Parallel Programming
with MPI and OpenMP

Michael J. Quinn

Chapter 6

Floyd's Algorithm

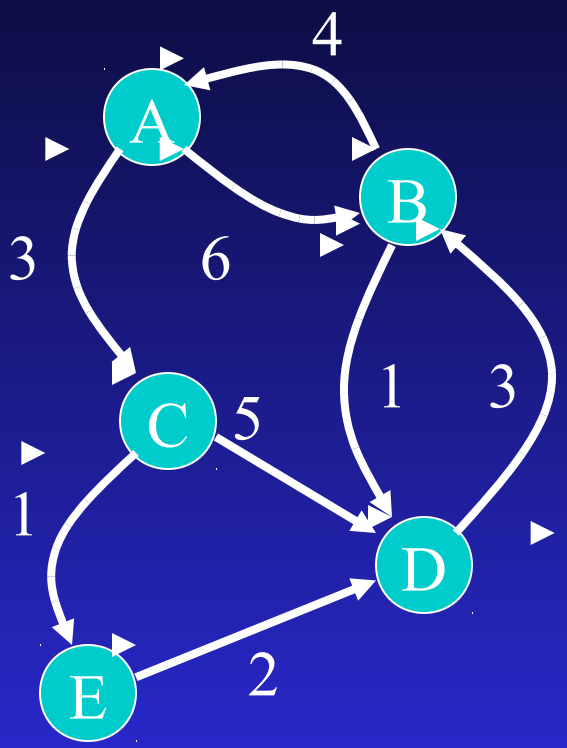
Chapter Objectives

- Creating 2-D arrays
- Thinking about “grain size”
- Introducing point-to-point communications
- Reading and printing 2-D matrices
- Analyzing performance when computations and communications overlap

Outline

- All-pairs shortest path problem
- Dynamic 2-D arrays
- Parallel algorithm design
- Point-to-point communication
- Block row matrix I/O
- Analysis and benchmarking

All-pairs Shortest Path Problem



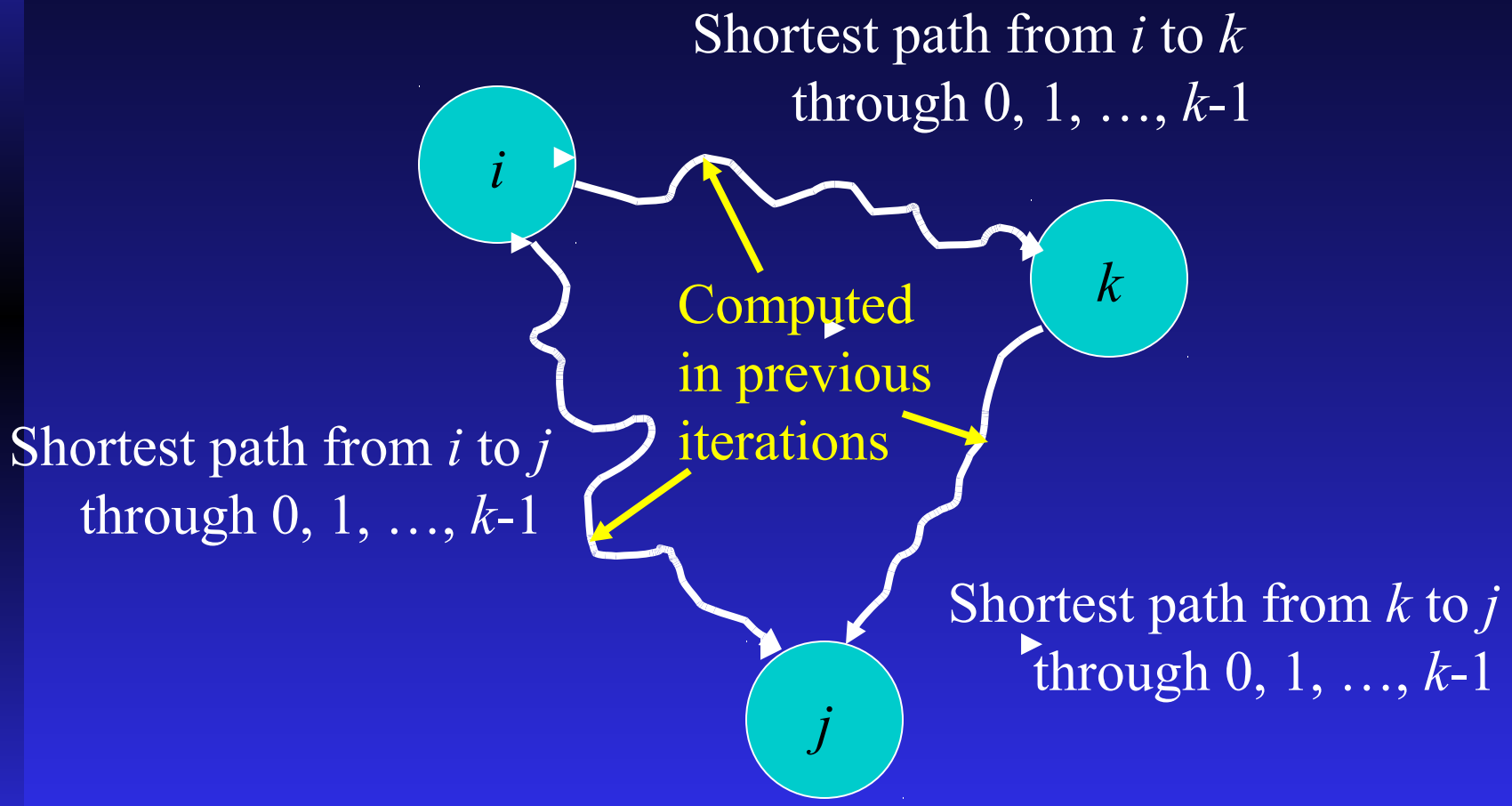
	A	B	C	D	E
A	0	6	3	6	4
B	4	0	7	10	8
C	12	6	0	3	1
D	7	3	10	0	11
E	9	5	12	2	0

Resulting Adjacency Matrix Containing Distances

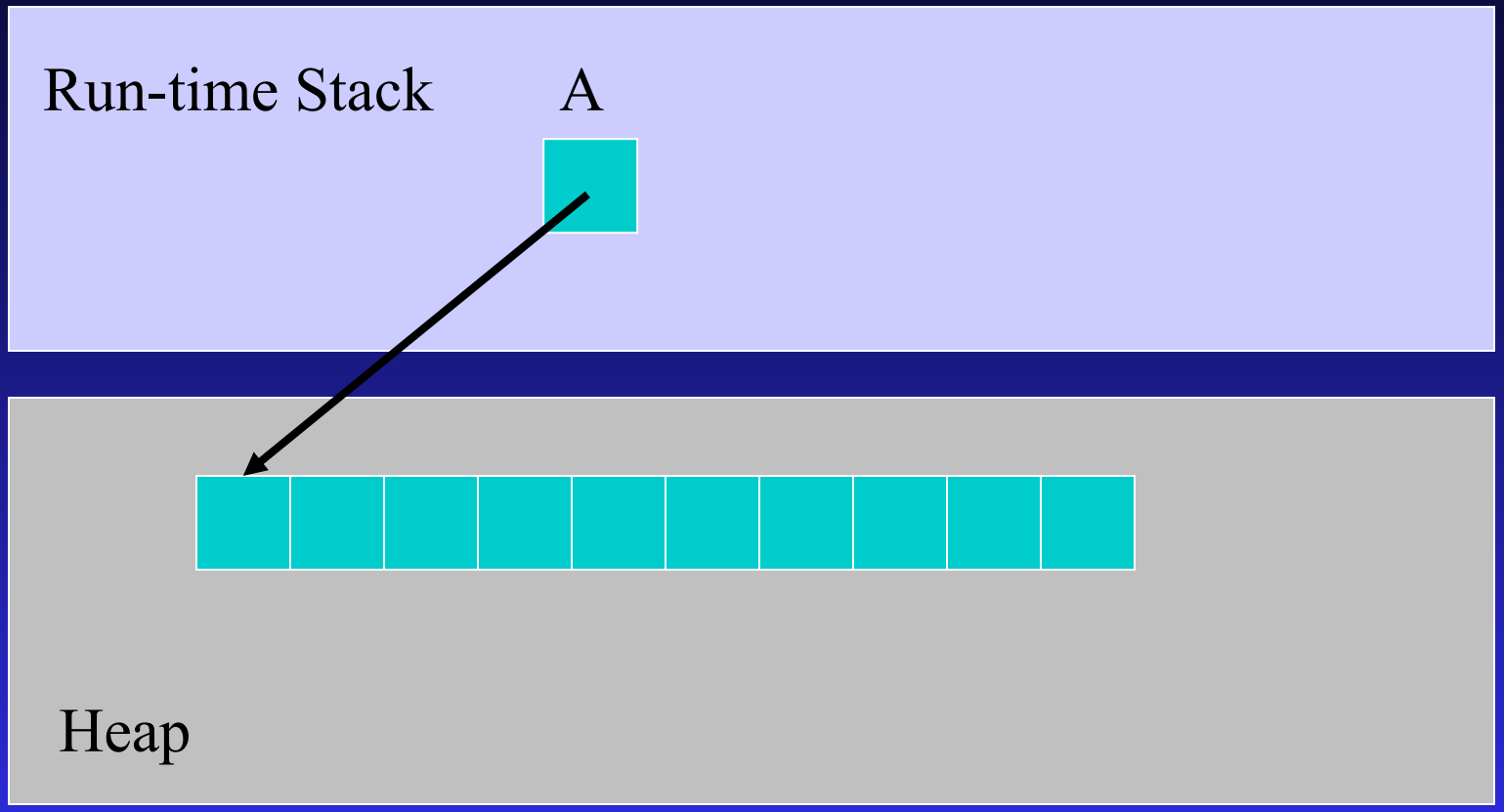
Floyd's Algorithm

```
for k ← 0 to n-1
        for i ← 0 to n-1
                for j ← 0 to n-1
                        a[i,j] ← min (a[i,j], a[i,k] + a[k,j])
                endfor
        endfor
endfor
```

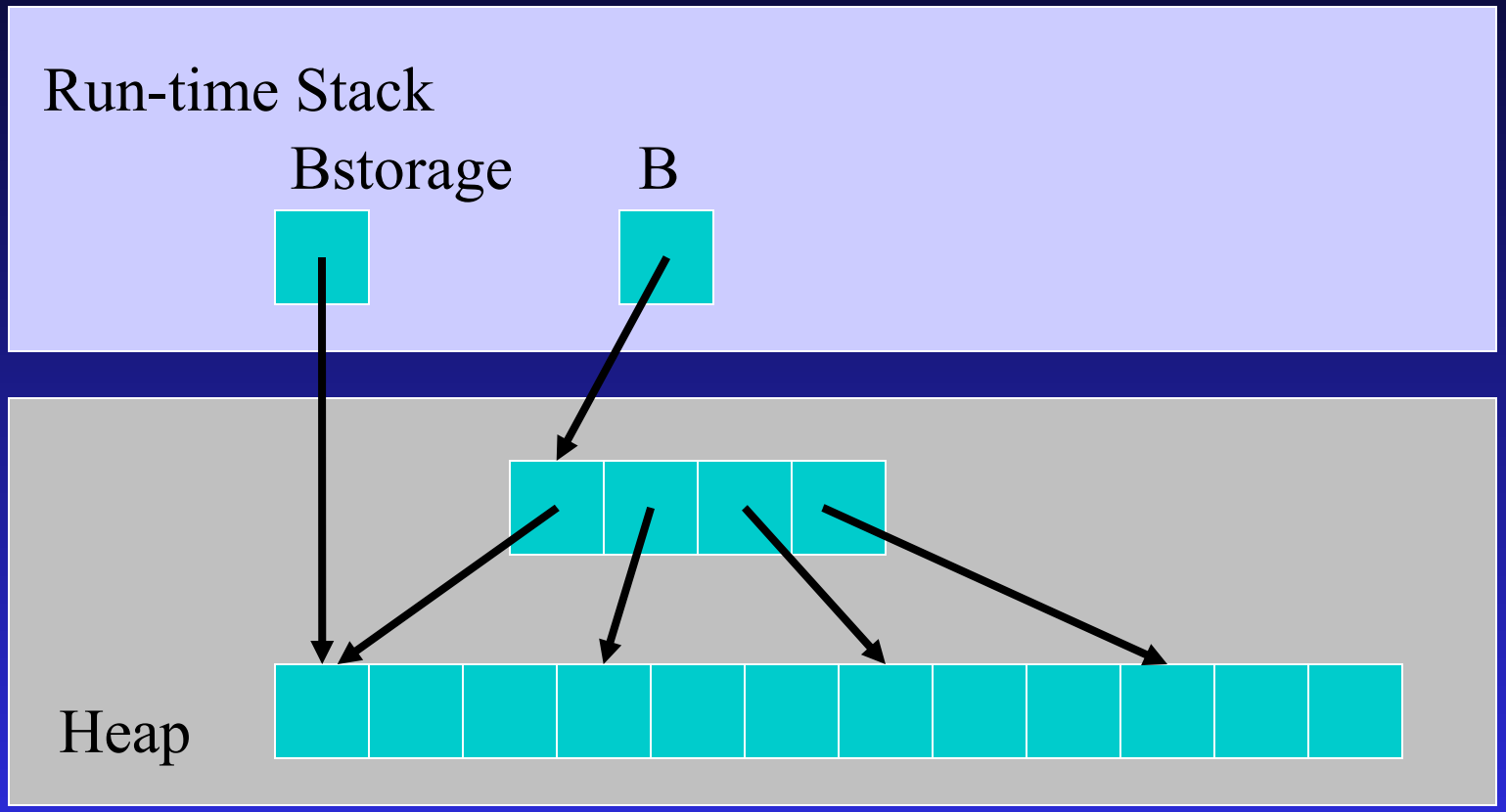
Why It Works


Shortest path from i to k through 0, 1, …, k-1
i
k
Computed in previous iterations
Shortest path from i to j through 0, 1, …, k-1
Shortest path from k to j through 0, 1, …, k-1
j

Dynamic 1-D Array Creation



Dynamic 2-D Array Creation



Designing Parallel Algorithm

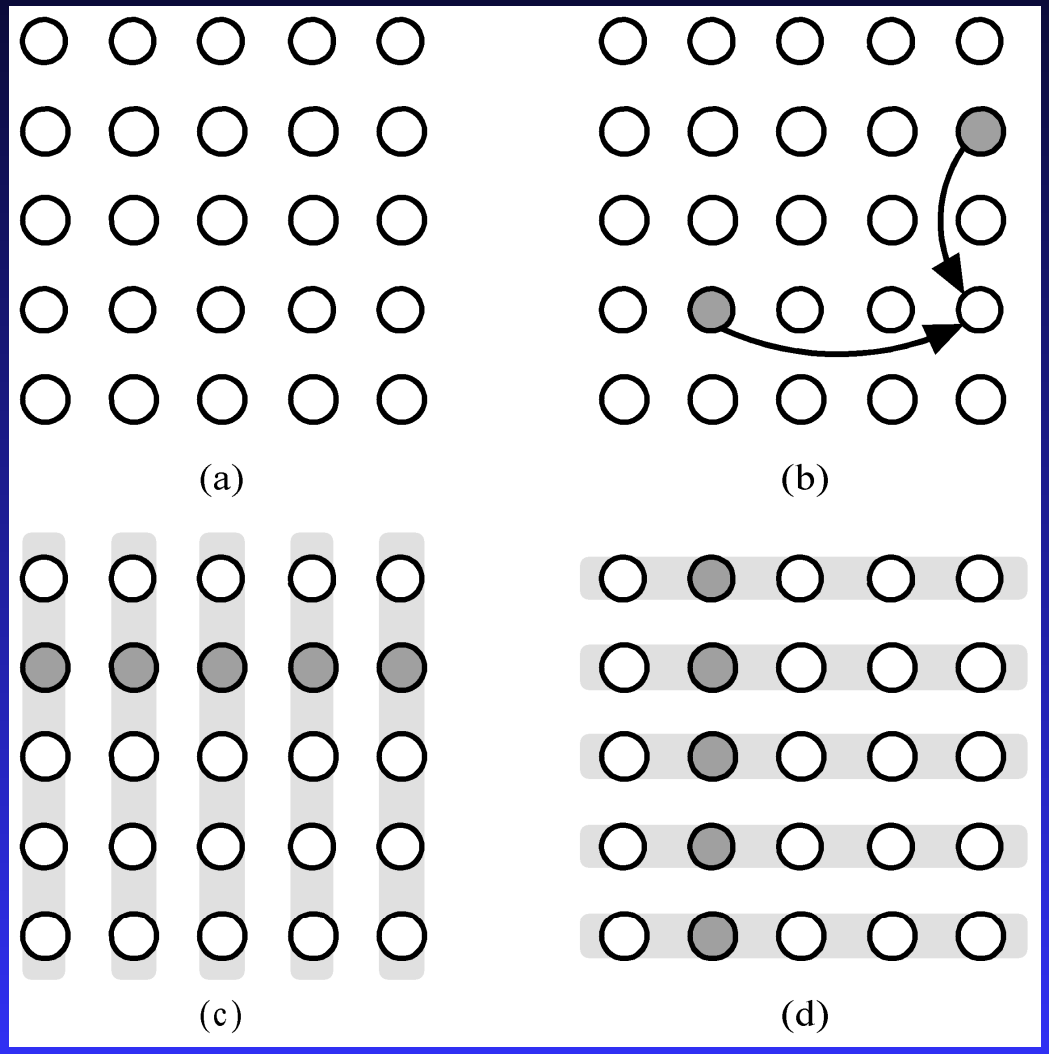
- Partitioning
- Communication
- Agglomeration and Mapping

Partitioning

- Domain or functional decomposition?
- Look at pseudocode
- Same assignment statement executed n^3 times
- No functional parallelism
- Domain decomposition: divide matrix **A** into its n^2 elements

Communication

Primitive tasks



Updating *a[3,4]* when *k* = 1

Iteration *k*: every task in row *k* broadcasts its value w/in task column

Iteration *k*: every task in column *k* broadcasts its value w/in task row

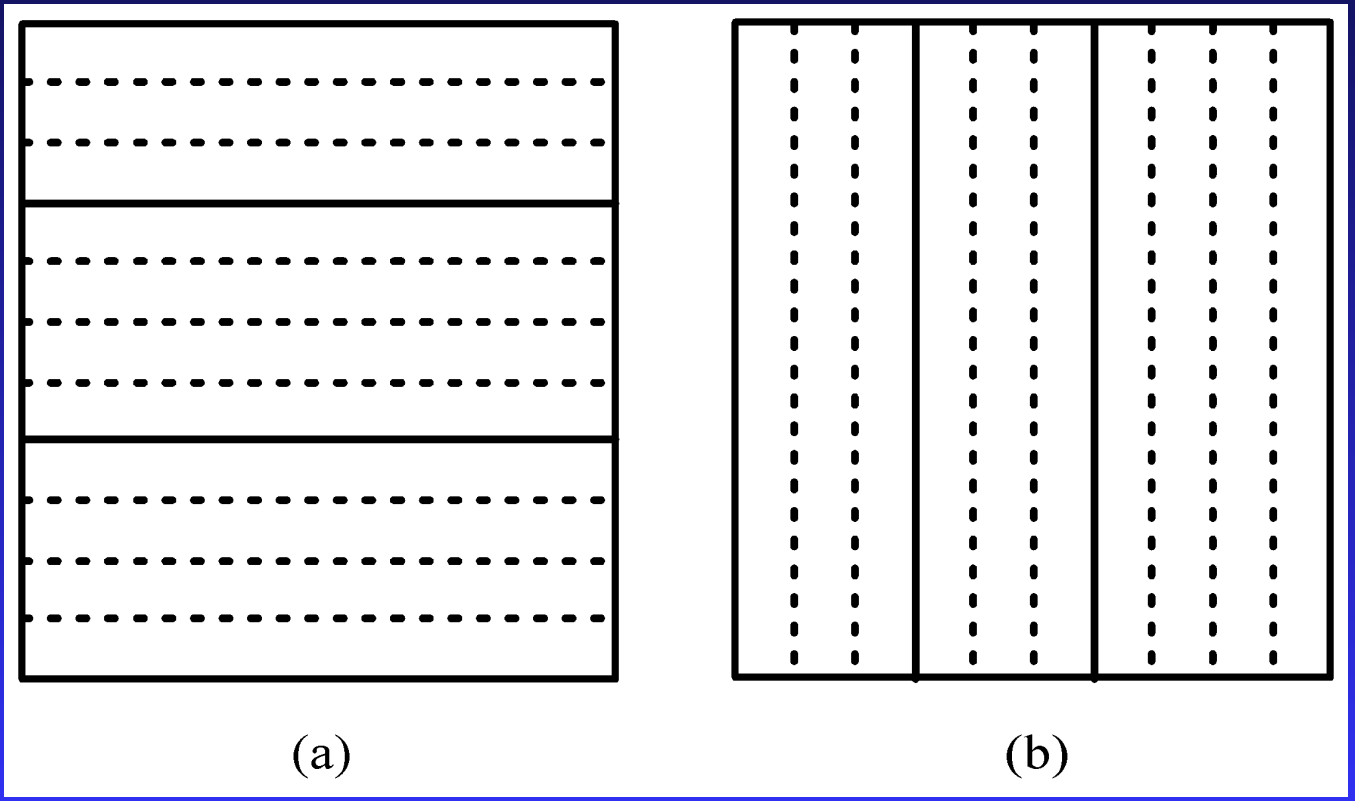
Agglomeration and Mapping

- Number of tasks: static
- Communication among tasks: structured
- Computation time per task: constant
- Strategy:
  - Agglomerate tasks to minimize communication
  - Create one task per MPI process

Two Data Decompositions

Rowwise block striped

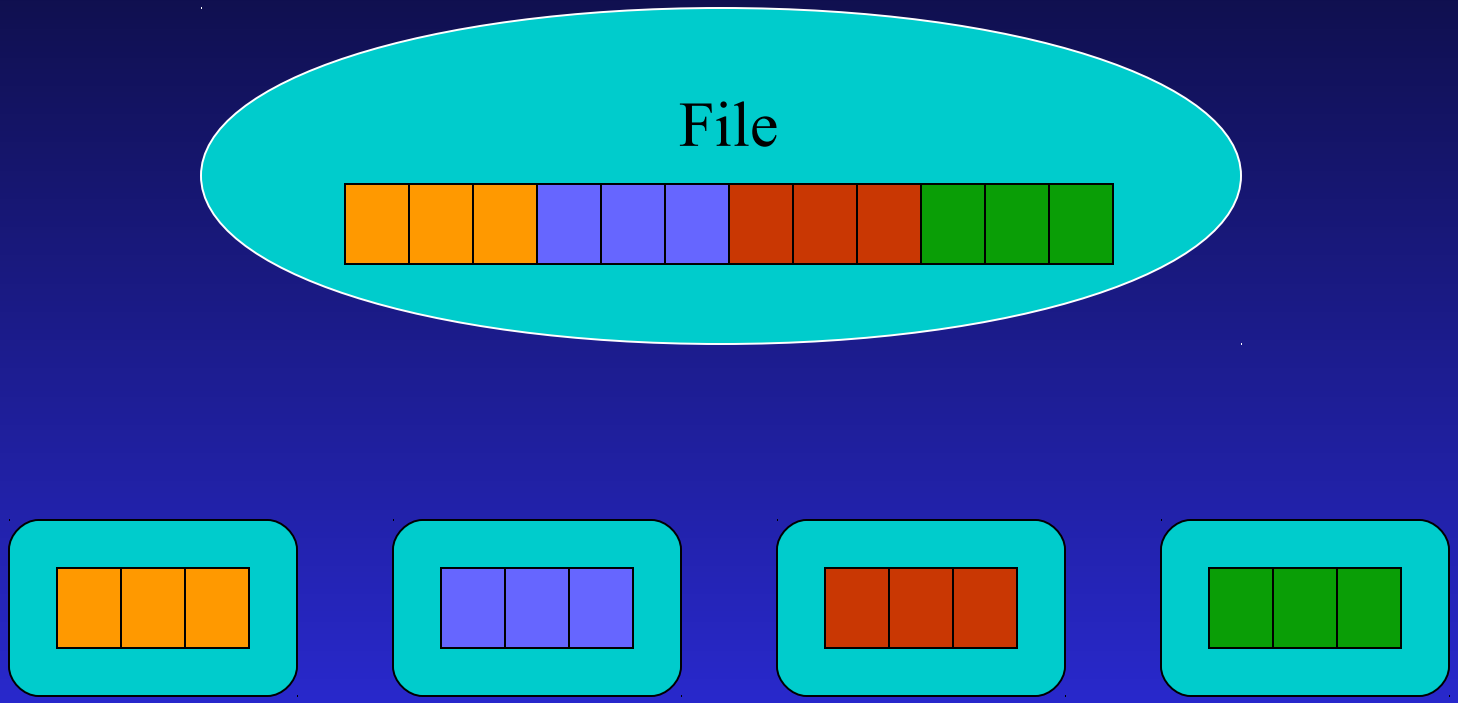
Columnwise block striped



Comparing Decompositions

- Columnwise block striped
  - Broadcast within columns eliminated
- Rowwise block striped
  - Broadcast within rows eliminated
  - Reading matrix from file simpler
- Choose rowwise block striped decomposition

File Input



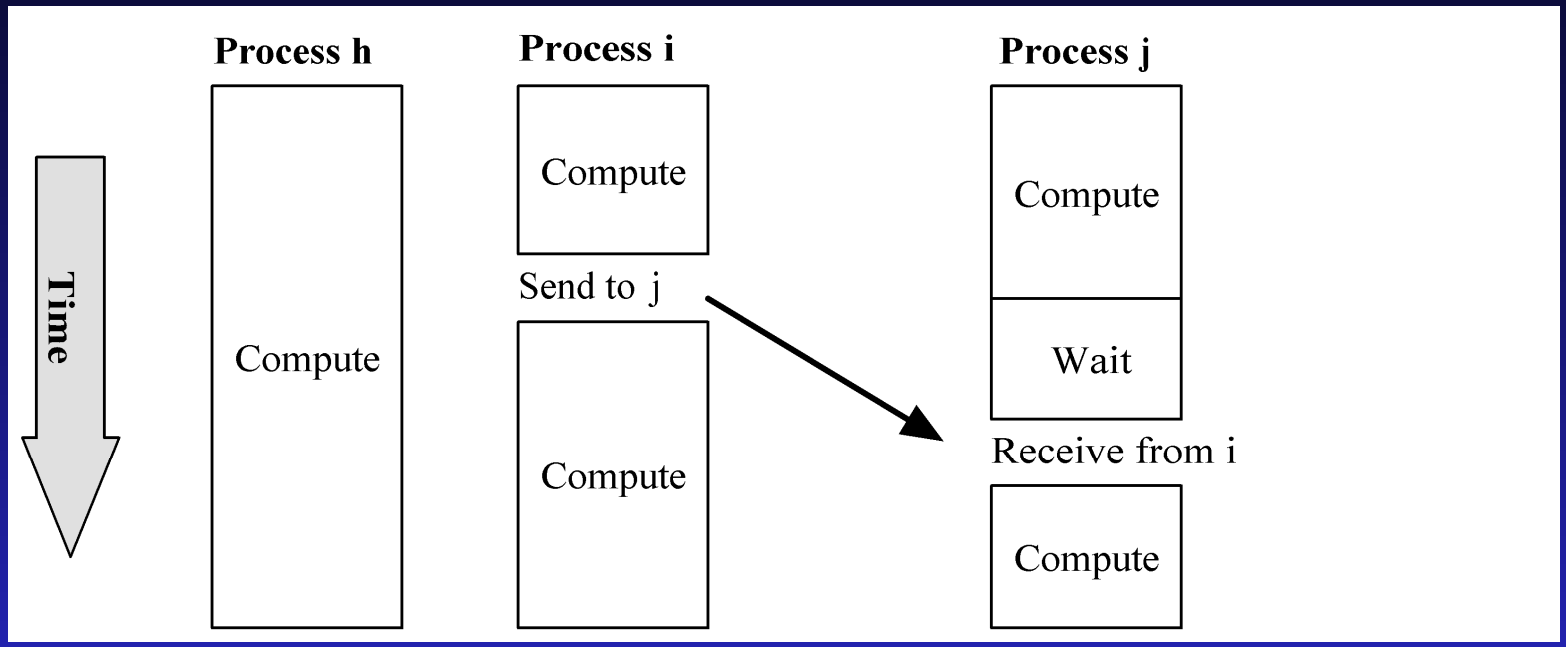
Pop Quiz

Why don’t we input the entire file at once and then scatter its contents among the processes, allowing concurrent message passing?

Point-to-point Communication

- Involves a pair of processes
- One process sends a message
- Other process receives the message

Send/Receive Not Collective



Function MPI_Send

```
int MPI_Send (
        void            *message,
        int              count,
        MPI_Datatype     datatype,
        int              dest,
        int              tag,
        MPI_Comm         comm
)
```

Function MPI_Recv

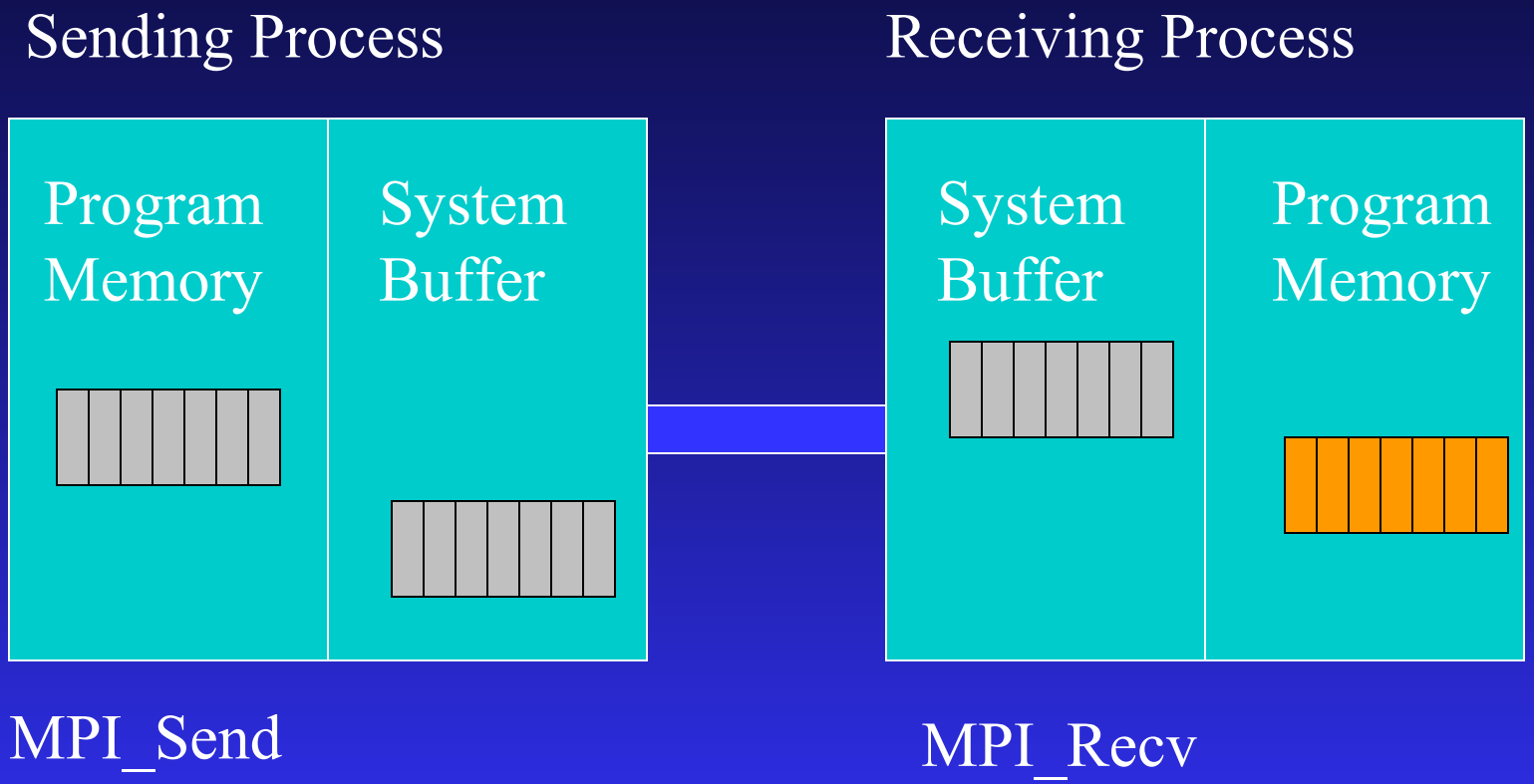
```
int MPI_Recv (
        void            *message,
        int              count,
        MPI_Datatype     datatype,
        int              source,
        int              tag,
        MPI_Comm         comm,
        MPI_Status      *status
)
```

Coding Send/Receive

```
…
if (ID == j) {
    …
    Receive from I
    …
}
…
if (ID == i) {
    …
    Send to j
    …
}
…
```

Receive is before Send.
Why does this work?

Inside MPI_Send and MPI_Recv



Return from MPI_Send

- Function blocks until message buffer free
- Message buffer is free when
  - Message copied to system buffer, or
  - Message transmitted
- Typical scenario
  - Message copied to system buffer
  - Transmission overlaps computation

Return from MPI_Recv

- Function blocks until message in buffer
- If message never arrives, function never returns

Deadlock

- Deadlock: process waiting for a condition that will never become true
- Easy to write send/receive code that deadlocks
  - Two processes: both receive before send
  - Send tag doesn't match receive tag
  - Process sends message to wrong destination process

Computational Complexity

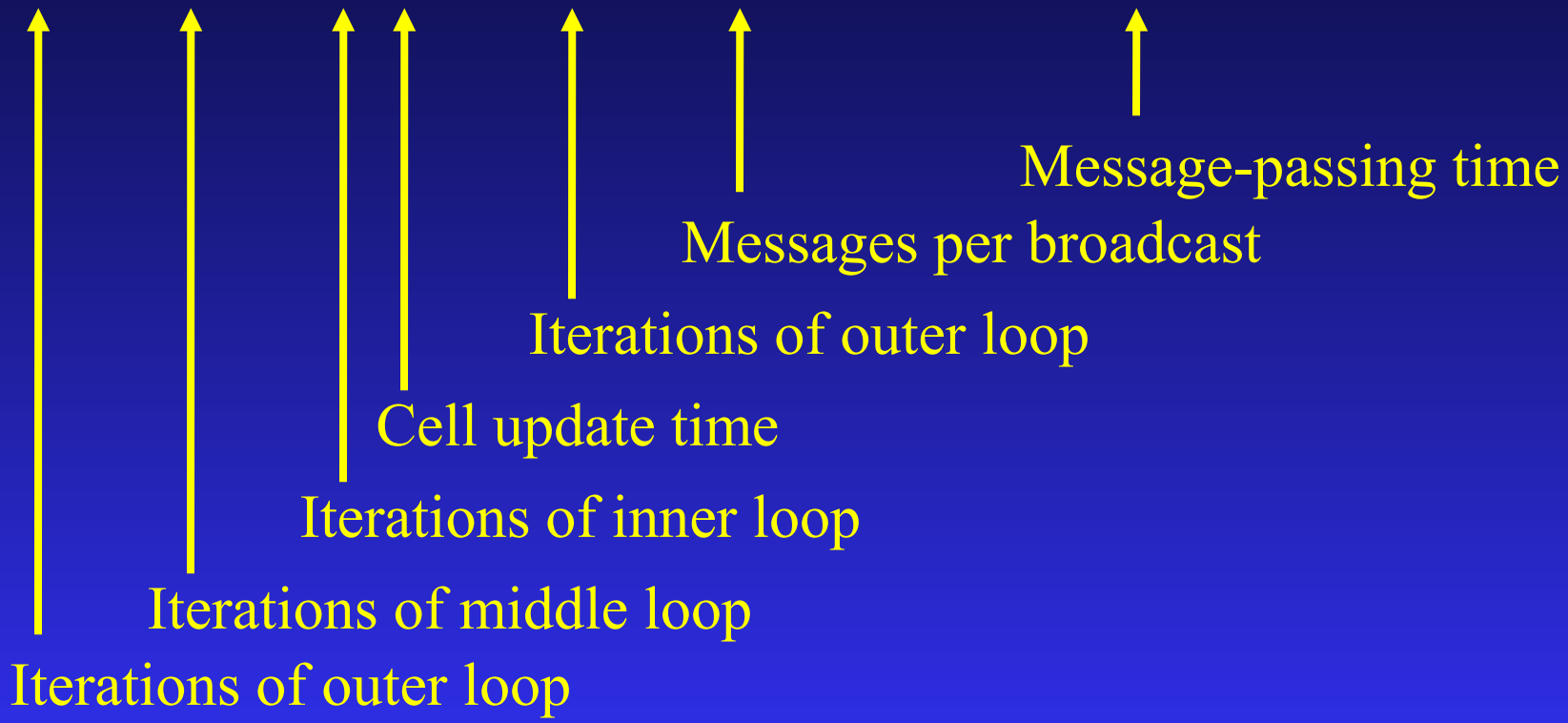
- Innermost loop has complexity $\Theta(n)$
- Middle loop executed at most $\lceil n/p \rceil$ times
- Outer loop executed n times
- Overall complexity $\Theta(n^3/p)$

Communication Complexity

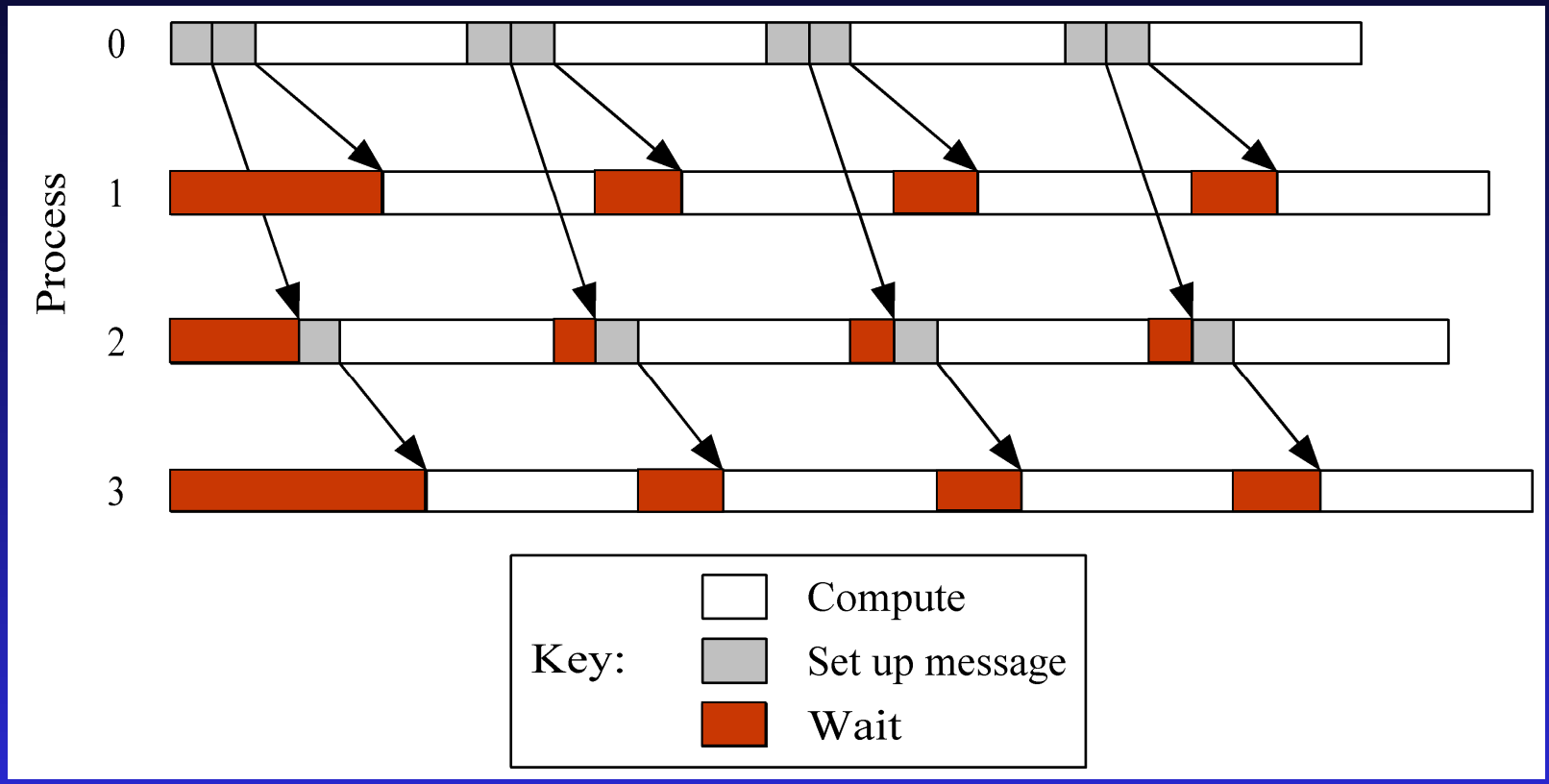
- No communication in inner loop
- No communication in middle loop
- Broadcast in outer loop — complexity is $\Theta(n \log p)$
- Overall complexity $\Theta(n^2 \log p)$

Execution Time Expression (1)

$$n\lceil n/p\rceil n\chi + n\lceil \log p\rceil(\lambda + 4n/\beta)$$

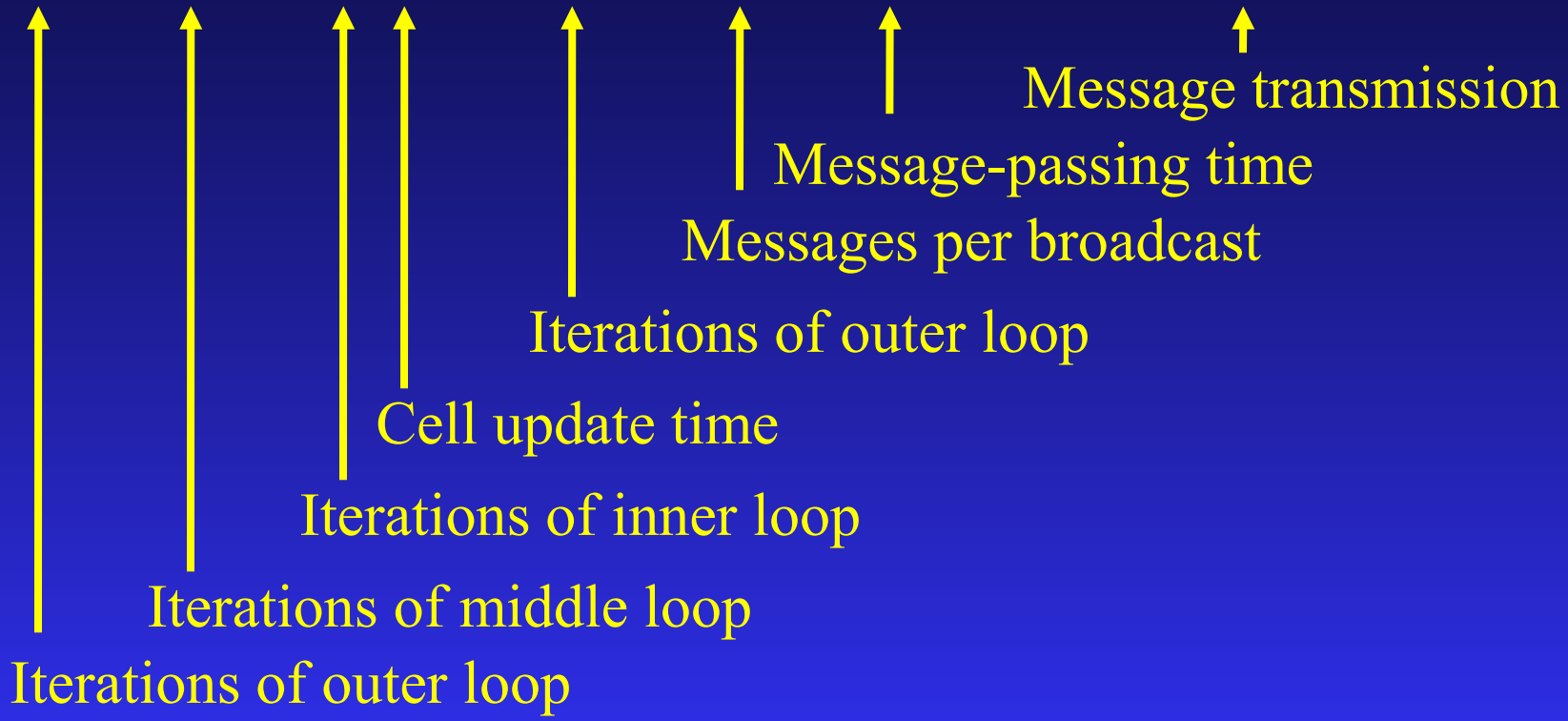


Computation/communication Overlap



Execution Time Expression (2)

$$n\lceil n/p\rceil n\chi + n\lceil \log p\rceil \lambda + \lceil \log p\rceil 4n/\beta$$



Predicted vs. Actual Performance

	Execution Time (sec)	
Processes	Predicted	Actual
1	25.54	25.54
2	13.02	13.89
3	9.01	9.60
4	6.89	7.29
5	5.86	5.99
6	5.01	5.16
7	4.40	4.50
8	3.94	3.98

Summary

- Two matrix decompositions
  - Rowwise block striped
  - Columnwise block striped
- Blocking send/receive functions
  - MPI_Send
  - MPI_Recv
- Overlapping communications with computations