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

Resulting Adjacency Matrix Containing Distances
Floyd’s Algorithm

for \( k \leftarrow 0 \) to \( n-1 \)
  for \( i \leftarrow 0 \) to \( n-1 \)
    for \( j \leftarrow 0 \) to \( n-1 \)
      \( a[i,j] \leftarrow \min(a[i,j], a[i,k] + a[k,j]) \)
    endfor
  endfor
endfor
Why It Works

Shortest path from $i$ to $k$ through 0, 1, …, $k-1$

Computed in previous iterations

Shortest path from $i$ to $j$ through 0, 1, …, $k-1$

Shortest path from $k$ to $j$ through 0, 1, …, $k-1$
Dynamic 1-D Array Creation

Run-time Stack  A

Heap
Dynamic 2-D Array Creation

Run-time Stack

BstORAGE

B

Heap
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

Iteration $k$: every task in row $k$ broadcasts its value within task column

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

Iteration $k$: every task in column $k$ broadcasts its value within 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

(a) (b)
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

File

---

This image represents a diagram of file input. Each rectangle likely corresponds to a data representation or a file type. The diagram visually illustrates the concept of file input, typically associated with data processing or computer science.
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

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

```c
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

Sending Process

Program Memory

System Buffer

MPI_Send

Receiving Process

System Buffer

Program Memory

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 $\Theta(n \log \rho)$
• Overall complexity $\Theta(n^2 \log \rho)$
Execution Time Expression (1)

\[ n \left\lfloor \frac{n}{p} \right\rfloor n \chi + n \left\lfloor \log p \right\rfloor (\lambda + 4n/\beta) \]

- \( n \left\lfloor \frac{n}{p} \right\rfloor n \chi \): Cell update time
- \( n \left\lfloor \log p \right\rfloor \): Messages per broadcast
- \( \lambda \): Iterations of outer loop
- \( 4n/\beta \): Iterations of inner loop
- \( n \left\lfloor \log p \right\rfloor (\lambda + 4n/\beta) \): Message-passing time
Computation/communication Overlap

0

1

2

3

Key:
- Compute
- Set up message
- Wait
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

<table>
<thead>
<tr>
<th>Processes</th>
<th>Execution Time (sec)</th>
<th>Actual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted</td>
<td>Actual</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>25.54</td>
<td>25.54</td>
</tr>
<tr>
<td>2</td>
<td>13.02</td>
<td>13.89</td>
</tr>
<tr>
<td>3</td>
<td>9.01</td>
<td>9.60</td>
</tr>
<tr>
<td>4</td>
<td>6.89</td>
<td>7.29</td>
</tr>
<tr>
<td>5</td>
<td>5.86</td>
<td>5.99</td>
</tr>
<tr>
<td>6</td>
<td>5.01</td>
<td>5.16</td>
</tr>
<tr>
<td>7</td>
<td>4.40</td>
<td>4.50</td>
</tr>
<tr>
<td>8</td>
<td>3.94</td>
<td>3.98</td>
</tr>
</tbody>
</table>
Summary

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