Introduction to Parallel Computing

George Karypis
Dense Matrix Algorithms
Outline

- Focus on numerical algorithms involving dense matrices:
  - Matrix-Vector Multiplication
  - Matrix-Matrix Multiplication
  - Gaussian Elimination
- Decompositions & Scalability
### Table 4.1  Summary of communication times of various operations discussed in Sections 4.1–4.7 on a hypercube interconnection network. The message size for each operation is $m$ and the number of nodes is $p$.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Hypercube Time</th>
<th>B/W Requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-to-all broadcast,</td>
<td>$\min((t_s + t_w m) \log p, 2(t_s \log p + t_w m))$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>All-to-one reduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All-to-all broadcast,</td>
<td>$t_s \log p + t_w m(p - 1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>All-to-all reduction</td>
<td></td>
<td></td>
</tr>
<tr>
<td>All-reduce</td>
<td>$\min((t_s + t_w m) \log p, 2(t_s \log p + t_w m))$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>Scatter, Gather</td>
<td>$t_s \log p + t_w m(p - 1)$</td>
<td>$\Theta(1)$</td>
</tr>
<tr>
<td>All-to-all personalized</td>
<td>$(t_s + t_w m)(p - 1)$</td>
<td>$\Theta(p)$</td>
</tr>
<tr>
<td>Circular shift</td>
<td>$t_s + t_w m$</td>
<td>$\Theta(p)$</td>
</tr>
</tbody>
</table>
Matrix-Vector Multiplication

- Compute: $y = Ax$
  - $y, x$ are $n \times 1$ vectors
  - $A$ is an $n \times n$ dense matrix

- Serial complexity: $W = O(n^2)$.

- We will consider:
  - 1D & 2D partitioning.

```python
1. procedure MAT_VECT (A, x, y)
2. begin
3.   for i := 0 to n - 1 do
4.     begin
5.       y[i] := 0;
6.       for j := 0 to n - 1 do
7.         y[i] := y[i] + A[i, j] \times x[j];
8.     endfor;
9.   end MAT_VECT
```
Row-wise 1D Partitioning

How do we perform the operation?
Row-wise 1D Partitioning

Each processor needs to have the entire $x$ vector.

Analysis?

$$T_{P} = \frac{n^2}{p} + t_{s} \log p + t_{w} n.$$  

$$T_{o} = t_{s} p \log p + t_{w} n p.$$  

$$W = \Theta(p^2).$$
Block 2D Partitioning

(a) Initial data distribution and communication steps to align the vector along the diagonal

(d) Final distribution of the result vector

How do we perform the operation?
Block 2D Partitioning

Each processor needs to have the portion of the $x$ vector that corresponds to the set of columns that it stores.

(b) One-to-all broadcast of portions of the vector along process columns

(c) All-to-one reduction of partial results

Analysis?

$$T_P = \frac{n^2}{p} + \frac{t_s + t_{wn}/\sqrt{p}}{\sqrt{p}} + \frac{(t_s + t_{wn}/\sqrt{p}) \log(\sqrt{p})}{\sqrt{p}} + \frac{(t_s + t_{wn}/\sqrt{p}) \log(\sqrt{p})}{\sqrt{p}}$$

$$T_o = t_s p \log p + t_{wn} n \sqrt{p} \log p.$$ 

$$W = \Theta(p \log^2 p).$$
1D vs 2D Formulation

- Which one is better?
Matrix-Matrix Multiplication

- Compute: $C = AB$
  - $A, B, \& C$ are $n \times n$ dense matrices.
- Serial complexity: $W = O(n^3)$.
- We will consider:
  - 2D & 3D partitioning.

1. procedure MATMULT $(A, B, C)$
2. begin
3. for $i := 0$ to $n - 1$ do
4.   for $j := 0$ to $n - 1$ do
5.     begin
6.       $C[i, j] := 0;$
7.     endfor;
8.     for $k := 0$ to $n - 1$ do
10. end MATMULT
Simple 2D Algorithm

- Processors are arranged in a logical \( \sqrt{p} \times \sqrt{p} \) 2D topology.
- Each processor gets a block of \( \frac{n}{\sqrt{p}} \times \frac{n}{\sqrt{p}} \) block of A, B, & C.
- It is responsible for computing the entries of C that it has been assigned to.

Analysis?

\[
T_p = \frac{n^3}{p} + t_s \log p + 2t_w \frac{n^2}{\sqrt{p}}. \\
W = \Theta(p^{3/2})
\]
Cannon’s Algorithm

- Memory efficient variant of the simple algorithm.
- Key idea:
  - Replace traditional loop:
    \[
    C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,k} \times B_{k,j}
    \]
  - With the following loop:
    \[
    C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,(i+j+k)\%\sqrt{p}} \times B_{(i+j+k)\%\sqrt{p},j}
    \]
- During each step, processors operate on different blocks of \(A\) and \(B\).

\[
T_P = \frac{n^3}{p} + 2\sqrt{p}t_s + 2t_w \frac{n^2}{\sqrt{p}}.
\]

Figure 8.3 The communication steps in Cannon’s algorithm on 16 processes.
Can we do better?

- Can we use more than $O(n^2)$ processors?
- So far the task corresponded to the dot-product of two vectors
  - i.e., $C_{i,j} = A_{i,*} \cdot B_{*j}$
- How about performing this dot-product in parallel?
- What is the maximum concurrency that we can extract?
3D Algorithm—DNS Algorithm

Partitioning the intermediate data

![Diagram](image)

**Figure 3.14** Multiplication of matrices $A$ and $B$ with partitioning of the three-dimensional intermediate matrix $D$.

**Stage I**

\[
\begin{pmatrix}
A_{1,1} & A_{1,2} \\
A_{2,1} & A_{2,2}
\end{pmatrix}
\times
\begin{pmatrix}
B_{1,1} & B_{1,2} \\
B_{2,1} & B_{2,2}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
D_{1,1,1} & D_{1,1,2} \\
D_{1,2,1} & D_{1,2,2}
\end{pmatrix}
\]

**Stage II**

\[
\begin{pmatrix}
D_{1,1,1} & D_{1,1,2} \\
D_{1,2,1} & D_{1,2,2}
\end{pmatrix}
+ \begin{pmatrix}
D_{2,1,1} & D_{2,1,2} \\
D_{2,2,1} & D_{2,2,2}
\end{pmatrix}
\rightarrow
\begin{pmatrix}
C_{1,1} & C_{1,2} \\
C_{2,1} & C_{2,2}
\end{pmatrix}
\]

A decomposition induced by a partitioning of $D$:

- Task 01: $D_{1,1,1} = A_{1,1}B_{1,1}$
- Task 02: $D_{2,1,1} = A_{1,2}B_{2,1}$
- Task 03: $D_{1,1,2} = A_{1,1}B_{1,2}$
- Task 04: $D_{2,1,2} = A_{1,2}B_{2,2}$
- Task 05: $D_{1,2,1} = A_{2,1}B_{1,1}$
- Task 06: $D_{2,2,1} = A_{2,2}B_{2,1}$
- Task 07: $D_{1,2,2} = A_{2,1}B_{2,2}$
- Task 08: $D_{2,2,2} = A_{2,2}B_{2,2}$
- Task 09: $C_{1,1} = D_{1,1,1} + D_{1,1,2}$
- Task 10: $C_{1,2} = D_{1,2,1} + D_{2,1,2}$
- Task 11: $C_{2,1} = D_{1,1,2} + D_{2,2,1}$
- Task 12: $C_{2,2} = D_{1,2,2} + D_{2,2,2}$

**Figure 3.15** A decomposition of matrix multiplication based on partitioning the intermediate three-dimensional matrix.

**Figure 3.16** The task-dependency graph of the decomposition shown in Figure 3.15.
3D Algorithm—DNS Algorithm

\[ q = p^{1/3} \]

\[ T_P \approx \left( \frac{n}{q} \right)^3 + 3t_s \log q + 3t_w \left( \frac{n}{q} \right)^2 \log q \]

\[ T_P = \frac{n^3}{p} + t_s \log p + t_w \frac{n^2}{p^{2/3}} \log p. \]

\[ W = O(p(\log p)^3) \]

Figure 8.4 The communication steps in the DNS algorithm while multiplying 4 × 4 matrices \( A \) and \( B \) on 64 processes. The shaded processes in part (c) store elements of the first row of \( A \) and the shaded processes in part (d) store elements of the first column of \( B \).
Which one is better?
Gaussian Elimination

- Solve $Ax=b$
  - $A$ is an $nxn$ dense matrix.
  - $x$ and $b$ are dense vectors

- Serial complexity: $W = O(n^3)$.

- There are two key steps in each iteration:
  - Division step
  - Rank-1 update

- We will consider:
  - 1D & 2D partitioning, and introduce the notion of pipelining.

```plaintext
procedure GAUSSIAN_ELIMINATION (A, b, y)
begin
  for $k := 0$ to $n-1$ do /* Outer loop */
  begin
    for $j := k+1$ to $n-1$ do
    begin
      $v[k] := v[k] / A[k, k];$
    endfor
    $A[k, k] := 1;$
    for $i := k+1$ to $n-1$ do
    begin
      for $j := k+1$ to $n-1$ do
      begin
        $b[i] := b[i] - A[i, k] \times y[k];$
      endfor
    endfor
    $A[i, k] := 0;$ /* Line 9 */
  endfor
  /* Line 3 */
end GAUSSIAN_ELIMINATION
```

Algorithm 8.4 A serial Gaussian elimination algorithm that converts the system of linear equations $Ax = b$ to a unit upper-triangular system $Ux = y$. The matrix $U$ occupies the upper-triangular locations of $A$. This algorithm assumes that $A[k, k] \neq 0$ when it is used as a divisor on lines 6 and 8.

![Gaussian Elimination Diagram](image)

Figure 8.5 A typical computation in Gaussian elimination.
1D Partitioning

- Assign \( n/p \) rows of \( A \) to each processor.

- During the \( j^{th} \) iteration:
  - Divide operation is performed by the processor who stores row \( i \).
  - Result is broadcasted to the rest of the processors.
  - Each processor performs the rank-1 update for its local rows.

- Analysis?

\[
T_P =\frac{3}{2}n(n - 1) + t_s n \log n + \frac{1}{2}t_w n(n - 1) \log n.
\]

(one element per processor)

```
<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>P_0</td>
<td>1</td>
<td>(0.1)</td>
<td>(0.2)</td>
<td>(0.3)</td>
<td>(0.4)</td>
<td>(0.5)</td>
<td>(0.6)</td>
</tr>
<tr>
<td>P_1</td>
<td>0</td>
<td>1</td>
<td>(1.2)</td>
<td>(1.3)</td>
<td>(1.4)</td>
<td>(1.5)</td>
<td>(1.6)</td>
</tr>
<tr>
<td>P_2</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(2.3)</td>
<td>(2.4)</td>
<td>(2.5)</td>
<td>(2.6)</td>
</tr>
<tr>
<td>P_3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(3.3)</td>
<td>(3.4)</td>
<td>(3.5)</td>
<td>(3.6)</td>
</tr>
<tr>
<td>P_4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(4.3)</td>
<td>(4.4)</td>
<td>(4.5)</td>
<td>(4.6)</td>
</tr>
<tr>
<td>P_5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(5.3)</td>
<td>(5.4)</td>
<td>(5.5)</td>
<td>(5.6)</td>
</tr>
<tr>
<td>P_6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(6.3)</td>
<td>(6.4)</td>
<td>(6.5)</td>
<td>(6.6)</td>
</tr>
<tr>
<td>P_7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(7.3)</td>
<td>(7.4)</td>
<td>(7.5)</td>
<td>(7.6)</td>
</tr>
</tbody>
</table>
```

(a) Computation:
(i) \( A[k,j] := A[k,j] / A[k,k] \) for \( k < j < i \)
(ii) \( A[k,k] := 1 \)

(b) Communication:
One-to-all broadcast of row \( A[k,\ast] \)

(c) Computation:
for \( k < i < n \) and \( k < j < n \)
(ii) \( A[i,k] := 0 \) for \( k < i < n \)

Figure 8.6 Gaussian elimination steps during the iteration corresponding to \( k = 3 \) for an \( 8 \times 8 \) matrix partitioned rowwise among eight processes.
1D Pipelined Formulation

- Existing Algorithm:
  Next iteration starts only when the previous iteration has finished.

- Key Idea:
  The next iteration can start as soon as the rank-1 update involving the next row has finished.

  - Essentially multiple iterations are perform simultaneously!
Cost-optimal with $n$ processors

<table>
<thead>
<tr>
<th>(a) Iteration $k = 0$ starts</th>
<th>(b)</th>
<th>(c)</th>
<th>(d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
</tr>
<tr>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
</tr>
<tr>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
</tr>
<tr>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
</tr>
<tr>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(e) Iteration $k = 1$ starts</th>
<th>(f)</th>
<th>(g) Iteration $k = 0$ ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
</tr>
<tr>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
</tr>
<tr>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
</tr>
<tr>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
</tr>
<tr>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(h) Iteration $k = 0$ ends</th>
<th>(i) Iteration $k = 2$ starts</th>
<th>(j) Iteration $k = 1$ ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
</tr>
<tr>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
</tr>
<tr>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
</tr>
<tr>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
</tr>
<tr>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
</tr>
<tr>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(k) Iteration $k = 1$ ends</th>
<th>(l) Iteration $k = 3$ starts</th>
<th>(m) Iteration $k = 3$ ends</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
</tr>
<tr>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
</tr>
<tr>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
</tr>
<tr>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
</tr>
<tr>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
</tr>
<tr>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(n) Iteration $k = 3$ ends</th>
<th>(o) Iteration $k = 3$ ends</th>
<th>(p) Iteration $k = 4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
<td>1 (0.1) (0.2) (0.3) (0.4)</td>
</tr>
<tr>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
<td>0 (1.1) (1.2) (1.3) (1.4)</td>
</tr>
<tr>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
<td>0 (2.1) (2.2) (2.3) (2.4)</td>
</tr>
<tr>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(2.0) (2.1) (2.2) (2.3) (2.4)</td>
</tr>
<tr>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td>(3.0) (3.1) (3.2) (3.3) (3.4)</td>
</tr>
<tr>
<td>(4.0) (4.1) (4.2) (4.3) (4.4)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Communication for $k = 0, 3$
- Communication for $k = 1$
- Communication for $k = 2$
- Communication for $k = 0, 3$
- Communication for $k = 1, 4$
- Communication for $k = 2$
1D Partitioning

- Is the block mapping a good idea?

![Diagram](image)

(a) Block 1-D mapping

<table>
<thead>
<tr>
<th>P₀</th>
<th>1</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>(1,2)</td>
<td>(1,3)</td>
<td>(1,4)</td>
<td>(1,5)</td>
<td>(1,6)</td>
<td>(1,7)</td>
<td></td>
</tr>
<tr>
<td>P₁</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(2,3)</td>
<td>(2,4)</td>
<td>(2,5)</td>
<td>(2,6)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(3,3)</td>
<td>(3,4)</td>
<td>(3,5)</td>
<td>(3,6)</td>
<td>(3,7)</td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(4,3)</td>
<td>(4,4)</td>
<td>(4,5)</td>
<td>(4,6)</td>
<td>(4,7)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(5,3)</td>
<td>(5,4)</td>
<td>(5,5)</td>
<td>(5,6)</td>
<td>(5,7)</td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(6,3)</td>
<td>(6,4)</td>
<td>(6,5)</td>
<td>(6,6)</td>
<td>(6,7)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(7,3)</td>
<td>(7,4)</td>
<td>(7,5)</td>
<td>(7,6)</td>
<td>(7,7)</td>
<td></td>
</tr>
</tbody>
</table>

(b) Cyclic 1-D mapping

<table>
<thead>
<tr>
<th>P₀</th>
<th>1</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(4.3)</td>
<td>(4.4)</td>
<td>(4.5)</td>
<td>(4.6)</td>
<td>(4.7)</td>
</tr>
<tr>
<td>P₁</td>
<td>0</td>
<td>1</td>
<td>(1.2)</td>
<td>(1.3)</td>
<td>(1.4)</td>
<td>(1.5)</td>
<td>(1.6)</td>
<td>(1.7)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(5.3)</td>
<td>(5.4)</td>
<td>(5.5)</td>
<td>(5.6)</td>
<td>(5.7)</td>
<td></td>
</tr>
<tr>
<td>P₂</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>(2.3)</td>
<td>(2.4)</td>
<td>(2.5)</td>
<td>(2.6)</td>
<td>(2.7)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(6.3)</td>
<td>(6.4)</td>
<td>(6.5)</td>
<td>(6.6)</td>
<td>(6.7)</td>
<td></td>
</tr>
<tr>
<td>P₃</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>(7.3)</td>
<td>(7.4)</td>
<td>(7.5)</td>
<td>(7.6)</td>
<td>(7.7)</td>
</tr>
</tbody>
</table>

**Figure 8.9** Computation load on different processes in block and cyclic 1-D partitioning of an $8 \times 8$ matrix on four processes during the Gaussian elimination iteration corresponding to $k = 3$. 
2D Mapping

- Each processor gets a 2D block of the matrix.

Steps:
- Broadcast of the “active” column along the rows.
- Divide step in parallel by the processors who own portions of the row.
- Broadcast along the columns.
- Rank-1 update.

Analysis?

Figure 8.10 Various steps in the Gaussian elimination iteration corresponding to $k = 3$ for an $8 \times 8$ matrix on 64 processes arranged in a logical two-dimensional mesh.
2D Pipelined

Cost-optimal with $n^2$ processors