Performance

P: number of processors
N: input problem size
\(T_k(N): \) time to solve a problem of size N on k processors

\[
\text{Speedup}(P, N) = \frac{T_1(N)}{T_P(N)}
\]

Ideally, *linear speedup* is the goal.

Fixed N problem size

![Graph showing linear and typical speedup curves.

- Linear Speedup
- Typical Speedup Curve

Number of Processors, P

Speedup(P, N)
Sources of Overhead

1) **Excessive Sequential Code** - portions of the code are purely sequential. Example, Master process sending initial message to the slaves or collecting results

2) **Process Creation Time** - spawning of "slave" processes takes time

3) **Communication Delay** - Initialization of the Buffer, packing the data into the buffer, routing of the message through the interconnection network, network contention, etc.

4) **Synchronization Delay** - When processes synchronize, all processes must wait until the last one arrives, e.g., processes must wait to receive a message that has not arrived

5) **Memory Contention** (shared memory) - two processes accessing a global/shared data value (or even the same memory module)

6) **Load Imbalance** - processors have an uneven amount of work to perform, so some processors are sitting idle while others are computing
Superlinear Speedup

Characteristics:

1) Low overhead, little comm., etc.

2) Gain from splitting the work over many processors

Usually from a cache effect, i.e., the locality of (data) reference is split over the p processors so the hit ratio of the cache goes up over the sequential program.
**Graph Algorithms**

**Prim's Algorithm:** Given a undirected, connected, weighted graph $G(V, E)$ find its Minimum Spanning Tree, MST.

![Graph](image)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>*</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>*</td>
<td>3</td>
<td>*</td>
<td>0</td>
<td>5</td>
<td>*</td>
</tr>
<tr>
<td>E</td>
<td>*</td>
<td>*</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>*</td>
<td>2</td>
<td>*</td>
<td>5</td>
<td>0</td>
</tr>
</tbody>
</table>

Prim's Algorithm: $r$ is the initial edge

$MST = \{r\}$

dist [r] = 0;

for all $v$ in $(V - MST)$ do
  if edge($r$, $v$) exists then
    dist[$v$] = weight($r$, $v$)
  else
    dist[$v$] = *;
end for

while MST !$= V$ do
  find a vertex $u$ such that dist[$u$] = min{ dist[$v$] | for all $v$ in { V-MST } }
  MST = MST $\cup$ {u}
  for all $v$ in { V - MST } do
    dist[$v$] = min{dist[$v$],weight(u,v)}
  end while
Initially:

\[
\begin{array}{cccccc}
\text{dist} & 0 & 1 & 6 & \ast & \ast & 1 \\
\text{MST} & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccccc}
\text{dist} & 0 & 1 & 6 & \ast & \ast & 1 \\
\text{MST} & 1 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

How Do You Partition the Problem for a Parallel Solution?
<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>3</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>4</td>
<td>0</td>
<td>*</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>*</td>
<td>3</td>
<td>*</td>
<td>0</td>
<td>5</td>
<td>*</td>
</tr>
<tr>
<td>E</td>
<td>*</td>
<td>*</td>
<td>1</td>
<td>5</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>*</td>
<td>2</td>
<td>*</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>dist</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>MST</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

A   B   C   D   E   F

<table>
<thead>
<tr>
<th></th>
<th>P0</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>D</td>
<td>*</td>
<td>3</td>
<td>*</td>
</tr>
<tr>
<td>E</td>
<td>*</td>
<td>*</td>
<td>1</td>
</tr>
<tr>
<td>F</td>
<td>1</td>
<td>*</td>
<td>2</td>
</tr>
<tr>
<td>dist</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>MST</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Floyd's Algorithm: All-Pairs-Shortest Path

Algorithm: A sequence of \( n = |V| \) matrices \( D^{(0)}, D^{(1)}, \ldots, D^{(n)} \) are produced with \( D^{(n)} \) containing the shortest paths.

The \( d_{i,j}^{(k)} \) entry contains the shortest path between vertices \( i \) and \( j \) that pass through intermediate node 1 to \( (k-1) \).

To find: \( d_{i,j}^{(k)} \)

\[
\min\{d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)}\}
\]
Sequential Floyd's Algorithm:

\( D^{(0)} = \text{initial adjacency matrix} \)

\[
\text{for } k = 1 \text{ to } n \text{ do }
\]

\[
\text{for } i = 1 \text{ to } n \text{ do }
\]

\[
\text{for } j = 1 \text{ to } n \text{ do }
\]

\[
d_{i,j}^{(k)} = \min \left\{ d_{i,j}^{(k-1)}, d_{i,k}^{(k-1)} + d_{k,j}^{(k-1)} \right\}
\]

\[
\text{end for } j
\]

\[
\text{end for } i
\]

\[
\text{end for } k
\]
Consider a block checkerboard partitioning of the matrix

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$d_{k,j}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>$d_{i,k}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td>$d_{i,j}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Row K

Column K

When determining $D^{(k)}$ at processor $P_{m,n}$ what information does it need to be sent?
Outline of Parallel Floyd's Algorithm

Master:

Send blocks of matrix to each slave
Receive results from slaves

Slaves:

Receive initial block of matrix

for k = 0 to matrix_dimension do
    if I hold part of the k\textsuperscript{th} column then
        broadcast my part of the k\textsuperscript{th} column across my row of processors
    else
        receive part of the k\textsuperscript{th} column I need
    end if

if I hold part of the k\textsuperscript{th} row then
    broadcast my part of the k\textsuperscript{th} row across my column of processors
else
    receive part of the k\textsuperscript{th} row I need
end if

for local\_i = 0 to block\_dimension do
    for local\_j = 0 to block\_dimension do
        d\textsuperscript{(k)}[local\_i, local\_j] = min {...}
    end for local\_j
end for local\_i
end for k

Send resulting block to Master