Electrical Engineering and Computer Science

EECS 358 - INTRODUCTION TO PARALLEL COMPUTING

Lecture 15

Dense Matrix Algorithms

Outline

- Design of parallel algorithms
- Dense matrix representations
- Matrix transpose algorithms
- Matrix vector multiplication
- Matrix matrix multiplication
- Linear system of equation solvers
- READING: Chapter 5, V. Kumar, "Introduction to Parallel Computing"

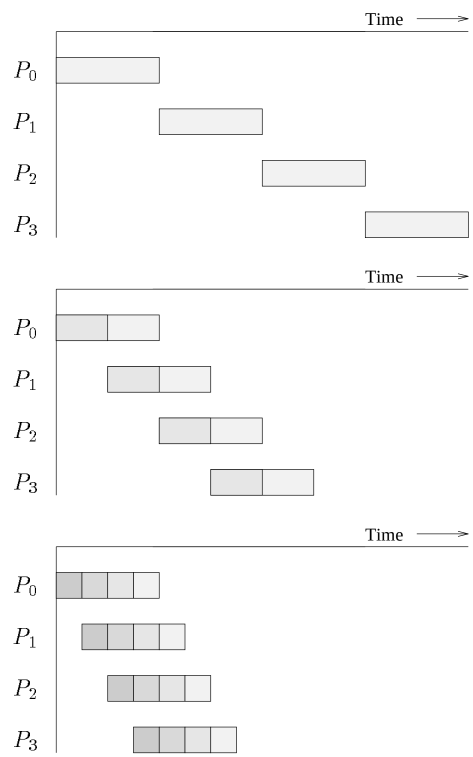
Design of Parallel Algorithms

- Assume performance model of cost of communication
  - Startup time for messages (t_s)
  - Per-hop time (t_h)
  - Per-word transfer time (t_w)
- Assume performance model of computation
  - Cost of multiplication or addition or any operation (t_c)

Store and Forward vs Cut Through Routing

- Cost of store and forward routing of message of size m via l hops, $t_{comm} = t_s + (m.t_w + t_h)l$
- In current parallel computers, t_h is small, hence approximate as $t_{comm} = t_s + m.t_w.l$
- Cost of cut through routing of message of size m via l hops, $t_{comm} = t_s + l.t_h + m.t_w$

Store and Forward vs Cut Through Routing



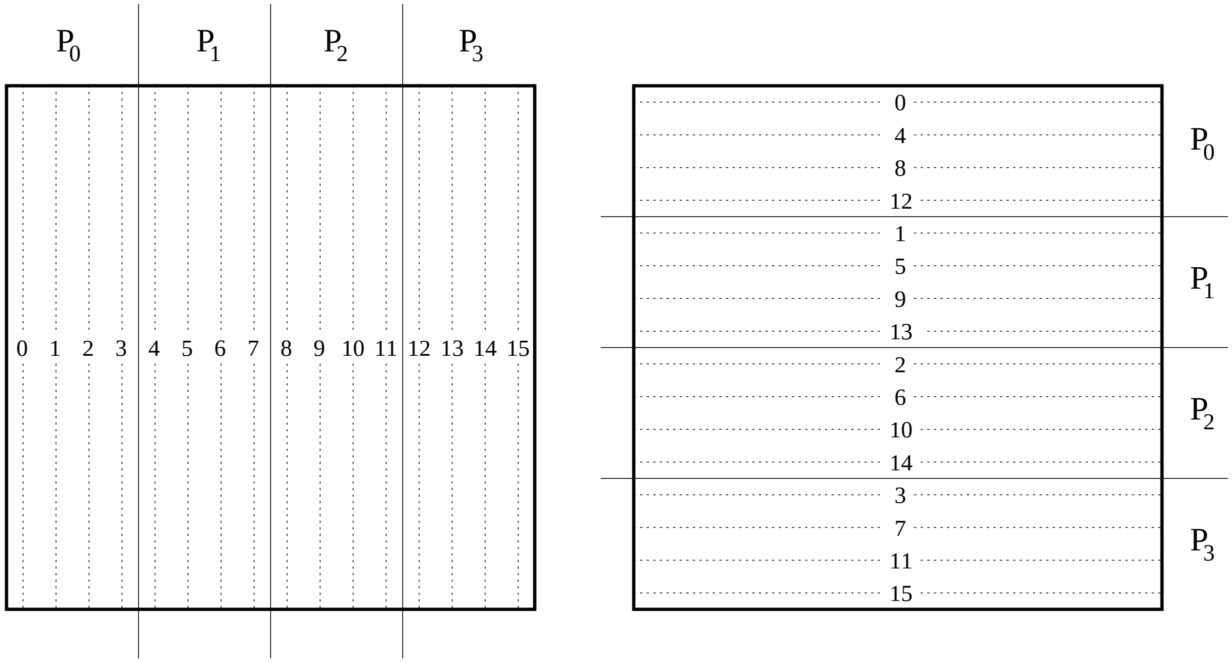
(a) A single message sent over a store-and-forward network

(b) The same message broken into two parts and sent over the network.

(c) The same message broken into four parts and sent over the network.

Dense Matrix Mappings

- Striped partitioning (one dimensional, row or column, block or cyclic)



(a) Columwise block striping (b) Rowwise cyclic striping

Dense Matrix Mappings-Continued

- Checkerboard partitioning (two dimensional, blocked or cyclic)

(0,0) (0,1) P_0 (1,0) (1,1)	(0,2) (0,3) P_1 (1,2) (1,3)	(0,4) (0,5) P_2 (1,4) (1,5)	(0,6) (0,7) P_3 (1,6) (1,7)
(2,0) (2,1) P_4 (3,0) (3,1)	(2,2) (2,3) P_5 (3,2) (3,3)	(2,4) (2,5) P_6 (3,4) (3,5)	(2,6) (2,7) P_7 (3,6) (3,7)
(4,0) (4,1) P_8 (5,0) (5,1)	(4,2) (4,3) P_9 (5,2) (5,3)	(4,4) (4,5) P_{10} (5,4) (5,5)	(4,6) (4,7) P_{11} (5,6) (5,7)
(6,0) (6,1) P_{12} (7,0) (7,1)	(6,2) (6,3) P_{13} (7,2) (7,3)	(6,4) (6,5) P_{14} (7,4) (7,5)	(6,6) (6,7) P_{15} (7,6) (7,7)

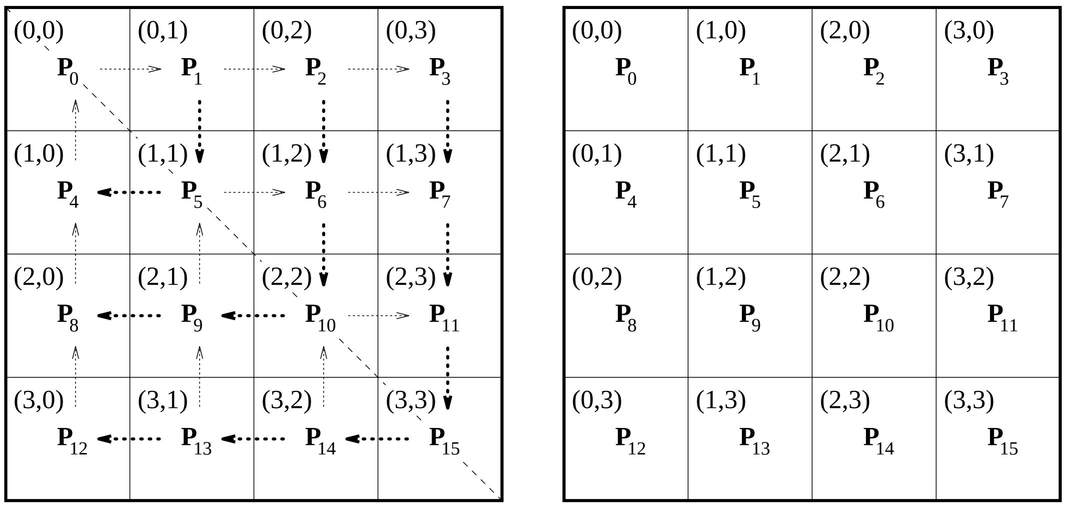
(a) Block-checkerboard partitioning

(0,0) (0,4) P_0 (4,0) (4,4)	(0,1) (0,5) P_1 (4,1) (4,5)	(0,2) (0,6) P_2 (4,2) (4,6)	(0,3) (0,7) P_3 (4,3) (4,7)
(1,0) (1,4) P_4 (5,0) (5,4)	(1,1) (1,5) P_5 (5,1) (5,5)	(1,2) (1,6) P_6 (5,2) (5,6)	(1,3) (1,7) P_7 (5,3) (5,7)
(2,0) (2,4) P_8 (6,0) (6,4)	(2,1) (2,5) P_9 (6,1) (6,5)	(2,2) (2,6) P_{10} (6,2) (6,6)	(2,3) (2,7) P_{11} (6,3) (6,7)
(3,0) (3,4) P_{12} (7,0) (7,4)	(3,1) (3,5) P_{13} (7,1) (7,5)	(3,2) (3,6) P_{14} (7,2) (7,6)	(3,3) (3,7) P_{15} (7,3) (7,7)

(b) Cyclic-checkerboard partitioning

Matrix Transposition

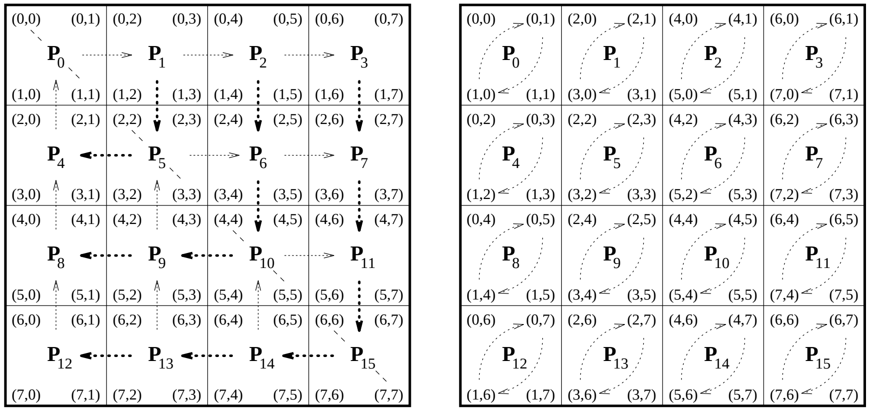
- Need to perform A[i,j] = A[j,i] for all i,j
- Checkerboard partitioning on a Mesh of processors
- Case 1: number of processors equal to array elements



(a) Communication steps (b) Final configuration

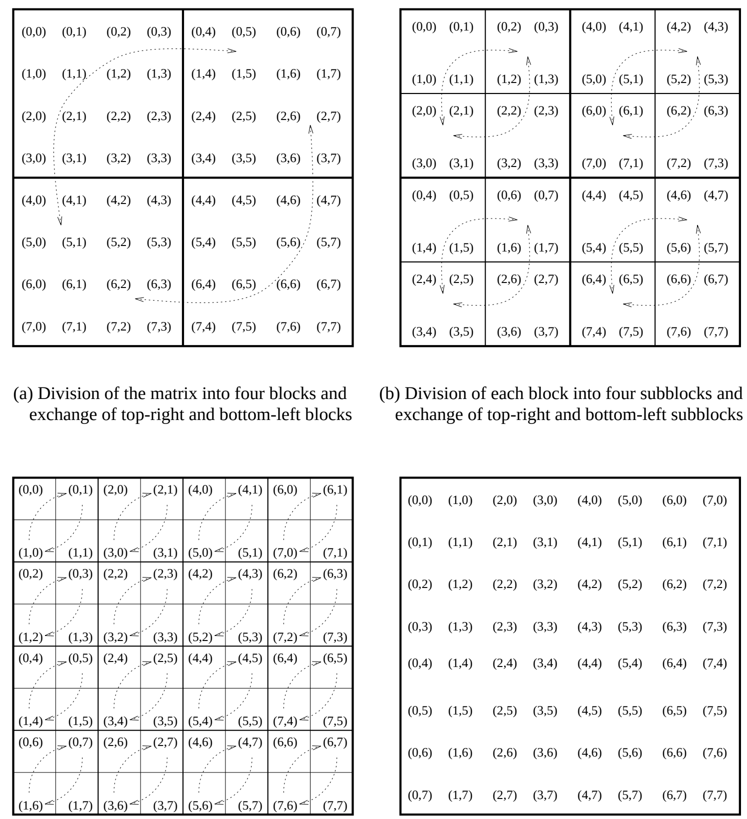
Matrix Transposition on a Mesh

- Case 2: number of processors less than array elements
- First phase: communicate blocks, second phase, local transpose on blocks
- $T_P = \frac{n^2}{2p} + 2t_s\sqrt{p} + 2t_w\frac{n^2}{\sqrt{p}}$



(a) Communication steps (b) Local rearrangement

Recursive Transposition Algorithm



(a) Division of the matrix into four blocks and exchange of top-right and bottom-left blocks

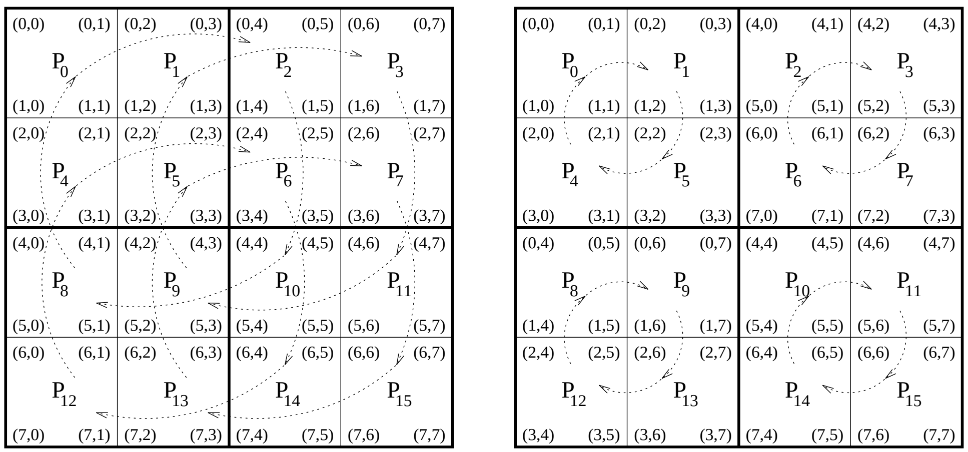
(b) Division of each block into four subblocks and exchange of top-right and bottom-left subblocks

(c) Last subdivision and transposition

(d) Final configuration

Matrix Transpose on the Hypercube

- $T_P = \frac{n^2}{2p} + (t_s + t_w \frac{n^2}{p}) log(p)$



(a)

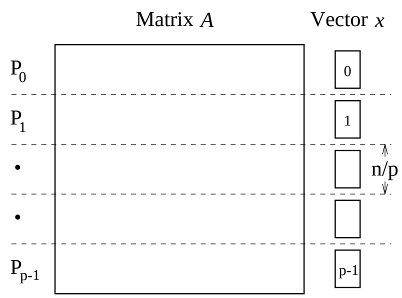
Division of the matrix into four blocks and exchange of top-right and bottom-left blocks

(b)

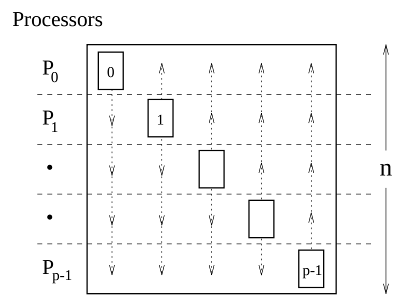
Division of each block into four subblocks and exchange of top-right and bottom-left subblocks

Matrix Vector Multiplication ($Ax = y$)

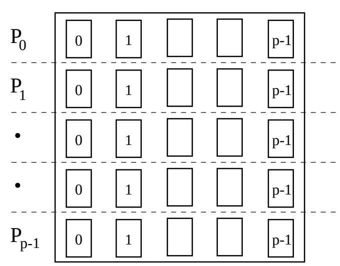
- Assume row-wise striping, one row per processor



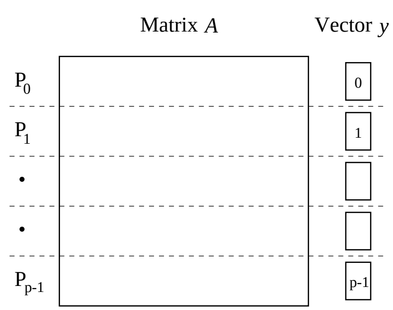
(a) Initial partitioning of the matrix and the starting vector x



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



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

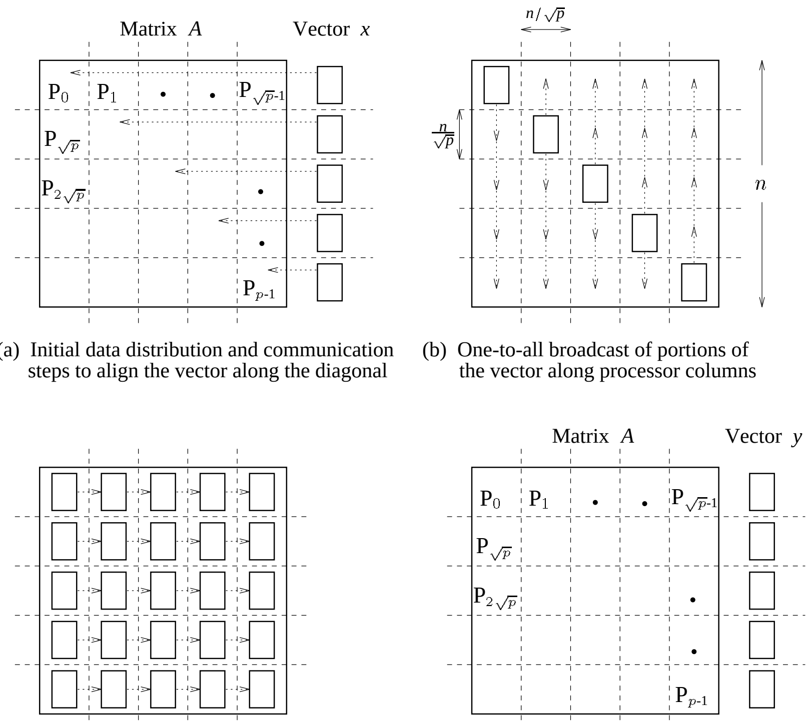


(d) Final distribution of the matrix and the result vector y

Runtime Analysis

- Assume number of processors less than number of rows
- Runtime on hypercube $T_P = \frac{n^2}{p} + t_s log(p) + t_w.(n - \frac{n}{p})$
- Runtime on torus $T_P = \frac{n^2}{p} + 2 * t_s(\sqrt{p} - 1) + t_w(n - \frac{n}{p})$

Matrix Vector Using Checkerboard Partitioning



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

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

(c) Single-node accumulation of partial results

(d) Final distribution of the result vector

Matrix Vector Analysis

- Runtime on a mesh with cut-through routing $T_P = \frac{n^2}{p} + t_s log(p) + t_w \frac{n}{\sqrt{p}} log(p) + 2 * t_h(\sqrt{p} - 1)$

- Based on the observation that the broadcast is performed on a ring

- Checkerboard (two-dim blocked) partitioning is faster than striped (one-dim blocked) for both mesh and hypercube architectures

Matrix Matrix Multiplication (C = A.B)

- Assume n X n matrices, serial algorithm takes time = n^3.
- Consider blocked matrix multiplication algorithm

```
procedure BLOCK_MAT_MULT (A, B, C)
begin
    for i := 0 to q − 1 do
        for j := 0 to q − 1 do
            begin
                Initialize all elements of C_{i,j} to zero;
                for k := 0 to q − 1 do
                    C_{i,j} := C_{i,j} + A_{i,k} × B_{k,j};
            endfor;
end BLOCK_MAT_MULT
```

Program 5.3 The block matrix multiplication algorithm for $n \times n$ matrices with a block size of $(n/q) \times (n/q)$.
Copyright (r) 1994 Benjamin/Cummings Publishing Co.

Simple Parallel Matrix Multiplication

- Consider two nXn matrices A and B partitioned into p blocks arranged as $\sqrt{p}X\sqrt{p}$ logical mesh of processors
- Processor $P_{i,j}$ initially stores $A_{i,j}$ and $B_{i,j}$ computes block $C_{i,j}$ of result matrix.
- In subsequent stages, $C_{i,j}$ computation requires all submatrices $A_{i,j}$ and $B_{k,j}$ for all $0 \leq k \leq \sqrt{p}$.
- An all-to-all broadcast is performed of matrix A's blocks for each row of the processors, and an all-to-all broadcast of matrix B's blocks in each column.
- Parallel run time on hypercube $T_P = \frac{n^3}{p} + t_s log(p) + 2 * t_w \frac{n^2}{p}(\sqrt{p} - 1)$
- Parallel run time on torus $T_P = \frac{n^3}{p} + 2 * t_s(\sqrt{p} - 1) + 2 * t_w \frac{n^2}{p}(\sqrt{p} - 1)$

Memory Efficient Matrix Multiplication

- Partition matrices A and B among p processors so each processor stores $(n/\sqrt{p}$ by $(n/\sqrt{p}$ blocks of each matrix.

- Algorithm performs $\sqrt{p}$ iterations of the following steps:

- Broadcast the selected block of A among the $\sqrt{p}$ processors of the row in which the block lies (initially start with block $A_{i,i}$).

- Multiply the block of A received with the resident block of B.

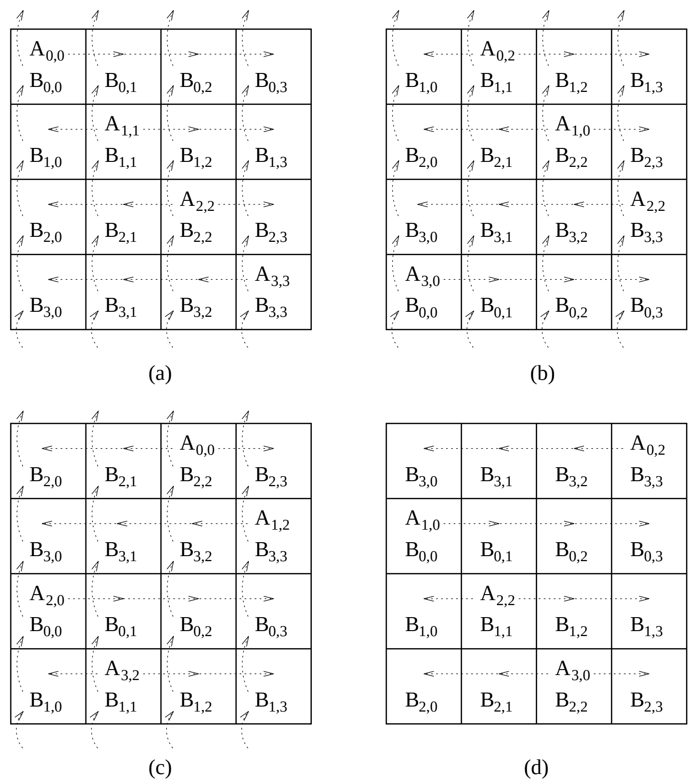
- Send the block of B to processor directly above it (with wraparound), and received new block of B from processor below it.

- Select the block of A for the next row broadcast.

- Parallel runtime of algorithm on the hypercube

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

Memory Efficient Matrix Multiplication



Solving System of Linear Equations

- Given system of linear equations $A.x = b$, solve for vector x in two stages.

- Reduce system of equations to a upper triangular system of equations $U.x = y$

- Finally solve for x using back substitution

- When this needs to be solved multiple times, usually perform LU -factorization using similar ideas.

- Serial runtime takes $n^2/2$ divisions and $(n^3/3) - (n^2/2)$ subtractions and multiplications, total approximately, $W = \frac{2}{3}n^3$

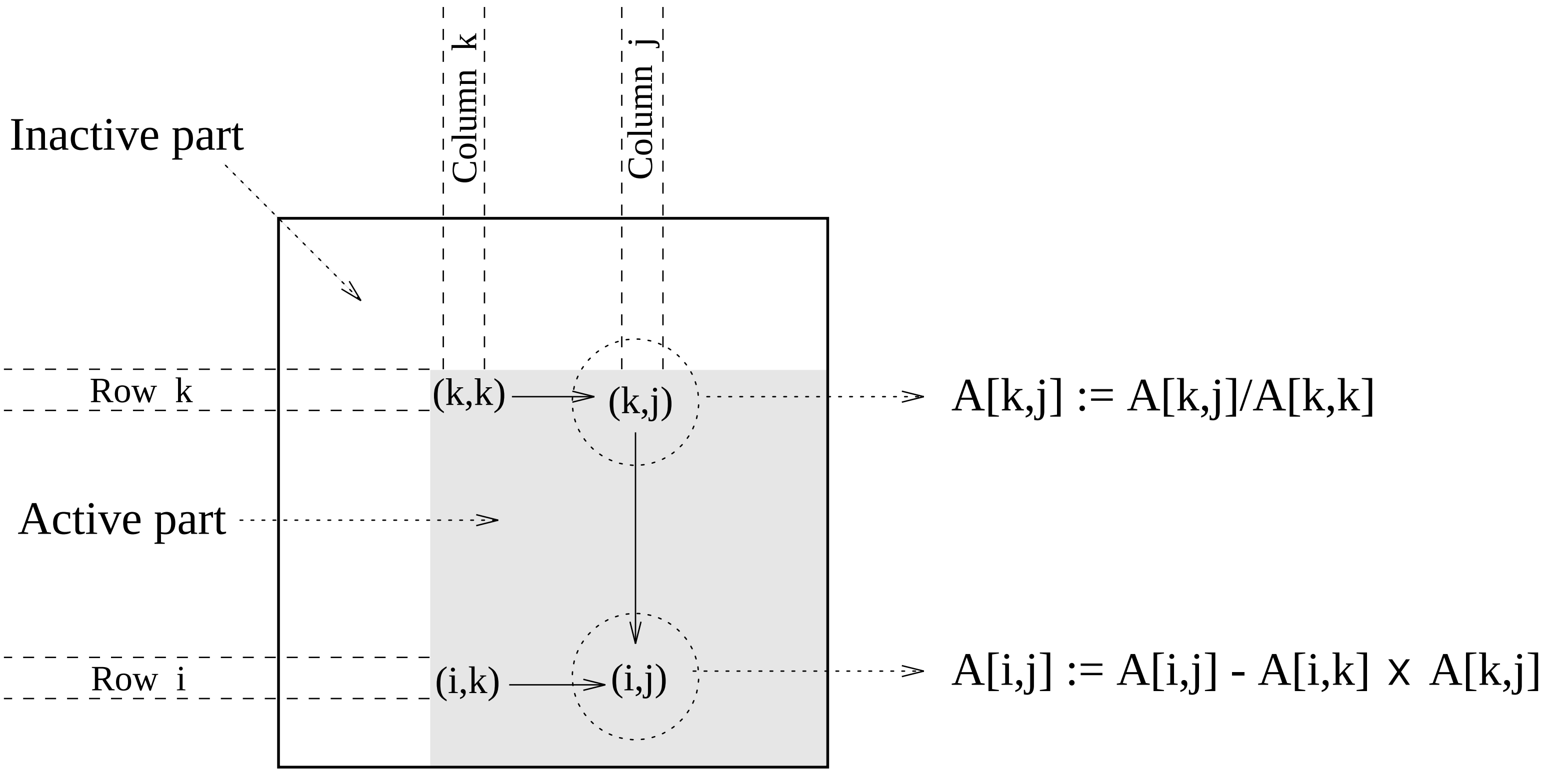
Serial Gaussian Elimination Algorithm

```
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
```

Program 5.4 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 7.

Copyright (r) 1994 Benjamin/Cummings Publishing Co.

Serial Gaussian Elimination Algorithm-Figure



Parallel Gaussian Elimination Algorithm

- Can use either two-dimensional blocked or cyclic distribution of matrix

1	(0,1)	(0,2)	(0,3)	(0,4)	(0,5)	(0,6)	(0,7)
0	1	(1,2)	(1,3)	(1,4)	(1,5)	(1,6)	(1,7)
0	0	1	(2,3)	(2,4)	(2,5)	(2,6)	(2,7)
0	0	0	(3,3)	(3,4)	(3,5)	(3,6)	(3,7)
0	0	0	(4,3)	(4,4)	(4,5)	(4,6)	(4,7)
0	0	0	(5,3)	(5,4)	(5,5)	(5,6)	(5,7)
0	0	0	(6,3)	(6,4)	(6,5)	(6,6)	(6,7)
0	0	0	(7,3)	(7,4)	(7,5)	(7,6)	(7,7)

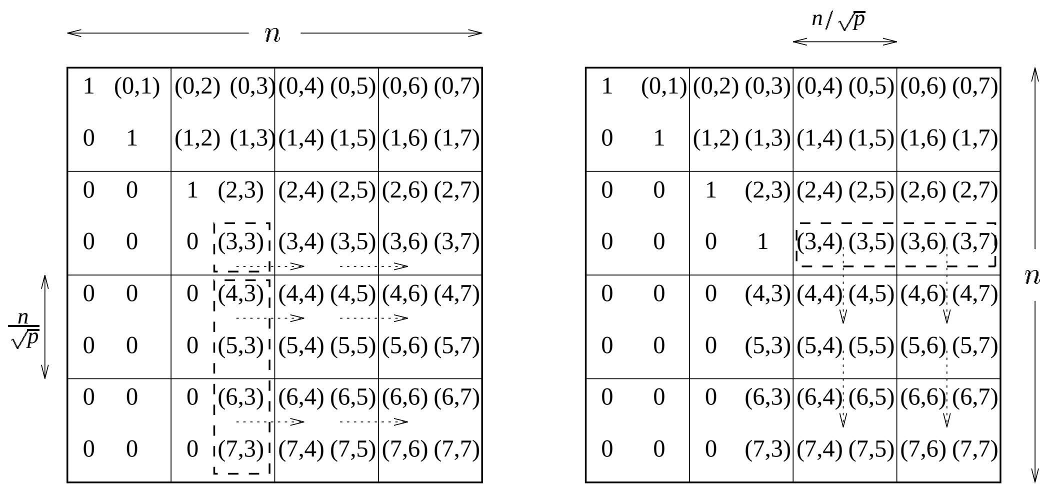
(a) Block-checkerboard mapping

1	(0,4)	(0,1)	(0,5)	(0,2)	(0,6)	(0,3)	(0,7)
0	(4,4)	0	(4,5)	0	(4,6)	(4,3)	(4,7)
0	(1,4)	1	(1,5)	(1,2)	(1,6)	(1,3)	(1,7)
0	(5,4)	0	(5,5)	0	(5,6)	(5,3)	(5,7)
0	(2,4)	0	(2,5)	1	(2,6)	(2,3)	(2,7)
0	(6,4)	0	(6,5)	0	(6,6)	(6,3)	(6,7)
0	(3,4)	0	(3,5)	0	(3,6)	(3,3)	(3,7)
0	(7,4)	0	(7,5)	0	(7,6)	(7,3)	(7,7)

(b) Cyclic-checkerboard mapping

Parallel Gaussian Elimination Algorithm

- Communication steps in step k=3 for 8X8 matrix on 16 processors.



(a) Rowwise broadcast of A[i,k] for i = k to (n - 1)

(b) Columnwise broadcast of A[k,j] for j = (k + 1) to (n - 1)

Optimizations on Parallel Algorithm

- In previous algorithm, processors worked synchronously in each step.
- At any given time, all processors worked on the same iteration, i.e. the (k+1)st iteration started only after all the computation and communication of the kth iteration was complete.
- Can design an asynchronous, or pipelined algorithm, where some processors perform the computation of a given iteration ealier than other processors, and start working on subsequent iterations sooner.
- Can overlap computations with communication.

Overview of Pipelined Parallel GE

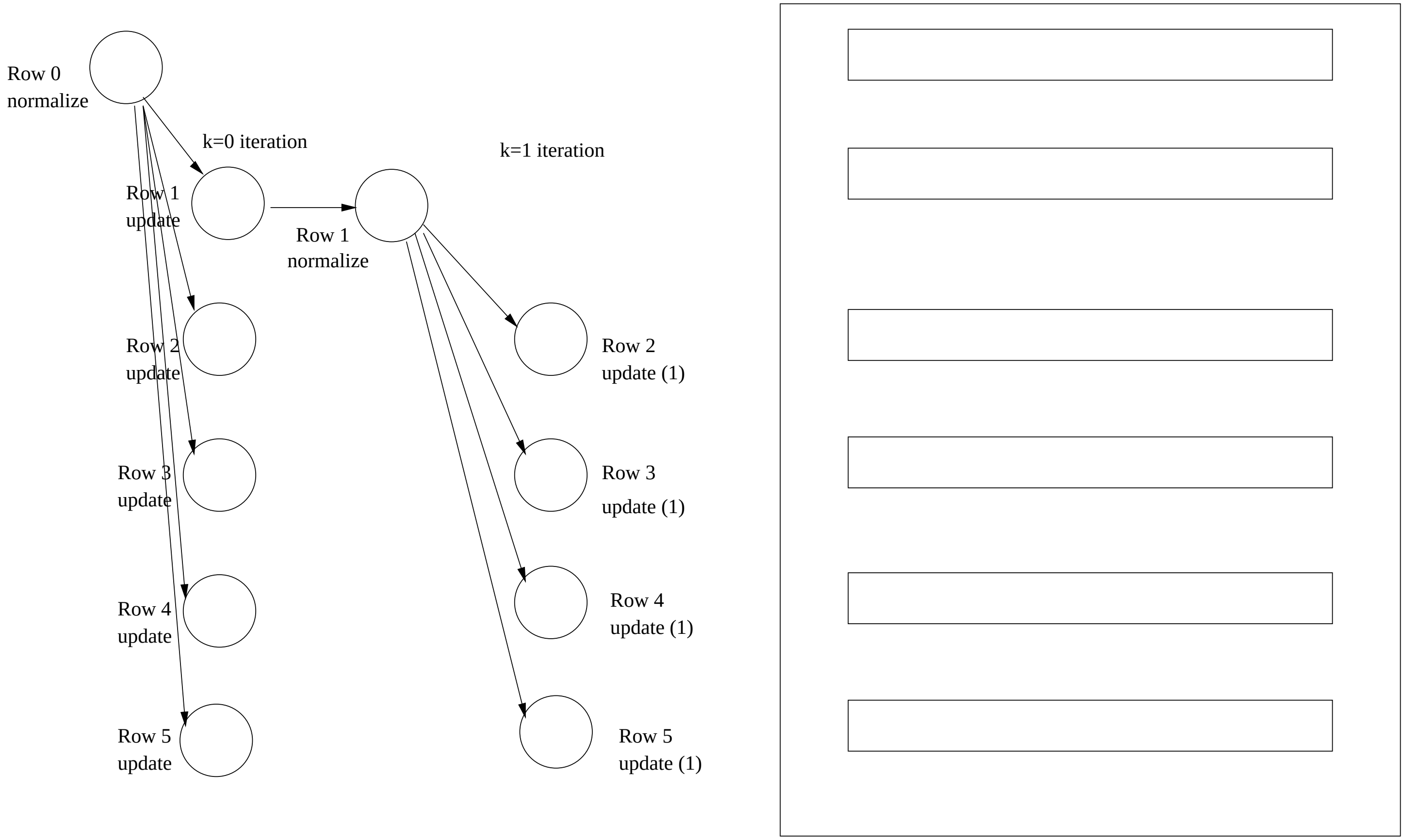
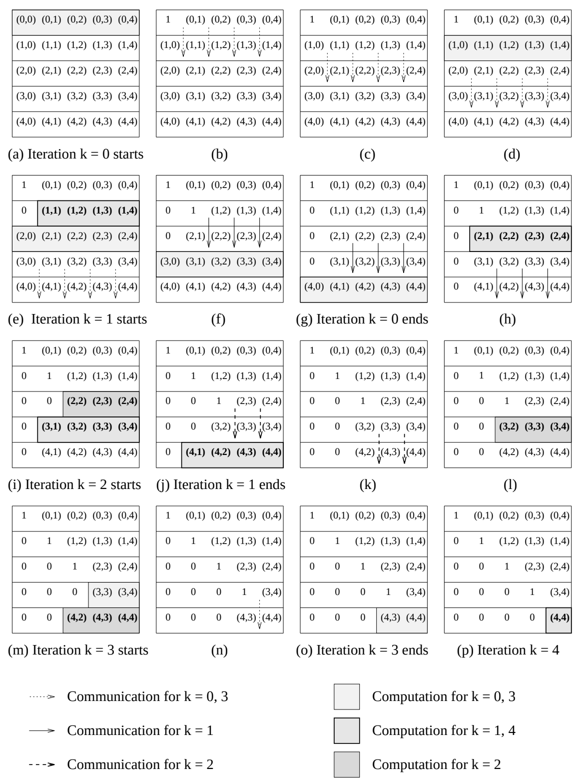


Illustration of Pipelined Parallel GE

- Assume that processor can do only one of compute/send/receive at a time



Summary

- Design of parallel algorithms
- Dense matrix representations
- Matrix transpose algorithms
- Matrix vector multiplication
- Matrix matrix multiplication
- Linear system of equation solvers
- NEXT LECTURE: Sparse Matrix Algorithms