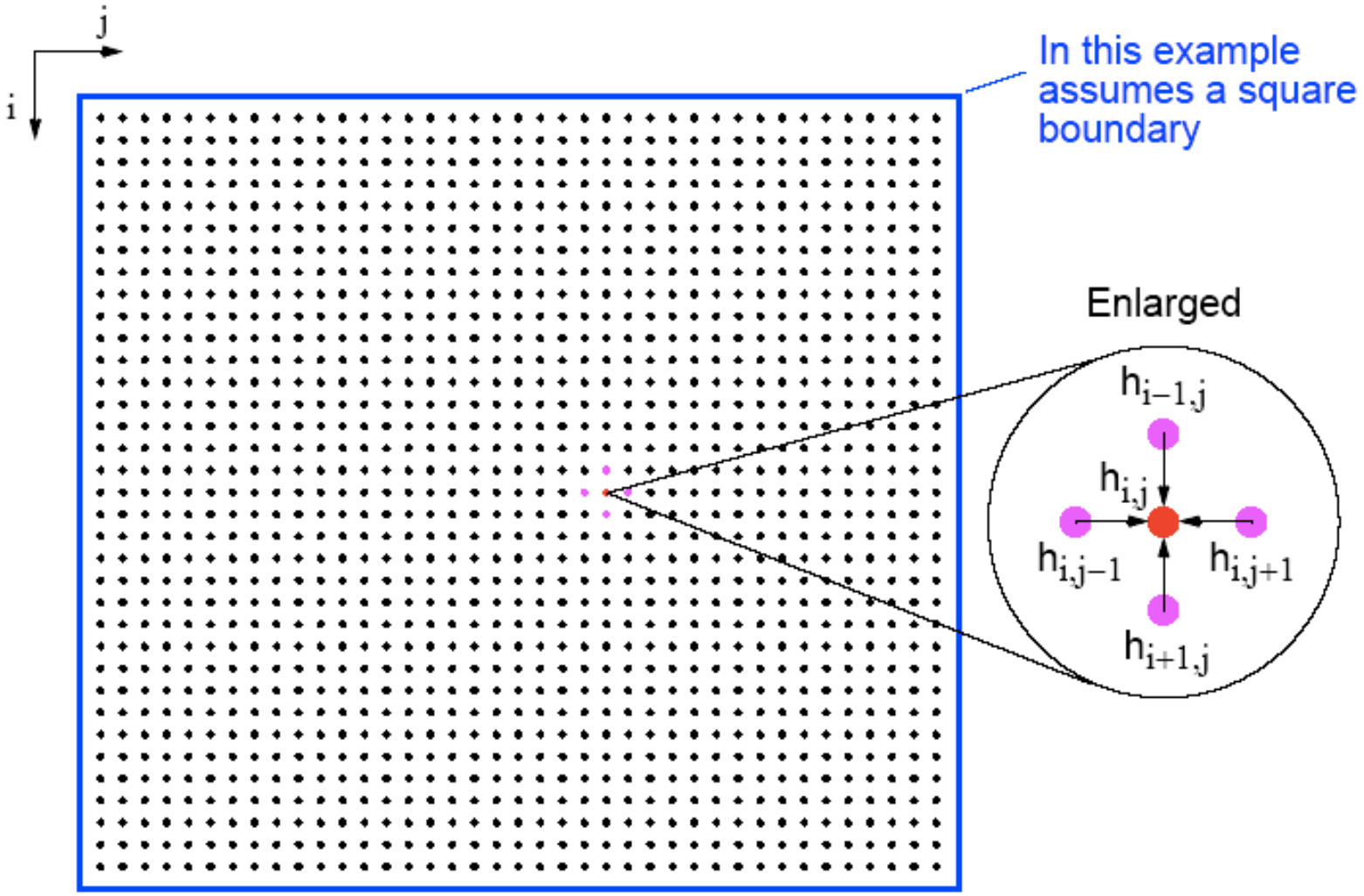
Temperature at an inside point taken to be average of temperatures of four neighboring points. Convenient to describe edges by points.

Temperature of each point by iterating the equation:

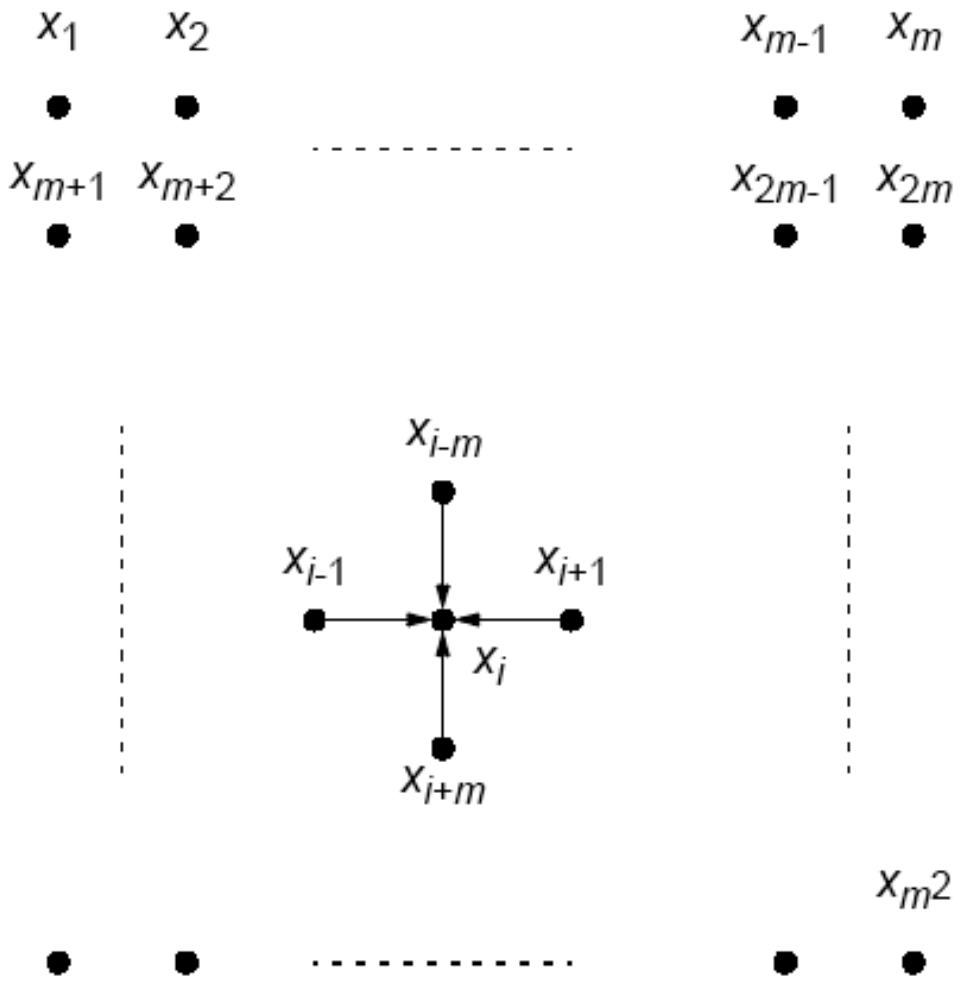
$$h_{i,j} = \frac{h_{i-1,j} + h_{i+1,j} + h_{i,j-1} + h_{i,j+1}}{4}$$

($0 < i < n$, $0 < j < n$) for a fixed number of iterations or until the difference between iterations less than some very small amount.

Temperature Problem



Natural ordering of temperature problem



Number points from 1 for convenience and include those representing the edges. Each point will then use the equation

$$x_i = \frac{x_{i-1} + x_{i+1} + x_{i-m} + x_{i+m}}{4}$$

Could be written as a linear equation containing the unknowns *xi-m*, *xi*-1, *xi*+1, and *xi*+*m*:

$$x_{i-m} + x_{i-1} - 4x_i + x_{i+1} + x_{i-m} = 0$$

Notice: solving a (sparse) system
Also solving Laplace's equation.

Sequential Code

Using a fixed number of iterations

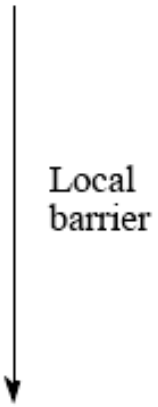
```
for (iteration = 0; iteration < limit; iteration++) {
   for (i = 1; i < n; i++)
      for (j = 1; j < n; j++)
          g[i][j] = 0.25*(h[i-1][j]+h[i+1][j]+h[i][j-1]+h[i][j+1]);
      for (i = 1; i < n; i++)                          /* update points */
        for (j = 1; j < n; j++)
          h[i][j] = g[i][j];
}
```

using original numbering system (*n* x *n* array).

Parallel Code

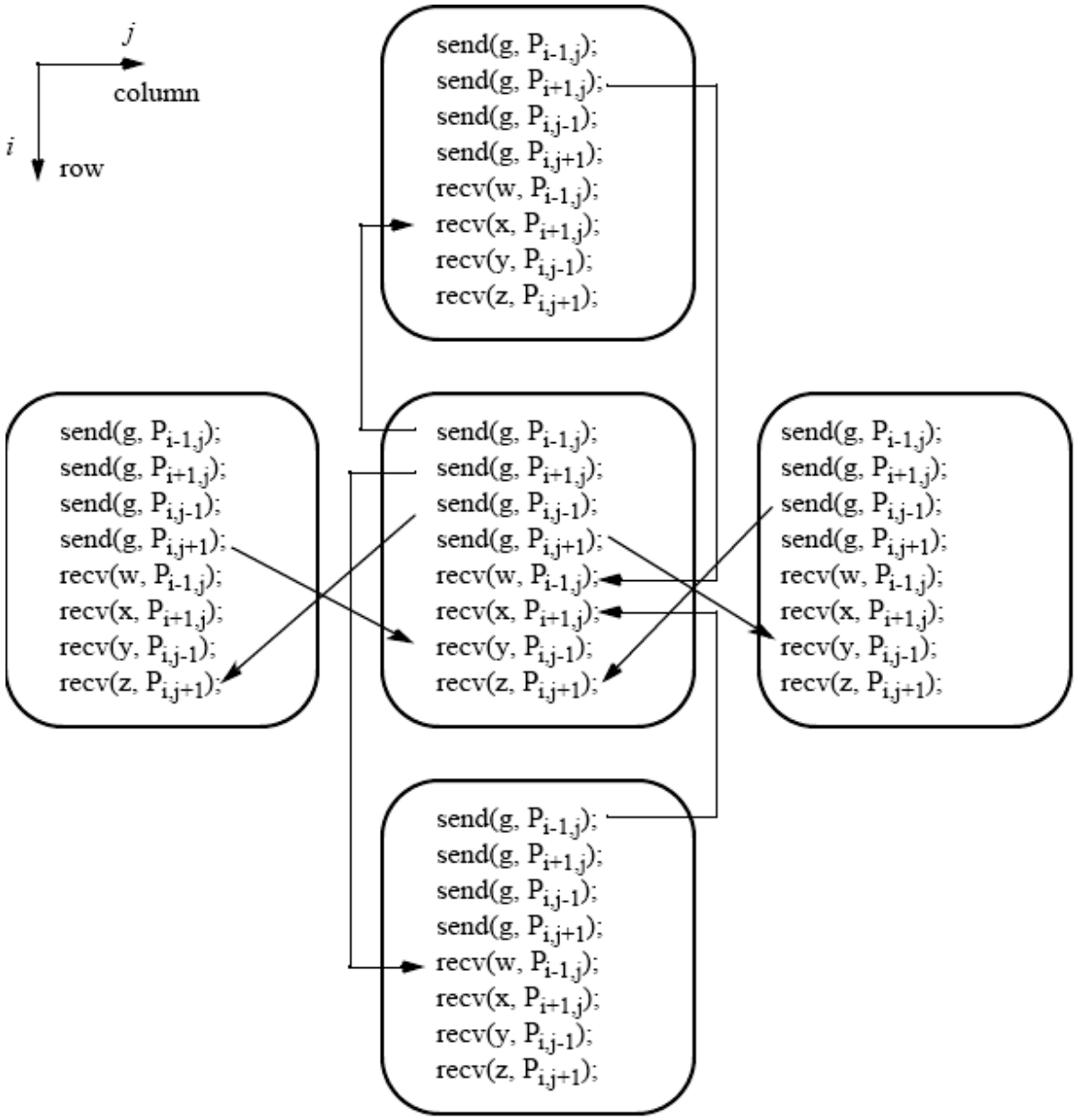
With fixed number of iterations, $P_{i,j}$ (except for the boundary points):

```
for (iteration = 0; iteration < limit; iteration++) {
    g = 0.25 * (w + x + y + z);
    send(&g, P_{i-1,j});   /* non-blocking sends */
    send(&g, P_{i+1,j});
    send(&g, P_{i,j-1});
    send(&g, P_{i,j+1});
    recv(&w, P_{i-1,j});   /* synchronous receives */
    recv(&x, P_{i+1,j});
    recv(&y, P_{i,j-1});
    recv(&z, P_{i,j+1});
}
```



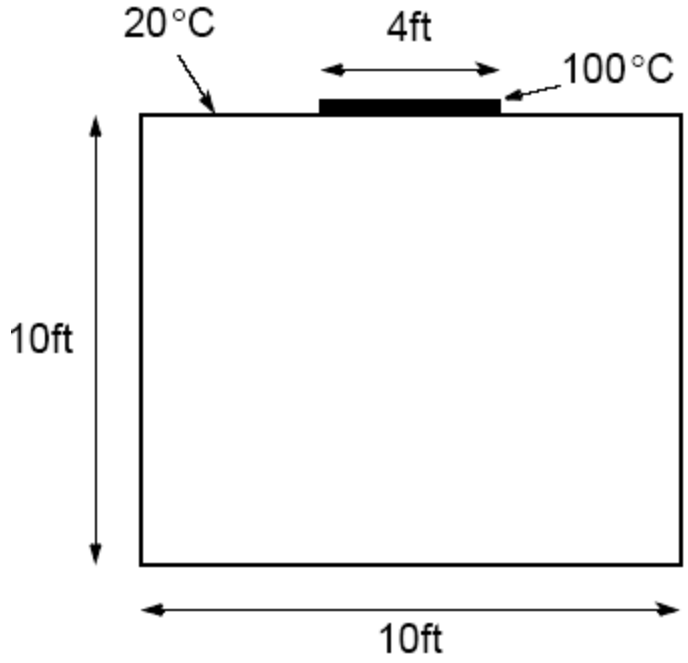
Important to use **send()**s that do not block while waiting for **recv()**s; otherwise processes would deadlock, each waiting for a **recv()** before moving on - **recv()**s must be synchronous and wait for **send()**s.

Message passing for temperature problem



Example

A room has four walls and a fireplace. Temperature of wall is 20°C, and temperature of fireplace is 100°C. Write a parallel program using Jacobi iteration to compute the temperature inside the room and plot (preferably in color) temperature contours at 10°C intervals using Xlib calls or similar graphics calls as available on your system.

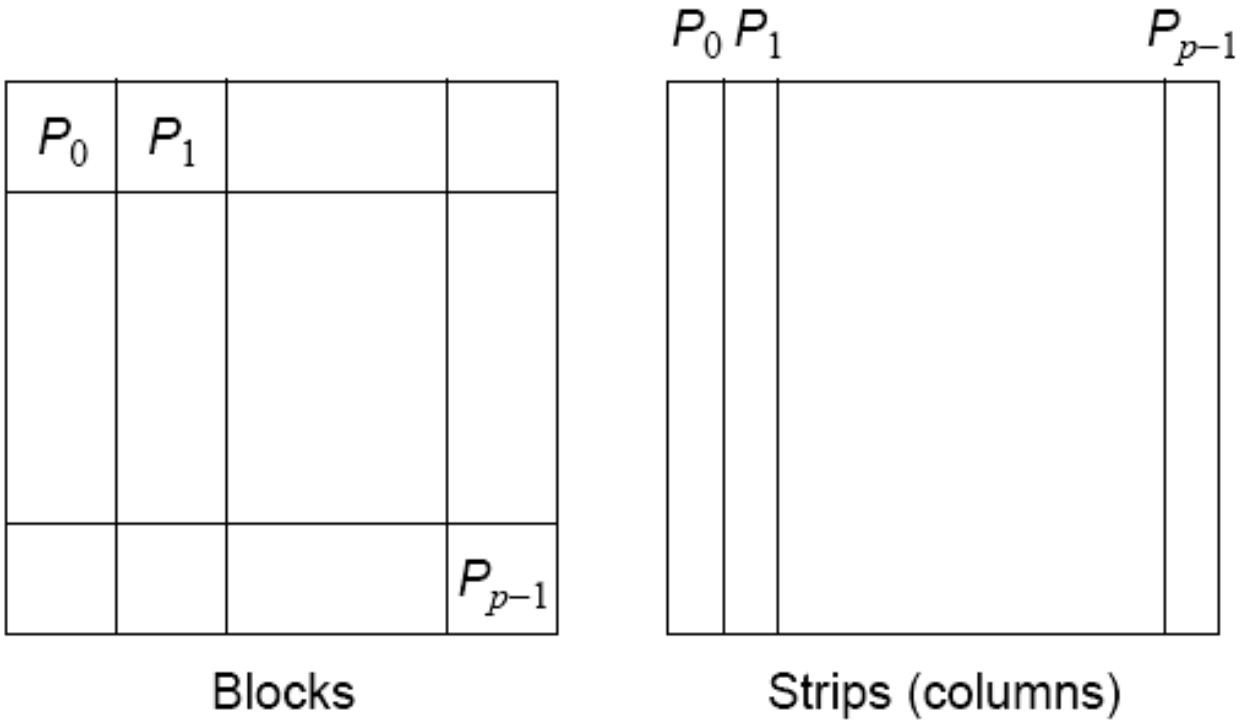


Sample student output

Partitioning

Normally allocate more than one point to each processor, because many more points than processors.

Points could be partitioned into square blocks or strips:

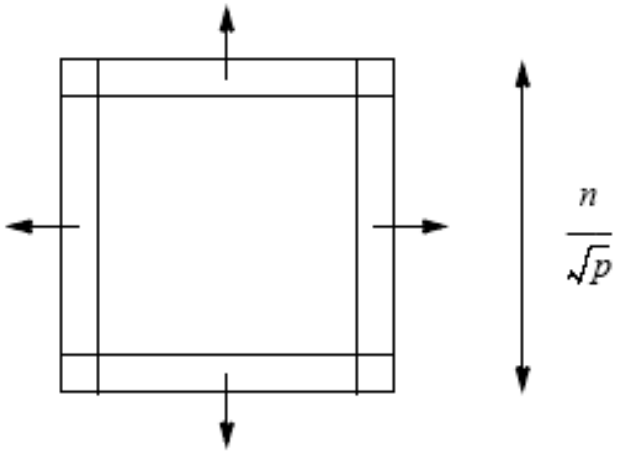


Block partition

Four edges where data points exchanged.

Communication time given by

$$t_{\text{commsq}} = 8\left(t_{\text{startup}} + \frac{n}{\sqrt{p}} t_{\text{data}}\right)$$



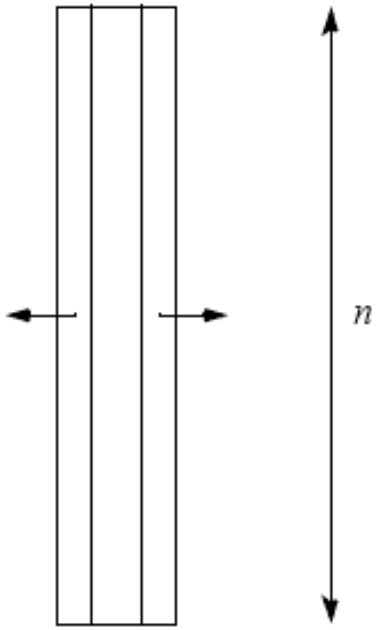
Square blocks

Strip partition

Two edges where data points are exchanged.

Communication time is given by

$$t_{\text{commcol}} = 4(t_{\text{startup}} + nt_{\text{data}})$$



Strips

Optimum

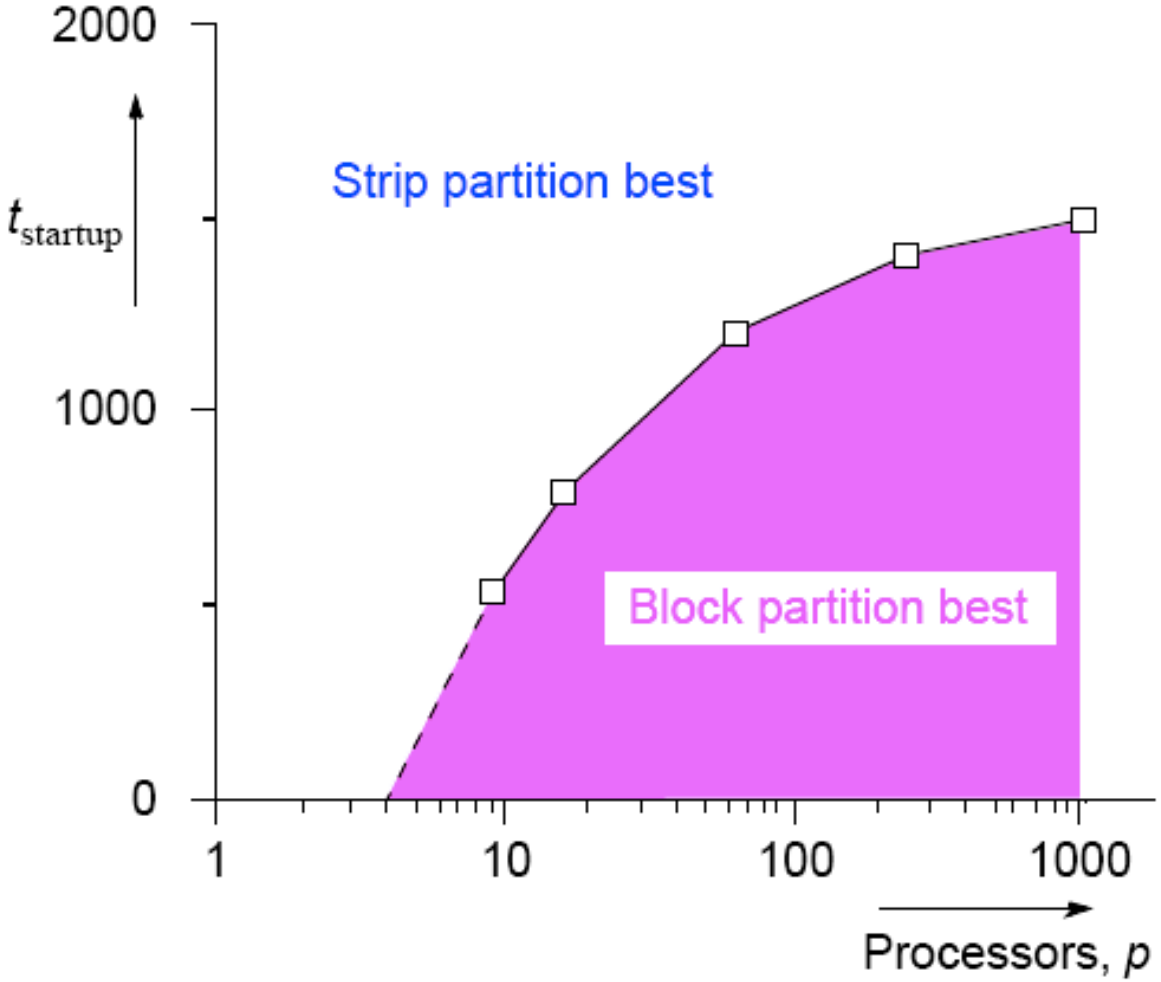
In general, strip partition best for large startup time, and block partition best for small startup time.

With the previous equations, block partition has a larger communication time than strip partition if

$$t_{\text{startup}} > n\left(1 - \frac{2}{\sqrt{p}}\right) t_{\text{data}}$$

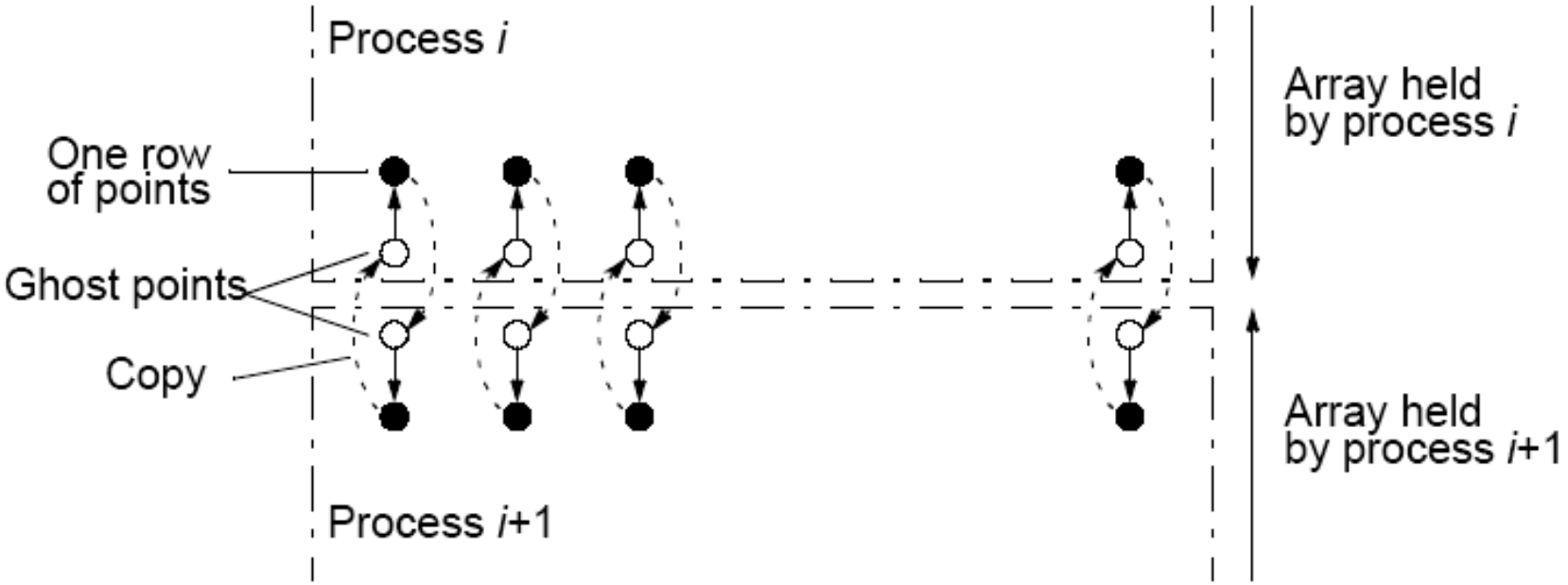
($p \geq 9$).

Startup times for block and strip partitions



Ghost Points

Additional row of points at each edge that hold values from adjacent edge. Each array of points increased to accommodate ghost rows.



Relationship of Matrices to Linear Equations

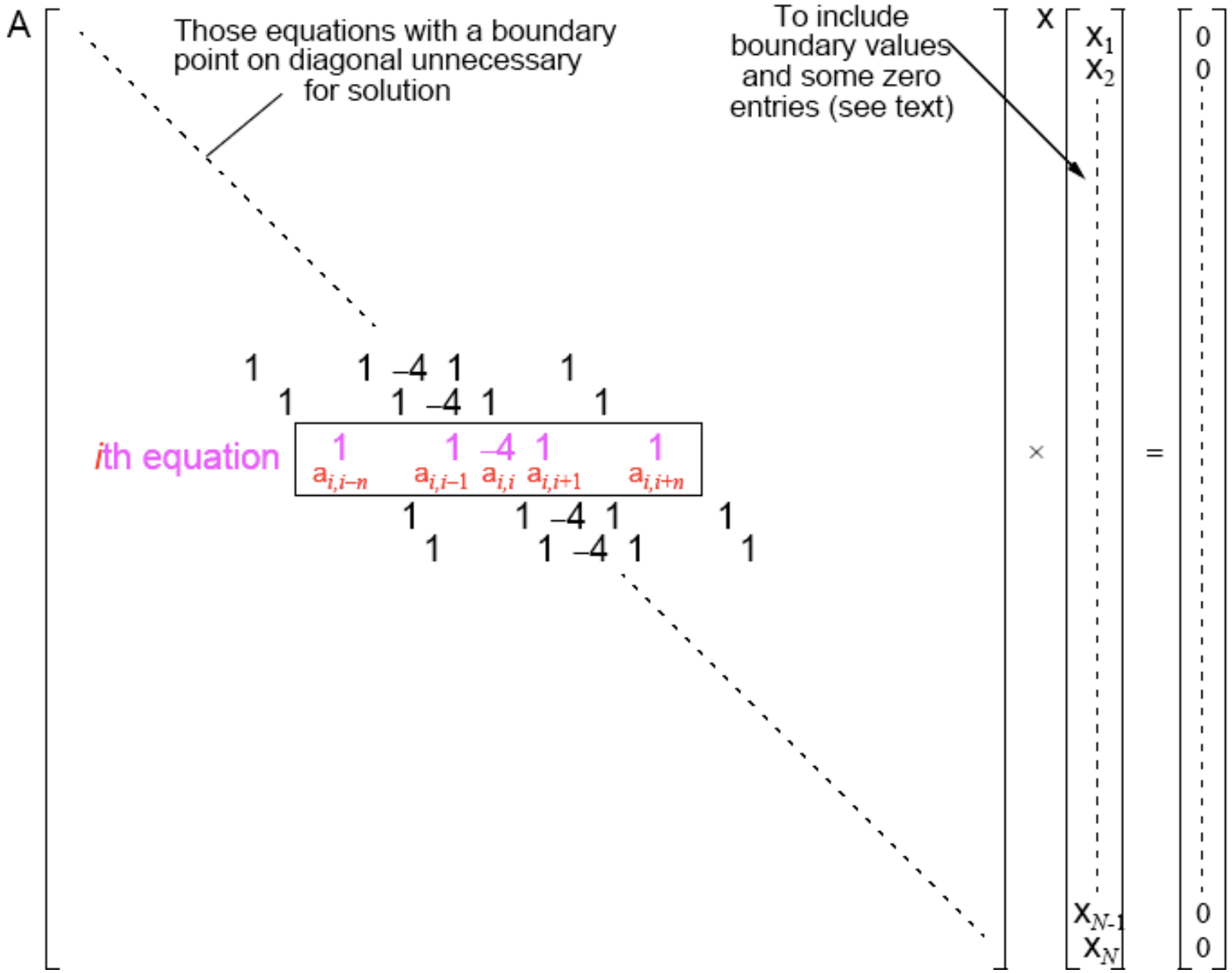
A system of linear equations can be written in matrix form:

$$\mathbf{Ax} = \mathbf{b}$$

Matrix **A** holds the *a* constants

x is a vector of the unknowns

b is a vector of the *b* constants.



Jacobi Iteration

Iteration formula - *i*th equation rearranged to have *i*th unknown on left side:

$$x_i^k = \frac{1}{a_{i,i}}\left[b_i - \sum_{j \neq i} a_{i,j} x_j^{k-1}\right]$$

Superscript indicates iteration:

x_i^k *is* kth iteration of x_i , x_j^{k-1} is $(k-1)$th iteration of x_j.

Example of a Sparse System of Linear Equations:

Laplace's Equation

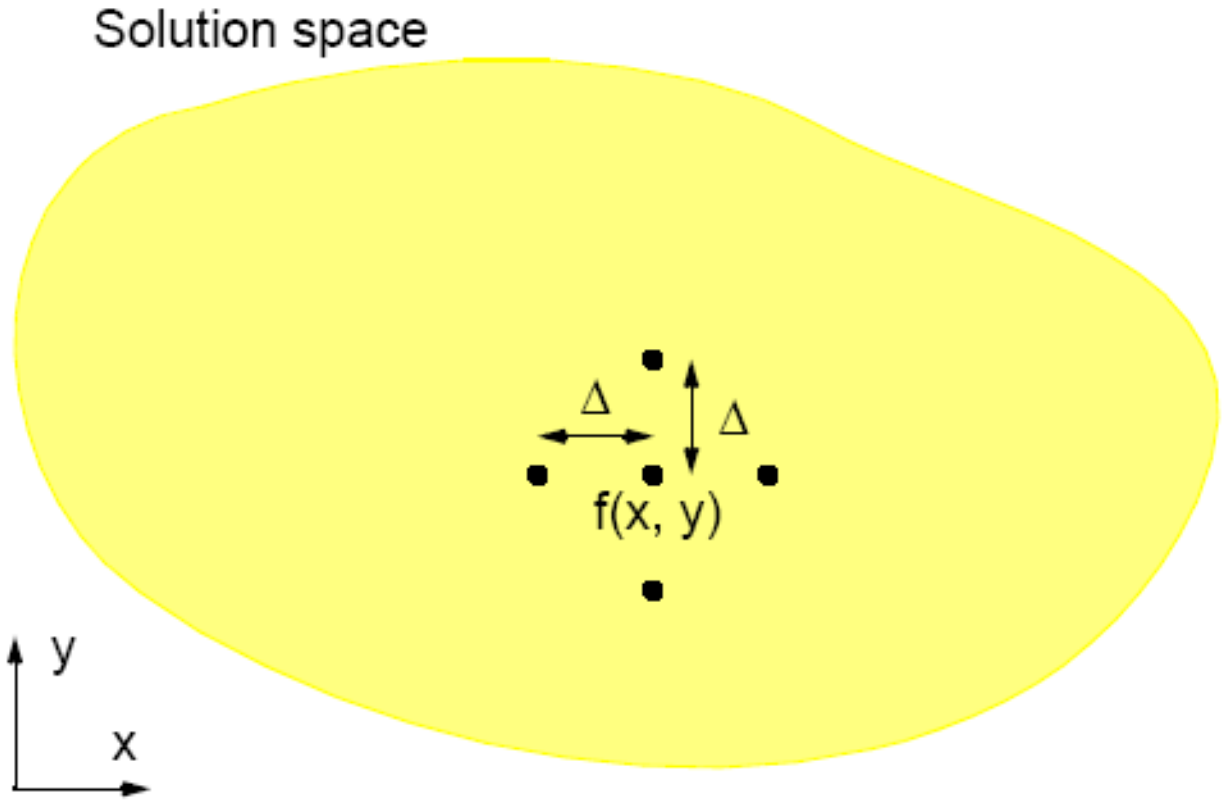
$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} = 0$$

Solve for *f* over the two-dimensional x-y space.

For a computer solution, *finite difference* methods are appropriate

Two-dimensional solution space is "discretized" into a large number of solution points.

Finite Difference Method



If distance between points, Δ, made small enough:

$$\frac{\partial^2 f}{\partial x^2} \approx \frac{1}{\Delta^2}[f(x+\Delta, y) - 2f(x, y) + f(x-\Delta, y)]$$

$$\frac{\partial^2 f}{\partial y^2} \approx \frac{1}{\Delta^2}[f(x, y+\Delta) - 2f(x, y) + f(x, y-\Delta)]$$

Substituting into Laplace's equation, we get

$$\frac{1}{\Delta^2}[f(x+\Delta, y) + f(x-\Delta, y) + f(x, y+\Delta) + f(x, y-\Delta) - 4f(x, y)] = 0$$

Rearranging, we get

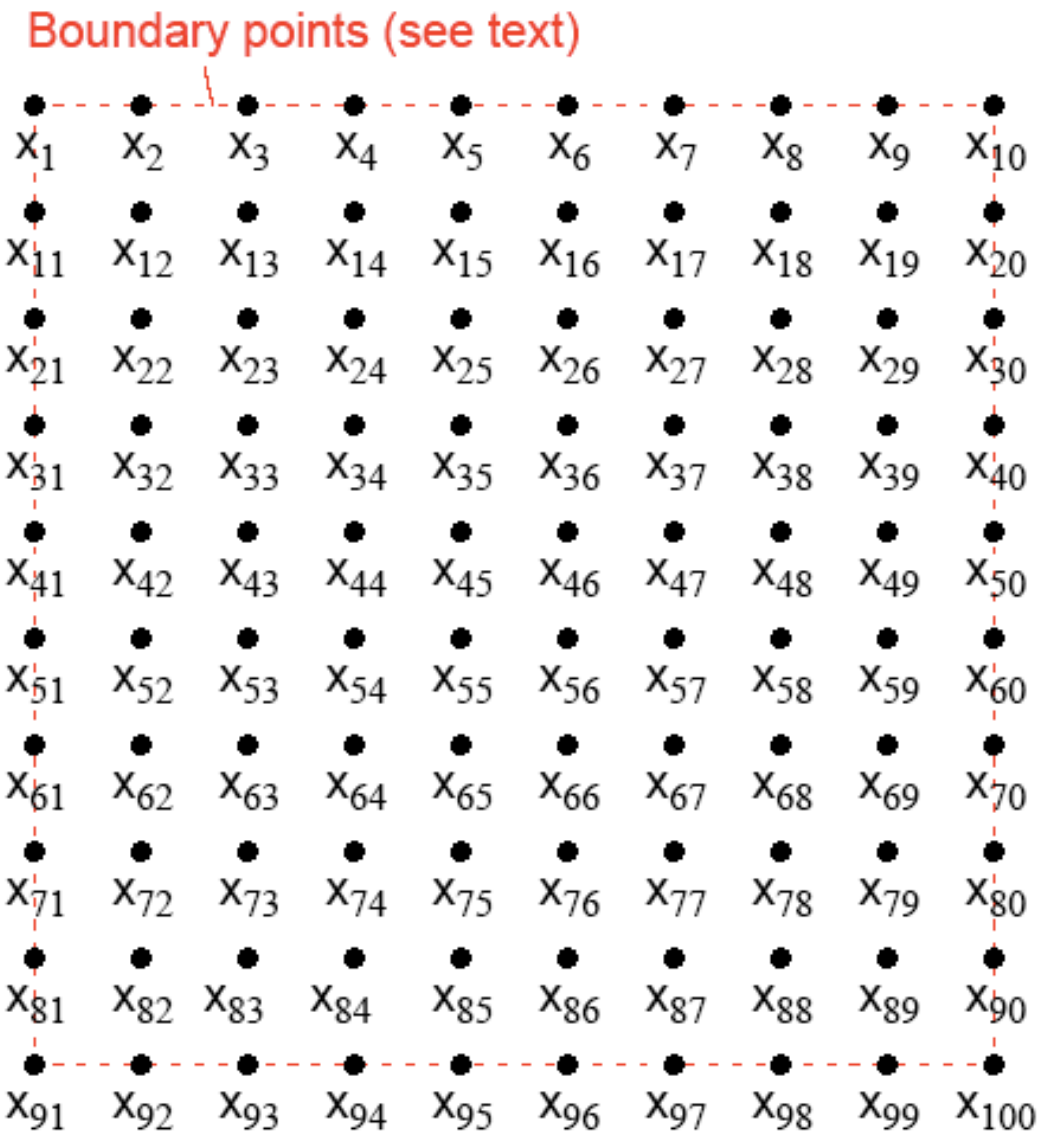
$$f(x, y) = \frac{[f(x-\Delta, y) + f(x, y-\Delta) + f(x+\Delta, y) + f(x, y+\Delta)]}{4}$$

Rewritten as an iterative formula:

$$f^k(x, y) = \frac{[f^{k-1}(x-\Delta, y) + f^{k-1}(x, y-\Delta) + f^{k-1}(x+\Delta, y) + f^{k-1}(x, y+\Delta)]}{4}$$

$f^k(x, y)$ - kth iteration, $f^{k-1}(x, y)$ - $(k-1)$th iteration.

Natural Order



Relationship with a General System of Linear Equations

Using natural ordering, *i*th point computed from *i*th equation:

$$x_i = \frac{x_{i-n} + x_{i-1} + x_{i+1} + x_{i+n}}{4}$$

or

$$x_{i-n} + x_{i-1} - 4x_i + x_{i+1} + x_{i+n} = 0$$

which is a linear equation with five unknowns (except those with boundary points).

In general form, the *i*th equation becomes:

$$a_{i,i-n}x_{i-n} + a_{i,i-1}x_{i-1} + a_{i,i}x_i + a_{i,i+1}x_{i+1} + a_{i,i+n}x_{i+n} = 0$$

where $a_{i,i} = -4$, and $a_{i,i-n} = a_{i,i-1} = a_{i,i+1} = a_{i,i+n} = 1$.