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

Review

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.

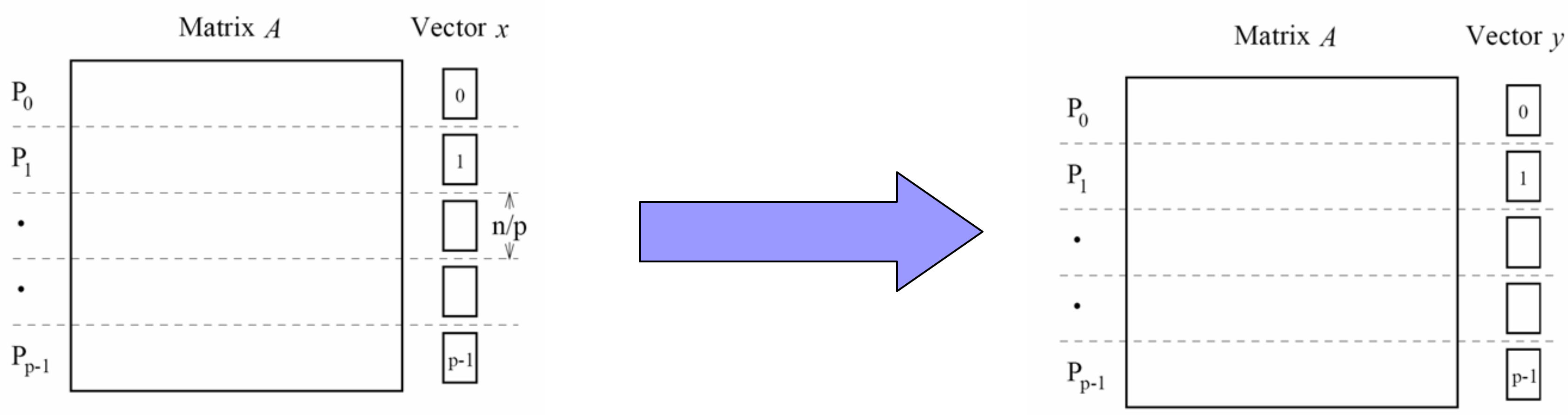
Operation	Hypercube Time	B/W Requirement
One-to-all broadcast, All-to-one reduction	$\min((t_s + t_w m)\log p, 2(t_s \log p + t_w m))$	$\Theta(1)$
All-to-all broadcast, All-to-all reduction	$t_s \log p + t_w m(p-1)$	$\Theta(1)$
All-reduce	$\min((t_s + t_w m)\log p, 2(t_s \log p + t_w m))$	$\Theta(1)$
Scatter, Gather	$t_s \log p + t_w m(p-1)$	$\Theta(1)$
All-to-all personalized	$(t_s + t_w m)(p-1)$	$\Theta(p)$
Circular shift	$t_s + t_w m$	$\Theta(p)$

Matrix-Vector Multiplication

- Compute: $y = Ax$
  - y, x are nx1 vectors
  - A is an *nxn dense matrix*
- Serial complexity: $W = O(n^2)$.
- We will consider:
  - 1D & 2D partitioning.

```
procedure MAT_VECT (A, x, y)
begin
    for i := 0 to n - 1 do
    begin
        y[i] := 0;
        for j := 0 to n - 1 do
            y[i] := y[i] + A[i, j] × x[j];
    endfor;
end MAT_VECT
```

Row-wise 1D Partitioning



(a) Initial partitioning of the matrix and the starting vector x

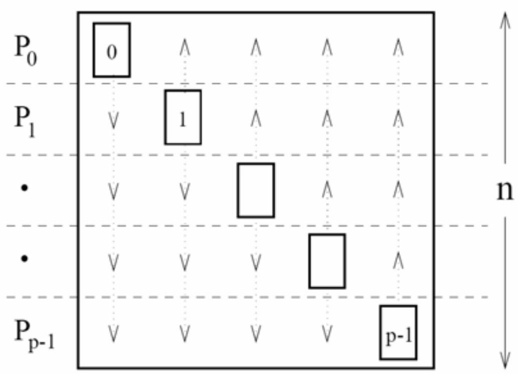
(d) Final distribution of the matrix and the result vector y

How do we perform the operation?

Row-wise 1D Partitioning

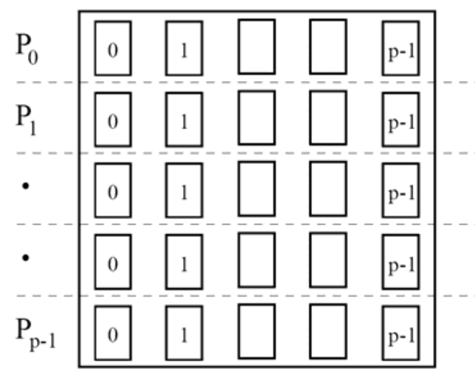
Each processor needs to have the entire *x* vector.

All-to-all broadcast



(b) Distribution of the full vector among all the processes by all-to-all broadcast

Local computations



(c) Entire vector distributed to each process after the broadcast

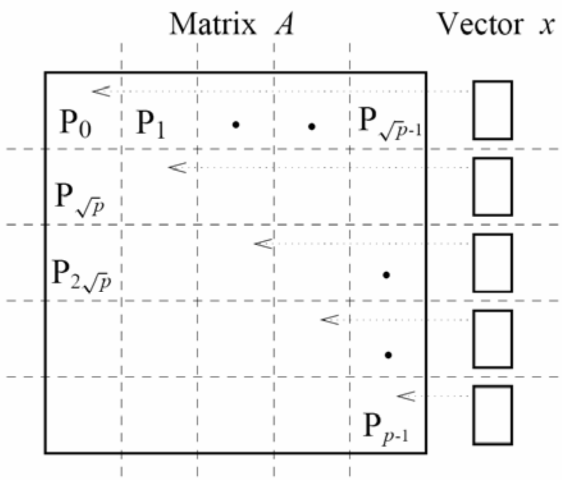
Analysis?

$$\boxed{T_P = \frac{n^2}{p} + t_s \log p + t_w n.}$$

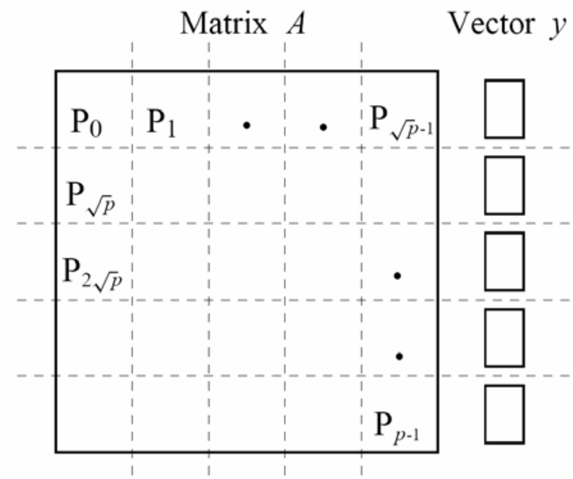
$$\boxed{T_o = t_s p \log p + t_w n p.}$$

$$W = \Theta(p^2)$$

Block 2D Partitioning



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

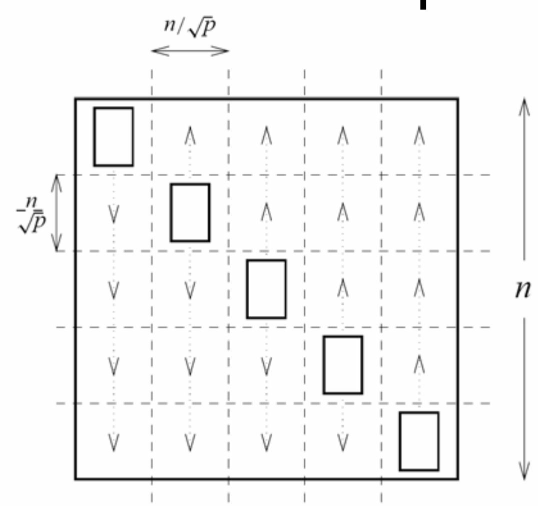


(d) Final distribution of the result vector

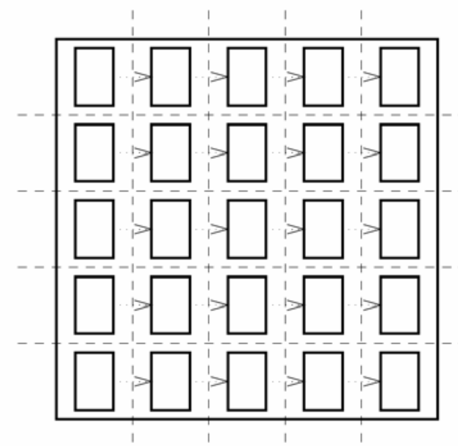
How do we perform the operation?

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?

$$\begin{aligned} T_P &= \overbrace{n^2/p}^{\text{computation}} + \overbrace{t_s + t_w n/\sqrt{p}}^{\text{aligning the vector}} + \\ & \overbrace{(t_s + t_w n/\sqrt{p})\log(\sqrt{p})}^{\text{columnwise one-to-all broadcast}} + \overbrace{(t_s + t_w n/\sqrt{p})\log(\sqrt{p})}^{\text{all-to-one reduction}} \\ &\approx \frac{n^2}{p} + t_s \log p + t_w \frac{n}{\sqrt{p}} \log p \end{aligned}$$

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

$$W = \Theta(p \log^2 p).$$

1D vs 2D Formulation

- Which one is better?

Matrix-Matrix Multiplication

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

```
procedure MAT_MULT (A, B, C)
begin
    for i := 0 to n − 1 do
        for j := 0 to n − 1 do
            begin
                C[i, j] := 0;
                for k := 0 to n − 1 do
                    C[i, j] := C[i, j] + A[i, k] × B[k, j];
            endfor;
end MAT_MULT
```

Simple 2D Algorithm

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

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

$$W = \Theta(p^{3/2})$$

How about the memory complexity?

Cannon's Algorithm

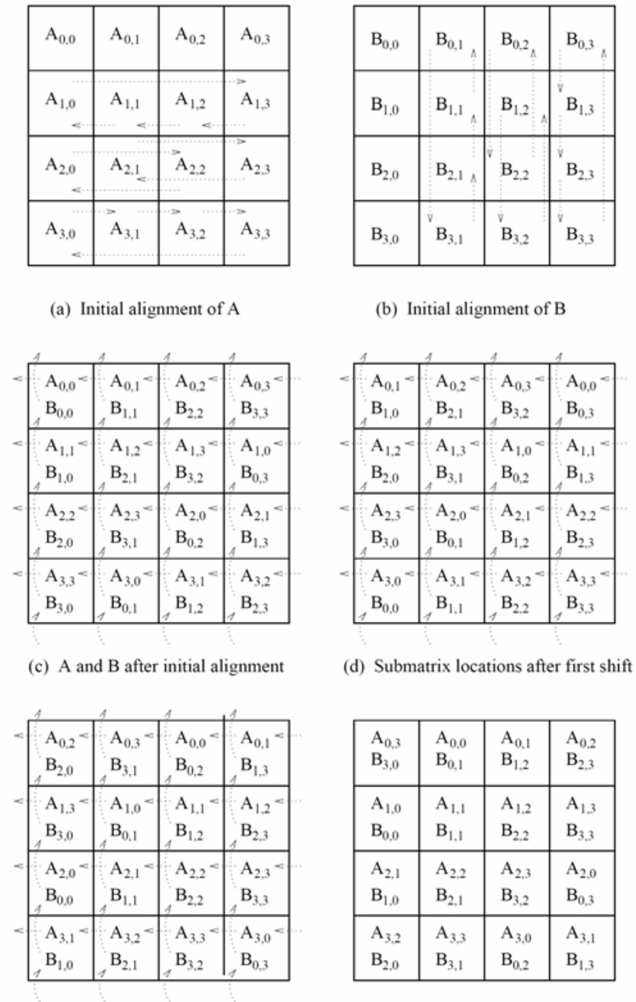
- Memory efficient variant of the simple algorithm.
- Key idea:
  - Replace traditional loop:

$$C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,k} * B_{k,j}$$

  - With the following loop:

$$C_{i,j} = \sum_{k=0}^{\sqrt{p}-1} A_{i,(i+j+k)\%\sqrt{p}} * B_{(i+j+k)\%\sqrt{p},j}$$

- During each step, processors operate on different blocks of *A* and *B*.



(a) Initial alignment of A (b) Initial alignment of B

(c) A and B after initial alignment (d) Submatrix locations after first shift

(e) Submatrix locations after second shift (f) Submatrix locations after third shift

Figure 8.3 The communication steps in Cannon's algorithm on 16 processes.

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

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., $C_{i,j} = A_{i,*} \, . \, B_{*,j}$
- 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

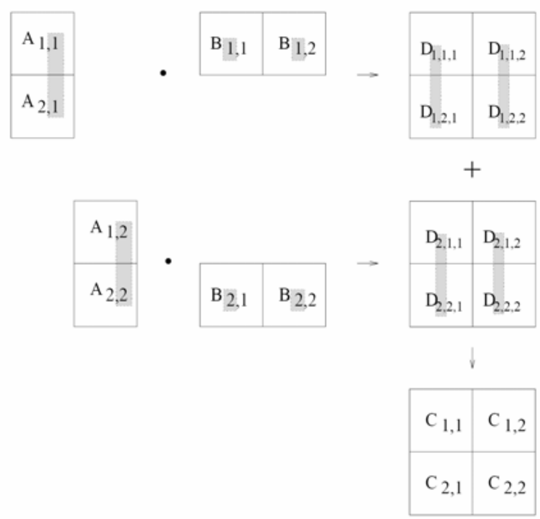


Figure 3.14 Multiplication of matrices A and B with partitioning of the three-dimensional intermediate matrix D.

Stage I

$$\begin{pmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{pmatrix} \cdot \begin{pmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{pmatrix} \rightarrow \left(\begin{matrix} \begin{pmatrix} D_{1,1,1} & D_{1,1,2} \\ D_{1,2,2} & D_{1,2,2} \end{pmatrix} \\ \begin{pmatrix} D_{2,1,1} & D_{2,1,2} \\ D_{2,2,2} & D_{2,2,2} \end{pmatrix} \end{matrix} \right)$$

Stage II

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

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: $D_{1,2,2} = A_{2,1} B_{1,2}$
Task 08: $D_{2,2,2} = A_{2,2} B_{2,2}$
Task 09: $C_{1,1} = D_{1,1,1} + D_{2,1,1}$
Task 10: $C_{1,2} = D_{1,1,2} + D_{2,1,2}$
Task 11: $C_{2,1} = D_{1,2,1} + D_{2,2,1}$
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.

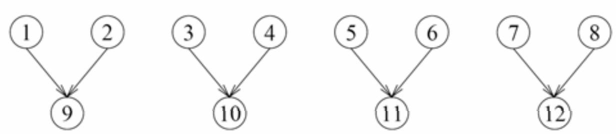


Figure 3.16 The task-dependency graph of the decomposition shown in Figure 3.15.

3D Algorithm—DNS Algorithm

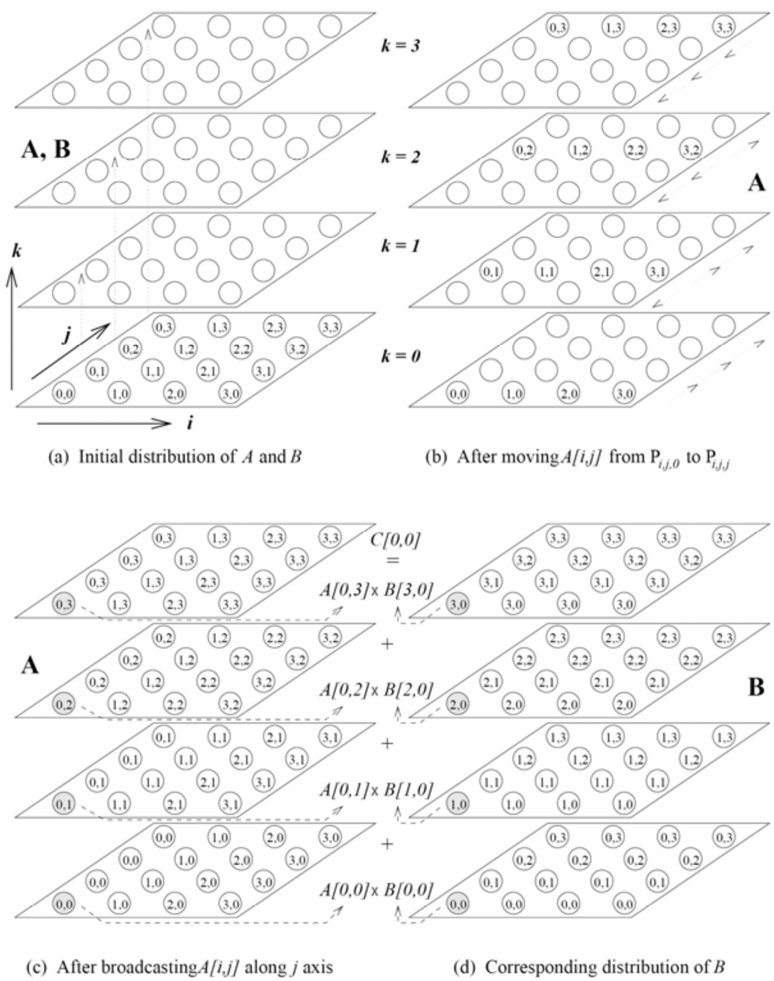


Figure 8.4 The communication steps in the DNS algorithm while multiplying 4 × 4 matrices A and B on 64 processes. The shaded processes in part (c) store elements of the first row of A and the shaded processes in part (d) store elements of the first column of B.

$$q = p^{1/3}$$

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

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

$$W = \Theta(p(\log p)^3)$$

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.

```
1.  procedure GAUSSIAN_ELIMINATION (A, b, y)
2.  begin
3.     for k := 0 to n − 1 do                /* Outer loop */
4.     begin
5.        for j := k + 1 to n − 1 do
6.           A[k, j] := A[k, j]/A[k, k];   /* Division step */
7.        y[k] := b[k]/A[k, k];
8.        A[k, k] := 1;
9.        for i := k + 1 to n − 1 do
10.       begin
11.          for j := k + 1 to n − 1 do
12.             A[i, j] := A[i, j] − A[i, k] × A[k, j]; /* Elimination step */
13.          b[i] := b[i] − A[i, k] × y[k];
14.          A[i, k] := 0;
15.       endfor;           /* Line 9 */
16.    endfor;              /* Line 3 */
17. 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

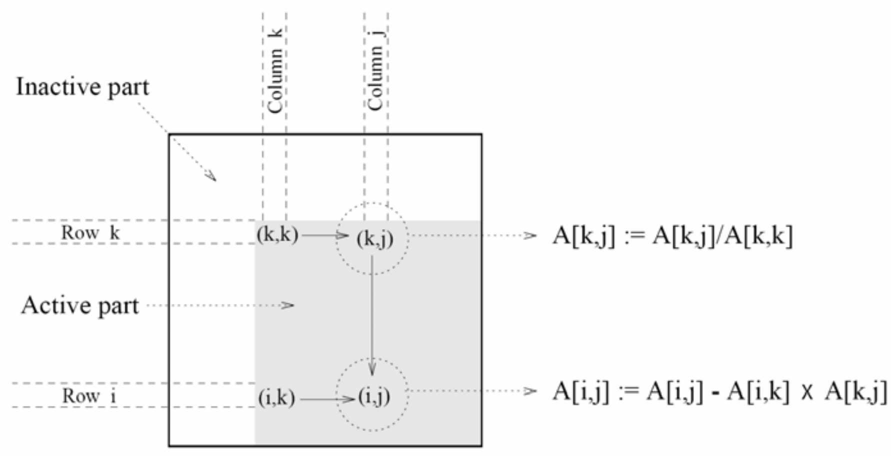


Figure 8.5 A typical computation in Gaussian elimination.

1D Partitioning

- Assign *n/p* rows of *A* to each processor.
- During the i^{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)

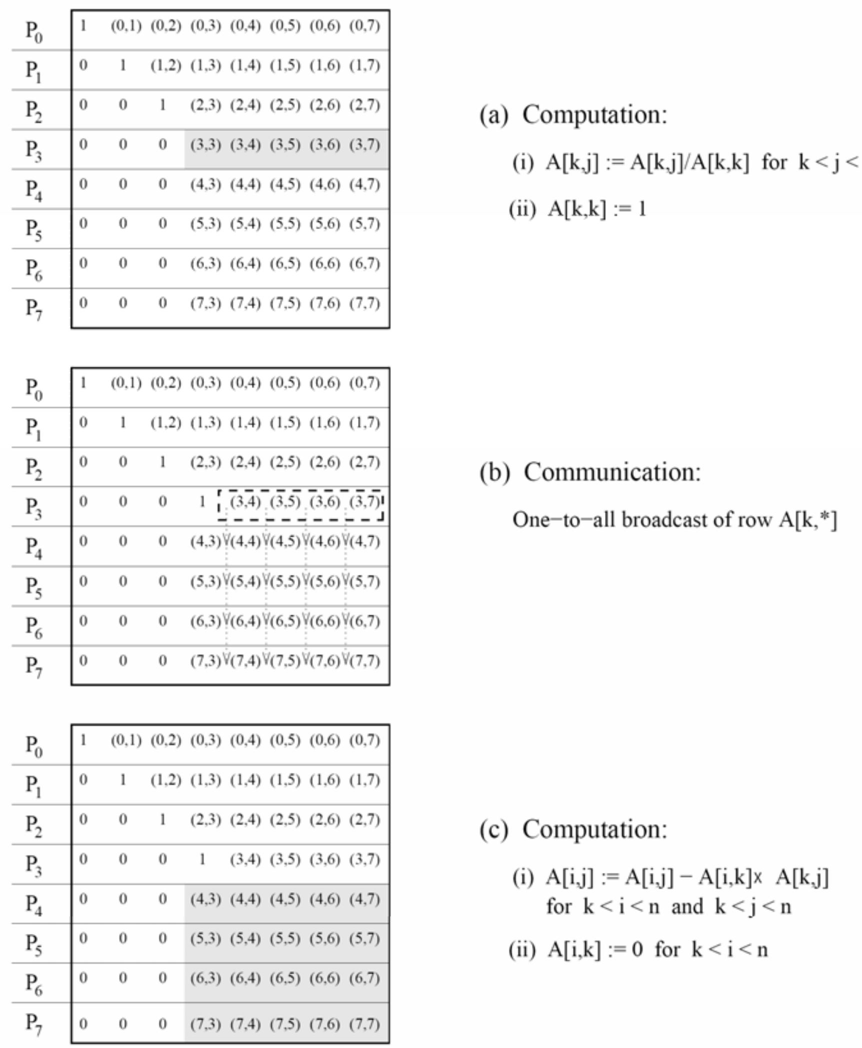
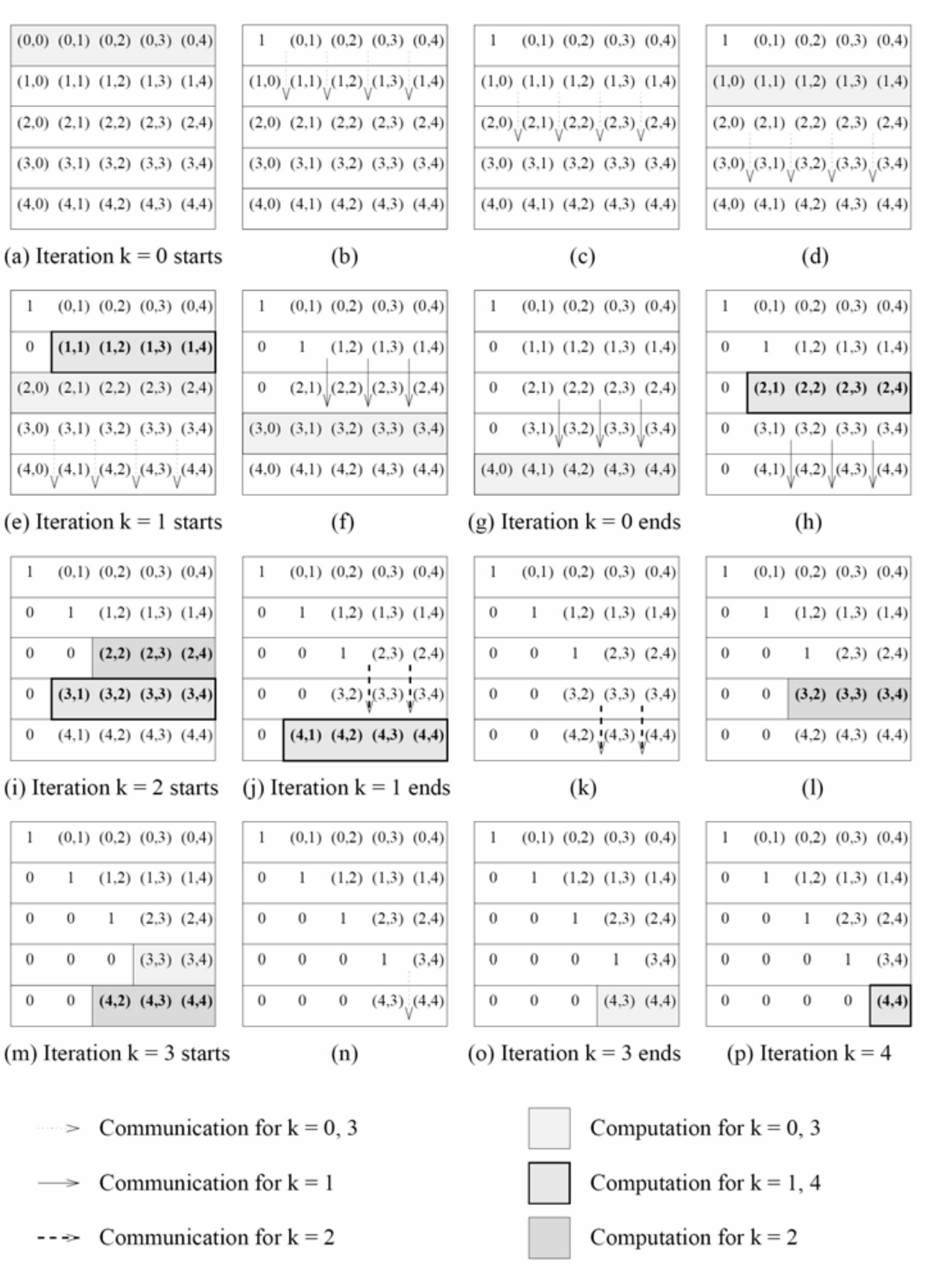


Figure 8.6 Gaussian elimination steps during the iteration corresponding to $k = 3$ for an 8×8 matrix partitioned rowwise among eight processes.

1D Pipelined Formulation

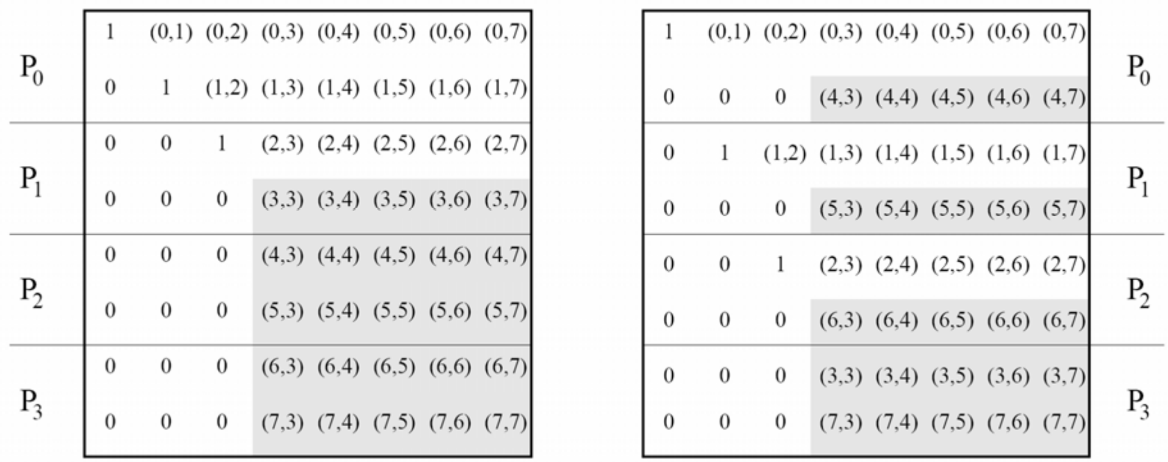
- Existing Algorithm:
  Next iteration starts only when the previous iteration has finished.
- Key Idea:
  The next iteration can start as soon as the rank-1 update involving the next row has finished.
  - Essentially multiple iterations are perform simultaneously!



Cost-optimal with *n* processors

1D Partitioning

- Is the block mapping a good idea?



(a) Block 1-D mapping (b) Cyclic 1-D mapping

Figure 8.9 Computation load on different processes in block and cyclic 1-D partitioning of an 8 × 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?

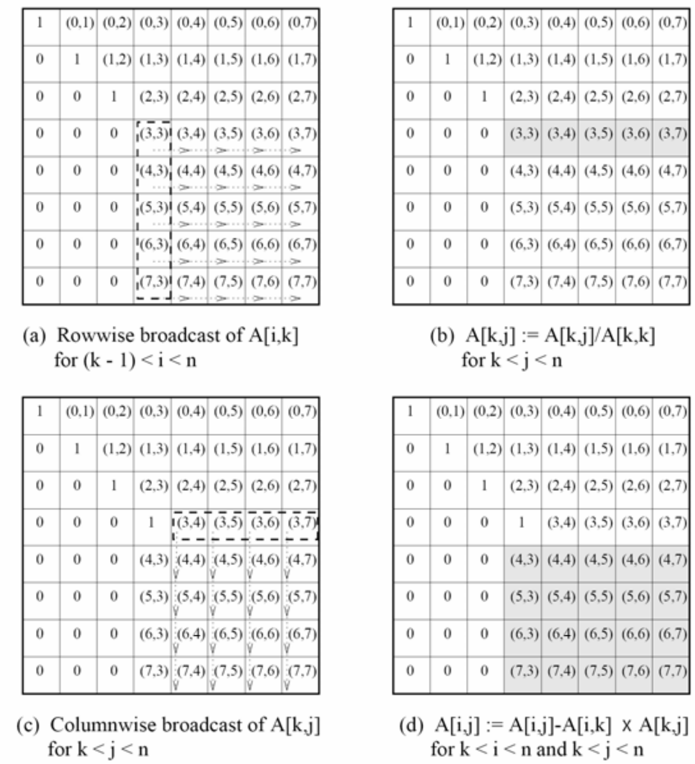
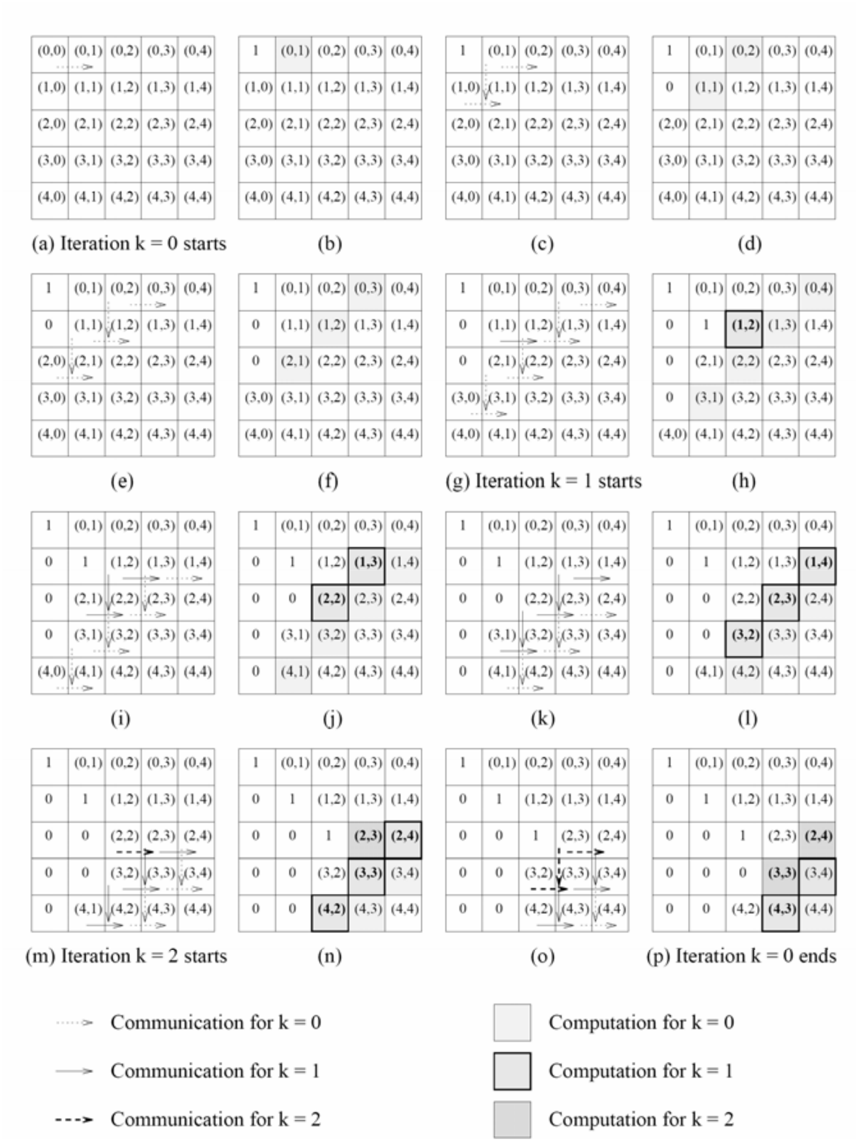


Figure 8.10 Various steps in the Gaussian elimination iteration corresponding to $k = 3$ for an 8×8 matrix on 64 processes arranged in a logical two-dimensional mesh.

2D Pipelined



Cost-optimal with
n^2 processors