Parallel Programming
in C with MPI and OpenMP
Parallel Programming
in C with MPI and OpenMP

Michael J. Quinn
Algorithm design and basic algorithms

Slides are modified from those found in *Parallel Programming in C with MPI and OpenMP*, Michael Quinn
Outline
Outline

- Task/channel model
Outline

- Task/channel model
- Algorithm design methodology
Outline

- Task/channel model
- Algorithm design methodology
- Case studies
Task/Channel Model
Task/Channel Model

- Parallel computation = set of tasks
Task/Channel Model

- Parallel computation = set of tasks
- Task
Task/Channel Model

- Parallel computation = set of tasks
- Task
  - Program
Task/Channel Model

- Parallel computation = set of tasks
- Task
  - Program
  - Local memory
Task/Channel Model

- Parallel computation = set of tasks
- Task
  - Program
  - Local memory
  - Collection of I/O ports
Task/Channel Model

- Parallel computation = set of tasks
- Task
  - Program
  - Local memory
  - Collection of I/O ports
- Tasks interact by sending messages through channels
Task/Channel Model
Foster’s Design Methodology
Foster’s Design Methodology

- Partitioning
Foster’s Design Methodology

- Partitioning
- Communication
Foster’s Design Methodology

- Partitioning
- Communication
- Agglomeration
Foster’s Design Methodology

- Partitioning
- Communication
- Agglomeration
- Mapping
Foster’s Methodology

Problem

Partitioning

Communication

Mapping

Agglomeration
Partitioning
Partitioning

- Dividing computation and data into pieces
Partitioning

- Dividing computation and data into pieces
- Domain decomposition
Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
- Functional decomposition
Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
- Functional decomposition
  - Divide computation into pieces
Partitioning

- Dividing computation and data into pieces
- Domain decomposition
  - Divide data into pieces
  - Determine how to associate computations with the data
- Functional decomposition
  - Divide computation into pieces
  - Determine how to associate data with the computations
Example Domain Decompositions

Primitive tasks is the number of scope, or order of magnitude, of the parallelism.

1-D has, in the example, n-way ||ism along the n-faces, 2-D has \( n^2 \) ||ism along the faces, and 3-way has \( n^3 \) ||ism along the faces.
Example Functional Decomposition

- Determine image location
- Register image
- Determine image location
- Track position of instruments
- Display image
Types of parallelism
Types of parallelism

- Numerical algorithms often have data-parallelism
Types of parallelism

- Numerical algorithms often have data-parallelism
- Non-numerical algorithms often have functional parallelism.
Types of parallelism

- Numerical algorithms often have data-parallelism.
- Non-numerical algorithms often have functional parallelism.
- Many algorithms, especially complex numerical algorithms, have both, e.g., data parallelism within an function, many functions that can be done in parallel.
Types of parallelism

- Numerical algorithms often have data-parallelism.
- Non-numerical algorithms often have functional parallelism.
- Many algorithms, especially complex numerical algorithms, have both, e.g., data parallelism within an function, many functions that can be done in parallel.
- Functional parallelism often scales worse with increasing data size (concurrency-limited in isoefficiency terms)
Partitioning Checklist
Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
- Minimize redundant computations and redundant data storage
Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
- Minimize redundant computations and redundant data storage
- Primitive tasks roughly the same size
Partitioning Checklist

- At least 10x more primitive tasks than processors in target computer
- Minimize redundant computations and redundant data storage
- Primitive tasks roughly the same size
- Number of tasks an increasing function of problem size
Communication
Communication

- Determine values passed among tasks
Communication

- Determine values passed among tasks
- Local communication
Communication

- Determine values passed among tasks
- Local communication
  - Task needs values from a small number of other tasks
Communication

- Determine values passed among tasks
- Local communication
  - Task needs values from a small number of other tasks
  - Create channels illustrating data flow
Communication

- Determine values passed among tasks
- Local communication
  - Task needs values from a small number of other tasks
  - Create channels illustrating data flow
- Global communication
Communication

- Determine values passed among tasks
- Local communication
  - Task needs values from a small number of other tasks
  - Create channels illustrating data flow
- Global communication
  - Significant number of tasks contribute data to perform a computation
Communication

- Determine values passed among tasks
- Local communication
  - Task needs values from a small number of other tasks
  - Create channels illustrating data flow
- Global communication
  - Significant number of tasks contribute data to perform a computation
  - Don’t create channels for them early in design
Communication Checklist
Communication Checklist

- Communication operations balanced among tasks
Communication Checklist

- Communication operations balanced among tasks
- Each task communicates with only small group of neighbors
Communication Checklist

- Communication operations balanced among tasks
- Each task communicates with only small group of neighbors
- Tasks can perform communications concurrently
Communication Checklist

- Communication operations balanced among tasks
- Each task communicates with only small group of neighbors
- Tasks can perform communications concurrently
- Task can perform computations concurrently
Agglomeration
Agglomeration

- Grouping tasks into larger tasks
Agglomeration

- Grouping tasks into larger tasks
- Goals
Agglomeration

- Grouping tasks into larger tasks
- Goals
  - Improve performance
Agglomeration

- Grouping tasks into larger tasks
- Goals
  - Improve performance
  - Maintain scalability of program
Agglomeration

- Grouping tasks into larger tasks
- Goals
  - Improve performance
  - Maintain scalability of program
  - Simplify programming
Agglomeration

- Grouping tasks into larger tasks
- Goals
  - Improve performance
  - Maintain scalability of program
  - Simplify programming
- In MPI programming, goal often to create one agglomerated task per processor
Agglomeration Can Improve Performance
Agglomeration Can Improve Performance

- Eliminate communication between primitive tasks agglomerated into consolidated task
Agglomeration Can Improve Performance

- Eliminate communication between primitive tasks agglomerated into consolidated task
- Combine groups of sending and receiving tasks
Agglomeration Checklist
Agglomeration Checklist

- Locality of parallel algorithm has increased
Agglomerative Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn’t affect scalability
Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn’t affect scalability
- Agglomerated tasks have similar computational and communications costs
Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn’t affect scalability
- Agglomerated tasks have similar computational and communications costs
- Number of tasks increases with problem size
Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn’t affect scalability
- Agglomerated tasks have similar computational and communications costs
- Number of tasks increases with problem size
- Number of tasks suitable for likely target systems
Agglomeration Checklist

- Locality of parallel algorithm has increased
- Replicated computations take less time than communications they replace
- Data replication doesn’t affect scalability
- Agglomerated tasks have similar computational and communications costs
- Number of tasks increases with problem size
- Number of tasks suitable for likely target systems
- Tradeoff between agglomeration and code modifications costs is reasonable
Mapping
Mapping

- Process of assigning tasks to processors
Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system
Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user
Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user
- Conflicting goals of mapping
Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user
- Conflicting goals of mapping
  - Maximize processor utilization
Mapping

- Process of assigning tasks to processors
- Shared memory system: mapping done by operating system
- Distributed memory system: mapping done by user
- Conflicting goals of mapping
  - Maximize processor utilization
  - Minimize interprocessor communication
Mapping Example

While this may reduce communication, load balance may be an issue
Optimal Mapping
Optimal Mapping

- Finding optimal mapping is NP-hard
Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics
Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics
- Metis is a popular package for partitioning graphs
Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics
- Metis is a popular package for partitioning graphs
  - Minimizes the number of edges between nodes in a graph
Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics
- Metis is a popular package for partitioning graphs
  - Minimizes the number of edges between nodes in a graph
  - Edges, for our purposes, can be thought of as communication
Mapping Decision Tree
Mapping Decision Tree

- Static number of tasks
Mapping Decision Tree

- Static number of tasks
  - Structured communication
Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
      - Cyclically map tasks to processors
Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
      - Cyclically map tasks to processors
      - GSS (guided self scheduling)
Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
      - Cyclically map tasks to processors
      - GSS (guided self scheduling)
  - Unstructured communication
Mapping Decision Tree

- Static number of tasks
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
      - Cyclically map tasks to processors
      - GSS (guided self scheduling)
  - Unstructured communication
    - Use a static load balancing algorithm
Mapping Decision Tree

- **Static number of tasks**
  - Structured communication
    - Constant computation time per task
      - Agglomerate tasks to minimize communication
      - Create one task per processor
    - Variable computation time per task
      - Cyclically map tasks to processors
      - GSS (guided self scheduling)
  - Unstructured communication
    - Use a static load balancing algorithm

- **Dynamic number of tasks**
Mapping Strategy
Mapping Strategy

- Static number of tasks
Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks
Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks
  - Use a dynamic load balancing algorithm
Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks
    - Use a dynamic load balancing algorithm
  - Many short-lived tasks
Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks
    - Use a dynamic load balancing algorithm
  - Many short-lived tasks
    - Use a run-time task-scheduling algorithm
Mapping Strategy

- Static number of tasks
- Dynamic number of tasks
  - Frequent communications between tasks
    - Use a dynamic load balancing algorithm
  - Many short-lived tasks
    - Use a run-time task-scheduling algorithm
    - Cilk and Galois, discussed in the next couple of weeks, do this.
Mapping Checklist
Mapping Checklist

- Considered designs based on one task per processor and multiple tasks per processor
Mapping Checklist

- Considered designs based on one task per processor and multiple tasks per processor
- Evaluated static and dynamic task allocation
Mapping Checklist

- Considered designs based on one task per processor and multiple tasks per processor
- Evaluated static and dynamic task allocation
- If dynamic task allocation chosen, task allocator is not a bottleneck to performance
Mapping Checklist

- Considered designs based on one task per processor and multiple tasks per processor
- Evaluated static and dynamic task allocation
- If dynamic task allocation chosen, task allocator is not a bottleneck to performance
- If static task allocation chosen, ratio of tasks to processors is at least 10:1
Case Studies
Case Studies

- Boundary value problem
Case Studies

- Boundary value problem
- Finding the maximum
Case Studies

- Boundary value problem
- Finding the maximum
- The n-body problem
Case Studies

- Boundary value problem
- Finding the maximum
- The n-body problem
- Adding data input
Boundary Value Problem

- Ice water
- Rod
- Insulation
Rod Cools as Time Progresses
Want to use finite-difference method over multiple time steps
Want to use finite-difference method over multiple time steps

- Each circle represents a computation
Want to use finite-difference method over multiple time steps
Want to use finite-difference method over multiple time steps

Temperature at time \( t+1 \) for a position on the rod represented by a node depends on the temperature of neighbors at time \( t \).
Partitioning

time

position
Partitioning

- One data item per grid point
Partitioning

- One data item per grid point
- Associate one primitive task with each grid point
Partitioning

- One data item per grid point
- Associate one primitive task with each grid point
- Two-dimensional domain decomposition
Communication
Communication

- Identify communication pattern between primitive tasks

![Diagram of communication pattern with arrows indicating communication between nodes. The diagram shows a grid with nodes and arrows pointing to other nodes.]

Tuesday, April 14, 15
Communication

- Identify communication pattern between primitive tasks
- Each interior primitive task has three incoming and three outgoing channels
Agglomeration and Mapping
Agglomeration and Mapping

(a)

(b)

(c)

Agglomeration
Sequential execution time
Sequential execution time

- $\chi$ – time to update element
Sequential execution time

- $\chi$ – time to update element
- $n$ – number of elements
Sequential execution time

- $\chi$ – time to update element
- $n$ – number of elements
- $m$ – number of iterations
Sequential execution time

- $\chi$ – time to update element
- $n$ – number of elements
- $m$ – number of iterations
- Sequential execution time: $m (n-1) \chi$
Parallel Execution Time
Parallel Execution Time

- \( \chi \) – time to update element
Parallel Execution Time

- $\chi$ – time to update element
- $n$ – number of elements
Parallel Execution Time

- \( \chi \) – time to update element
- \( n \) – number of elements
- \( m \) – number of iterations
Parallel Execution Time

- $\chi$ – time to update element
- $n$ – number of elements
- $m$ – number of iterations
- $p$ – number of processors
Parallel Execution Time

- $\chi$ – time to update element
- $n$ – number of elements
- $m$ – number of iterations
- $p$ – number of processors
- $\lambda$ – message latency
Parallel Execution Time

- $\chi$ – time to update element
- $n$ – number of elements
- $m$ – number of iterations
- $p$ – number of processors
- $\lambda$ – message latency

Parallel execution time $m(\chi[(n-1)/p]+2\lambda)$
Finding the Maximum Error from measured data

<table>
<thead>
<tr>
<th></th>
<th>0.15</th>
<th>0.16</th>
<th>0.16</th>
<th>0.19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Correct</td>
<td>0.15</td>
<td>0.16</td>
<td>0.17</td>
<td>0.18</td>
</tr>
<tr>
<td>Error (%)</td>
<td>0.00%</td>
<td>0.00%</td>
<td>6.25%</td>
<td>5.26%</td>
</tr>
</tbody>
</table>

Need to do a reduction.

6.25%
Parallel Reduction Evolution

\( n - 1 \text{ tasks} \)
Parallel Reduction Evolution

\( n - 1 \) tasks
\( n/2 - 1 \) tasks
\( n/4 - 1 \) tasks
\( n/4 - 1 \) tasks

\( n/2 - 1 \) tasks

Tuesday, April 14, 15
Parallel Reduction Evolution

- $n - 1$ tasks
- $n/2 - 1$ tasks
- $n/4 - 1$ tasks

(a) $n/4 - 1$ tasks
(b) $n/4 - 1$ tasks
(c) $n/4 - 1$ tasks
Binomial Trees
Binomial Trees
Binomial Trees
Binomial Trees
Binomial Trees
Binomial Trees

Subgraph of hypercube
Finding Global Sum

4  2  0  7
-3  5 -6 -3
8  1  2  3
-4  4  6 -1
Finding Global Sum

\[
\begin{align*}
4 & \rightarrow -3 \\
2 & \rightarrow 5 \\
0 & \rightarrow -6 \\
7 & \rightarrow -3 \\
8 & \rightarrow 1 \\
1 & \rightarrow 2 \\
2 & \rightarrow 3 \\
3 & \rightarrow -1
\end{align*}
\]
Finding Global Sum

1  7  -6  4
4  5  8  2
Finding Global Sum

1 → 7 → -6 → 4

4 → 5 → 8 → 2
Finding Global Sum

8
-2
9
10
Finding Global Sum

\[ 8 + (-2) + 9 + 10 = 19 \]
Finding Global Sum

17 8

Tuesday, April 14, 15
Finding Global Sum

17 → 8
## Finding Global Sum

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>25</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total Sum: 25
Finding Global Sum

Binomial Tree

25
Agglomeration
Agglomeration leads to actual communication
Agglomeration leads to actual communication

Diagram showing the process of agglomeration with arrows indicating movement from one sum to another.
Agglomeration leads to actual communication
The n-body Problem
The n-body Problem
The n-body Problem
The n-body Problem
The $n$-body Problem
The n-body Problem
The n-body Problem
The n-body Problem
Partitioning
Partitioning

- Domain partitioning
Partitioning

- Domain partitioning
- Assume one task per particle
Partitioning

- Domain partitioning
- Assume one task per particle
- Task has particle’s position, velocity vector
Partitioning

- Domain partitioning
- Assume one task per particle
- Task has particle’s position, velocity vector
- Iteration
Partitioning

- Domain partitioning
- Assume one task per particle
- Task has particle’s position, velocity vector
- Iteration
  - Get positions of all other particles
Partitioning

- Domain partitioning
- Assume one task per particle
- Task has particle’s position, velocity vector
- Iteration
  - Get positions of all other particles
  - Compute new position, velocity
Gather
Gather
All-gather
All-gather
Complete Graph for All-gather -- operations shown, no ordering required
Complete Graph for All-gather -- operations shown, no ordering required
Hypercube-based All-gather -- ordering required
Complete Graph for All-gather
Communication Time

Complete graph

\[(p - 1)(\lambda + \frac{n/p}{\beta}) = (p - 1)\lambda + \frac{n(p - 1)}{\beta p}\]

Hypercube

\[\log p \sum_{i=1}^{\log p} \left( \lambda + \frac{2^{i-1} n}{\beta p} \right) = \lambda \log p + \frac{n(p - 1)}{\beta p}\]
Adding Data Input
Scatter
Scatter
Scatter in log $p$ Steps

in an application buffer

in a system buffer
Summary: Task/channel Model
Summary: Task/channel Model

- Parallel computation
Summary: Task/channel Model

- Parallel computation
  - Set of tasks
Summary: Task/channel Model

- Parallel computation
  - Set of tasks
  - Interactions through channels
Summary: Task/channel Model

- Parallel computation
  - Set of tasks
  - Interactions through channels
- Good designs
Summary: Task/channel Model

- Parallel computation
  - Set of tasks
  - Interactions through channels
- Good designs
  - Maximize local computations
Summary: Task/channel Model

- Parallel computation
  - Set of tasks
  - Interactions through channels

- Good designs
  - Maximize local computations
  - Minimize communications
Summary: Task/channel Model

- Parallel computation
  - Set of tasks
  - Interactions through channels
- Good designs
  - Maximize local computations
  - Minimize communications
  - Scale up
Summary: Design Steps
Summary: Design Steps

- Partition computation
Summary: Design Steps

- Partition computation
- Agglomerate tasks
Summary: Design Steps

- Partition computation
- Agglomerate tasks
- Map tasks to processors
Summary: Design Steps

- Partition computation
- Agglomerate tasks
- Map tasks to processors
- Goals
Summary: Design Steps

- Partition computation
- Agglomerate tasks
- Map tasks to processors
- Goals
  - Maximize processor utilization
Summary: Design Steps

- Partition computation
- Agglomerate tasks
- Map tasks to processors
- Goals
  - Maximize processor utilization
  - Minimize inter-processor communication
Summary: Fundamental Algorithms
Summary: Fundamental Algorithms

- Reduction
Summary: Fundamental Algorithms

- Reduction
- Gather and scatter
Summary: Fundamental Algorithms

- Reduction
- Gather and scatter
- All-gather