Parallel Algorithm Design
Outline

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

(a) Memory

(b) Task

Channel
Foster’s Design Methodology

- Partitioning
- Communication
- Agglomeration
- Mapping
Partitioning
- Computation/data are decomposed.
- No practical issues

Communication
- Communication structure is determined.

Agglomeration
- Evaluate the first two outcomes (performance-wise)
- Tasks may get combined.

Mapping
- Assign tasks to processors to
  - Maximize processor utilization
  - Minimize communication costs
Foster’s Methodology

- Problem
- Partitioning
- Communication
- Mapping
- Agglomeration
Partitioning

- Expose opportunities for parallel execution.
  - By dividing computation and data into pieces
- As fine-grained as possible.
  - Could be combined later.
- Works on both computation and data.
- Domain decomposition:
  - Focus on data first, then associate computation with data.
- Functional decomposition
  - Decompose computation first, then associate data with the computation.
Domain Decomposition

- Divide the data into small pieces of approximately equal size.
- Determine how to associate computations with the data.
- Partition the computation to be performed.
- Focus on largest data structure first.
- Widely used on spatial data structure
  - 1-D, 2-D, 3-D decomposition
Example Domain Decompositions
Functional Decomposition

- Divide the computation into disjoint tasks.
- Determine how to associate data with the computation
- Examine the data requirements of these tasks
  - If these data requirements are disjoint -> O.K.
Example Functional Decomposition
(interactive image-guided brain surgery)

- Acquire Patient Images
- Register Images
- Track Position of Instruments
- Determine Image Locations
- Display Image
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
- Consider several alternatives
Communication

- Determine values passed among tasks
- Easier for functional decomposition
  - Corresponds to data flow between tasks.
- Types
  - Local/global communication
  - Structured/unstructured
  - Static/dynamic
  - Synchronous/asynchronous
Communication

- **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 operations balanced among tasks
- Each task communicates with only small group of neighbors
- Tasks can perform communications concurrently
- Task can perform computations concurrently
Agglomeration (abstract -> concrete)

- The result so far is abstract that it’s not specialized for efficient execution on any particular computer.
- Would it be useful to combine tasks, replicate data and/or computation?
- Sometimes conflicting goals
  - Reducing communication costs by increasing computation and communication granularity
  - Retaining flexibility w.r.t scalability and mapping decisions
  - Reducing Software engineering costs
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 (abstract -> concrete)

- Reducing communication costs by increasing computation and communication granularity
  - Sending less data, fewer messages, create fewer tasks
  - Surface to volume effects
    - Large granule -> comm/computation ration small
  - Replicating computation
    - Summation and broadcast => summation by all
  - Avoiding communication
    - Combining tasks at different level which can’t execute concurrently
Agglomeration (abstract -> concrete)

- Retaining flexibility w.r.t scalability and mapping decisions
  - The ability to create a varying number of tasks is critical to be portable/scalable
  - Overlapping communication/computation
  - Balanced load over processors

- Reducing Software engineering costs
  - Not only to improve efficiency and flexibility, it is important to reduce development costs with parallelizing.
Agglomeration Can Improve Performance

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

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

(a)

(b)
Optimal Mapping

- Finding optimal mapping is NP-hard
- Must rely on heuristics
How to map?

Goal: minimize total execution time

Domain decomposition with fixed # of equal sized tasks and structural communication

- Efficient mapping is straight forward. (minimize IPC)

- Complex algorithms with variable amount of work per tasks
  - Use load balancing.

- Dynamic # of tasks/communication
  - Dynamic load balancing.
Recursive bisection (divide and conquer)

- Recursive coordinate bisection:
  - Irregular grids with mostly local communication.

- Unbalanced recursive bisection:
  - Optimize communication with better aspect ratio.

- Recursive graph bisection:
  - Use connectivity info to reduce communication

- Recursive spectral bisection
Load Balancing Algorithms (2)

- **Local algorithms can be used dynamically**
  - Inexpensive since no need of global info

- **Probabilistic methods**
  - Adv: low cost and scalable
  - Dis : off processor communication is needed

- **Cyclic mapping**
  - Computation load per grid point varies and significant spatial locality
  - Load balanced, but communication cost increases
Task scheduling Algorithms

- Manager/worker
- Hierarchical manager/worker
- Decentralized
  - Separate task pool is maintained on each processor,
  - Idle workers request problems from other processors.
- Termination detection (when to stop requesting?)
Mapping Decision Tree

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

- **Dynamic number of 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 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
Parallel Algorithm Design (examples)
Atmospheric model

\[ Nz = 15 - 30 \quad Ny = 50 - 500 \quad Nx = 2 \]

\[ Ny \]
The finite difference stencils

3 for vertical motion
9 for horizontal motion
partition

- $N_x \times N_y \times N_z$ tasks
- $O(1)$ data and computation per time step
Communication

Total mass computation

\[ Total\ Mass = \sum_{i=0}^{N_x-1} \sum_{j=0}^{N_y-1} \sum_{k=0}^{N_z-1} M_{ijk}, \]

Physics computation

\[ TCS_k = \prod_{i=1}^{k} (1 - cld_i) TCS_1 \]
\[ = TCS_{k-1}(1 - cld_k), \]
Agglomeration

- Reduce 8 messages to 4 messages
Mapping

- A simple mapping would be enough
Floorplan Optimization

- VLSI design problem
- 2 dimensional layout design
- Given components with their relative placement requirements and their alternative implementation, find an optimal layout.
Floorplan Optimization

(a) A  B0  B1  B2  C0  C1

(b) G: A → B → C

H: A → B → C
Floorplan Optimization
Floorplan Optimization
Branch and bound
partition

- No obvious data structure for decomposition
- Find grained functional decomposition
  - One task for each search node
- Breadth-first-search to generate tasks to run concurrently
- BFS might delay the discovery of $A_{\text{min}}$ which hampers pruning.
Communication

• In simple search, each task can work independently until a solution is found.
• In branch and bound, frequent communication is required to access $A_{\text{min}}$
• Centralized approach where a root is responsible for $A_{\text{min}}$
• Various refinement possible where $A_{\text{min}}$ is only accessed periodically, or use submanagers to communicate
After reaching a certain level, switch to DFS
Mapping

1. A central manager first constructs a number of coarse-grained tasks, by exploring the search tree to depth $D$, these tasks are then assigned to idle workers in a demand-driven manner.

2. Every worker expands the tree to depth $D$. Then, each worker takes responsibility for a disjoint subset of the tasks generated.

3. Allocate the root node to a single worker. Load balancing is then achieved by causing workers with empty queues to request problems from other workers. Each worker can then enforce a local DFS strategy.
Matrix multiplication

\[ C = A \times B \]

\[ C_{ij} = \sum_{k=0}^{n-1} A_{ik} \cdot B_{kj} \]

\[ O(n^3) \text{ operations} \]
partition

- 1-D decomposition

- 2-D decomposition
Communication (2-D case)

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{00}$</td>
<td>$A_{11}$</td>
<td>$A_{22}$</td>
<td>$A_{33}$</td>
<td></td>
</tr>
</tbody>
</table>

A submatrix is broadcast to the same row

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{00}$</td>
<td>$A_{00}$</td>
<td>$A_{00}$</td>
<td>$A_{00}$</td>
<td></td>
</tr>
<tr>
<td>$A_{11}$</td>
<td>$A_{11}$</td>
<td>$A_{11}$</td>
<td>$A_{11}$</td>
<td></td>
</tr>
<tr>
<td>$A_{22}$</td>
<td>$A_{22}$</td>
<td>$A_{22}$</td>
<td>$A_{22}$</td>
<td></td>
</tr>
<tr>
<td>$A_{33}$</td>
<td>$A_{33}$</td>
<td>$A_{33}$</td>
<td>$A_{33}$</td>
<td></td>
</tr>
</tbody>
</table>

Computation is performed

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{00}$</td>
<td>$B_{01}$</td>
<td>$B_{02}$</td>
<td>$B_{03}$</td>
<td></td>
</tr>
<tr>
<td>$B_{10}$</td>
<td>$B_{11}$</td>
<td>$B_{12}$</td>
<td>$B_{13}$</td>
<td></td>
</tr>
<tr>
<td>$B_{20}$</td>
<td>$B_{21}$</td>
<td>$B_{22}$</td>
<td>$B_{23}$</td>
<td></td>
</tr>
<tr>
<td>$B_{30}$</td>
<td>$B_{31}$</td>
<td>$B_{32}$</td>
<td>$B_{33}$</td>
<td></td>
</tr>
<tr>
<td>$B_{00}$</td>
<td>$B_{01}$</td>
<td>$B_{02}$</td>
<td>$B_{03}$</td>
<td></td>
</tr>
</tbody>
</table>

B submatrix is rotated upwards in the same column
Communication (systolic array)
Circuit Satisfiability

Not satisfied
Solution Method

- Circuit satisfiability is NP-complete
- No known algorithms to solve in polynomial time
- We seek all solutions
- We find through exhaustive search
- 16 inputs ⇒ 65,536 combinations to test
Partitioning: Functional Decomposition

- Embarrassingly parallel: No channels between tasks
Agglomeration and Mapping

- **Properties of parallel algorithm**
  - Fixed number of tasks
  - No communications between tasks
  - Time needed per task is variable

- **Consult mapping strategy decision tree**
  - Map tasks to processors in a cyclic fashion
Cyclic (interleaved) Allocation

- Assume \( p \) processes
- Each process gets every \( p^{th} \) piece of work
- Example: 5 processes and 12 pieces of work
  - \( P_0: 0, 5, 10 \)
  - \( P_1: 1, 6, 11 \)
  - \( P_2: 2, 7 \)
  - \( P_3: 3, 8 \)
  - \( P_4: 4, 9 \)
Pop Quiz

- Assume $n$ pieces of work, $p$ processes, and cyclic allocation.
- What is the most pieces of work any process has?
- What is the least pieces of work any process has?
- How many processes have the most pieces of work?
Summary of Program Design

- Program will consider all 65,536 combinations of 16 boolean inputs
- Combinations allocated in cyclic fashion to processes
- Each process examines each of its combinations
- If it finds a satisfiable combination, it will print it
Thank You!

| Q & A |