Lecture 11 - Patterns for Parallel Programming (III)
Credits

• Most of the slides courtesy Dr. Rodric Rabbah (IBM)
  - Taken from 6.189 IAP taught at MIT in 2007.
Patterns for Parallelizing Programs

4 Design Spaces

Algorithm Expression

• Finding Concurrency
  – Expose concurrent tasks

• Algorithm Structure
  – Map tasks to processes to exploit parallel architecture

Software Construction

• Supporting Structures
  – Code and data structuring patterns

• Implementation Mechanisms
  – Low level mechanisms used to write parallel programs

Code Supporting Structures

- Loop parallelism
- Master/Worker
- Fork/Join
- SPM D
- Map/Reduce
Loop Parallelism Pattern

- Many programs are expressed using iterative constructs
  - Programming models like OpenMP provide directives to automatically assign loop iteration to execution units
  - Especially good when code cannot be massively restructured

```c
#pragma omp parallel for
for(i = 0; i < 12; i++)
    C[i] = A[i] + B[i];
```
Master/Worker Pattern

Independent Tasks

master

worker
worker
worker
worker

A
B
C
D
E

A
B
C
D
E
Master/Worker Pattern

- Particularly relevant for problems using task parallelism pattern where tasks have no dependencies
  - Embarrassingly parallel problems

- Main challenge in determining when the entire problem is complete
Fork/Join Pattern

• Tasks are created dynamically
  - Tasks can create more tasks

• Manages tasks according to their relationship

• Parent task creates new tasks (fork) then waits until they complete (join) before continuing on with the computation
SPMD Pattern

• Single Program Multiple Data: create a single source-code image that runs on each processor
  - Initialize
  - Obtain a unique identifier
  - Run the same program each processor
    • Identifier and input data differentiate behavior
  - Distribute data
  - Finalize
**SPMD Challenges**

- Split data correctly
- Correctly combine the results
- Achieve an even distribution of the work
- For programs that need dynamic load balancing, an alternative pattern is more suitable
Map/Reduce Pattern

• Two phases in the program
• Map phase applies a single function to all data
  – Each result is a tuple of value and tag
• Reduce phase combines the results
  – The values of elements with the same tag are combined to a single value per tag -- reduction
  – Semantics of combining function are associative
  – Can be done in parallel
  – Can be pipelined with map

• Google uses this for all their parallel programs
Communication and Synchronization Patterns

- Communication
  - Point-to-point
  - Broadcast
  - Reduction
  - Multicast

- Synchronization
  - Locks (mutual exclusion)
  - Monitors (events)
  - Barriers (wait for all)
    - Split-phase barriers (separate signal and wait)
      - Sometimes called “fuzzy barriers”
    - Named barriers allow waiting on subset
Algorithm Structure and Organization (from the Book)

<table>
<thead>
<tr>
<th>Algorithm Type</th>
<th>Task parallelism</th>
<th>Divide and conquer</th>
<th>Geometric decomposition</th>
<th>Recursive data</th>
<th>Pipeline</th>
<th>Event-based coordination</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPMD</td>
<td>*****</td>
<td>***</td>
<td>*****</td>
<td>**</td>
<td>***</td>
<td>**</td>
</tr>
<tr>
<td>Loop Parallelism</td>
<td>*****</td>
<td>**</td>
<td>***</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Master/Worker</td>
<td>*****</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td>*****</td>
<td>*</td>
</tr>
<tr>
<td>Fork/Join</td>
<td>**</td>
<td>*****</td>
<td>**</td>
<td></td>
<td>*****</td>
<td>****</td>
</tr>
</tbody>
</table>

- Patterns can be hierarchically composed so that a program uses more than one pattern
Algorithm Structure and Organization (my view)

<table>
<thead>
<tr>
<th></th>
<th>Task parallelism</th>
<th>Divide and conquer</th>
<th>Geometric decomposition</th>
<th>Recursive data</th>
<th>Pipeline</th>
<th>Event-based coordination</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPMD</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Loop Parallelism</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Master/Worker</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fork/Join</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Patterns can be hierarchically composed so that a program uses more than one pattern.
## Algorithm Structure and Organization (my view)

<table>
<thead>
<tr>
<th></th>
<th>Task parallelism</th>
<th>Divide and conquer</th>
<th>Geometric decomposition</th>
<th>Recursive data</th>
<th>Pipeline</th>
<th>Event-based coordination</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SPMD</strong></td>
<td>*****</td>
<td>**</td>
<td>*****</td>
<td>**</td>
<td>*****</td>
<td>*</td>
</tr>
<tr>
<td><strong>Loop Parallelism</strong></td>
<td>***** when no dependencies</td>
<td>*</td>
<td>*****</td>
<td>*</td>
<td>*****</td>
<td>*</td>
</tr>
<tr>
<td><strong>Master/Worker</strong></td>
<td>*****</td>
<td>***</td>
<td>***</td>
<td>***</td>
<td>**</td>
<td>*****</td>
</tr>
<tr>
<td><strong>Fork/Join</strong></td>
<td>*****</td>
<td>****</td>
<td>**</td>
<td>****</td>
<td></td>
<td>*</td>
</tr>
</tbody>
</table>

- Patterns can be hierarchically composed so that a program uses more than one pattern.
Patterns for Parallelizing Programs

4 Design Spaces

**Algorithm Expression**
- Finding Concurrency
  - Expose concurrent tasks

**Algorithm Structure**
- Map tasks to processes to exploit parallel architecture

**Software Construction**
- Supporting Structures
  - Code and data structuring patterns

- Implementation Mechanisms
  - Low level mechanisms used to write parallel programs

ILP, DLP, and TLP in SW and HW

- **ILP**
  - OOO
  - Dataflow
  - VLIW

- **DLP**
  - SIMD
  - Vector

- **TLP**
  - Essentially multiple cores with multiple sequencers

- **ILP**
  - Within straight-line code

- **DLP**
  - Parallel loops
  - Tasks operating on disjoint data
    - No dependencies within parallelism phase

- **TLP**
  - All of DLP +
  - Producer-consumer chains
## ILP, DLP, and TLP and Supporting Patterns

<table>
<thead>
<tr>
<th></th>
<th>Task parallelism</th>
<th>Divide and conquer</th>
<th>Geometric decomposition</th>
<th>Recursive data</th>
<th>Pipeline</th>
<th>Event-based coordination</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>DLP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>TLP</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## ILP, DLP, and TLP and Supporting Patterns

<table>
<thead>
<tr>
<th></th>
<th>Task parallelism</th>
<th>Divide and conquer</th>
<th>Geometric decomposition</th>
<th>Recursive data</th>
<th>Pipeline</th>
<th>Event-based coordination</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILP</strong></td>
<td>inline / unroll</td>
<td>inline</td>
<td>unroll</td>
<td>inline</td>
<td>inline / unroll</td>
<td>inline</td>
</tr>
<tr>
<td><strong>DLP</strong></td>
<td>natural or local-conditions</td>
<td>after enough divisions</td>
<td>natural</td>
<td>after enough branches</td>
<td>difficult</td>
<td>local-conditions</td>
</tr>
<tr>
<td><strong>TLP</strong></td>
<td>natural</td>
<td>natural</td>
<td>natural</td>
<td>natural</td>
<td>natural</td>
<td>natural</td>
</tr>
</tbody>
</table>
## ILP, DLP, and TLP and Implementation Patterns

<table>
<thead>
<tr>
<th></th>
<th>SPMD</th>
<th>Loop Parallelism</th>
<th>Mater/Worker</th>
<th>Fork/Join</th>
</tr>
</thead>
<tbody>
<tr>
<td>ILP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DLP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **SPMD**
- **Loop Parallelism**
- **Mater/Worker**
- **Fork/Join**

After enough divisions, use the local-conditional approach to improve performance.

**Fork/Join**

**Mater/Worker**
## ILP, DLP, and TLP and Implementation Patterns

<table>
<thead>
<tr>
<th></th>
<th>SPMD</th>
<th>Loop Parallelism</th>
<th>Master/Worker</th>
<th>Fork/Join</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ILP</strong></td>
<td>pipeline</td>
<td>unroll</td>
<td>inline</td>
<td>inline</td>
</tr>
<tr>
<td><strong>DLP</strong></td>
<td>natural or local-conditional</td>
<td>natural</td>
<td>local-conditional</td>
<td>after enough divisions + local-conditional</td>
</tr>
<tr>
<td><strong>TLP</strong></td>
<td>natural</td>
<td>natural</td>
<td>natural</td>
<td>natural</td>
</tr>
</tbody>
</table>