Parallel Programming Patterns

Eun-Gyu Kim
June 10, 2004
Overview

• Definition
• Algorithm Patterns Example
• Problems / Weakness
• More Examples
• Directions
Defining Patterns

- **Design Patterns** - Quality description of problem and solution to a frequently occurring problem in some domain.
  - (object oriented programming, network apps...)
  - Description usually involves problem definition, driving forces, solution, benefits, difficulties, related patterns.

- **Pattern Language** - a collection of design patterns, guiding the users through the decision process in building a system.
“Lunch” Pattern

**Definition**
- we get hungry every lunch hour.
- usually around 11:30am – 1:00pm.
- must resolve hunger.

**Driving Forces**
- appetite, intensity of hunger, nearby restaurants

**Solution**
- eat at nearest restaurant that satisfies minimum appetite requirement.

**Benefits**
- hunger is resolved

**Difficulties**
- must not fall asleep afterwards.

**Related Patterns**
- dinner, breakfast, brunch, and snack

“Eat” Pattern Language

- Wake-Up
- Breakfast
- Lunch
- Dinner
- Brunch

Example Design Space
Parallel Programming Pattern: WHY?

• Parallel software does not fully exploit parallel hardware.
  – Too difficult for programmers
  – Parallel Programming Environments do not focus on design issues.

• Need a “cookbook” that will guide the programmers systematically to achieve peak parallel performance.
  – (decomposition, algorithm, program structure, programming environment, optimizations)

• Provide common vocabulary to the programming community.

• Software reusability & modularity.
Some approaches

- Foster Methodology
  - Decomposition
  - Communication
  - Agglomeration
  - Mapping

- Massingil
  - FindingConcurrency
    - data vs. control
  - AlgorithmStructure
    - pipeline, replicate…
  - SupportingStructure
    - SPMD, fork/join…
  - Implementation
    - barriers, locks…
Common Parallel Patterns

- Embarassingly Parallel
- Replicable
- Repository
- Divide&Conquer
- Pipeline
- Recursive Data
- Geometric
- IrregularMesh
- Inseparable
Embarassingly Parallel

Problem: Need to perform same operations to tasks that are independent
Replicable

Sets of operations need to be performed using global data structure, causing dependency.

Solution

Operation I

Operation II

Operation III

Global Data

replicate

reduce
Repository

Independent computations needs to be applied to a centralized data structure in a non-deterministic way.

- Compute A
- Compute B
- Compute E
- Compute D
- Compute C

- Asynchronous Function calls

Compute object cannot access the same element simultaneously. (Repository controls access)
Divide & Conquer

A problem is structured to be solved in sub-problems independently, and merging them later.

* Split level needs to be adjusted appropriately.
Pipeline

A series of ordered but independent computation stages need to be applied on data, where each output of a computation becomes input of subsequent computation.

Time

Stage 1
- C1
- C2
- C3
- C4
- C5
- C6

Stage 2
- C1
- C2
- C3
- C4
- C5
- C6

Stage 3
- C1
- C2
- C3
- C4
- C5
- C6

Stage 4
- C1
- C2
- C3
- C4
- C5
- C6
Recursive Data

Recursive data structures seem to have little exploitable concurrency. But in some cases, the structure can be transformed.

Find Root Problem

Step 1

Step 2

Step 3
Geometric

Dependencies exist, but communicate in predictable (geometric) neighbor-to-neighbor paths.

Neighbor-To-Neighbor communication
Irregular Mesh

Communication in non-predictable paths in mesh topology.

Hard to define due to varying communication patterns.

Start point:
Pattern that constructed this mesh.
Inseparable

• **All other** parallel patterns fails, and need explicit protection when dependent data elements are accessed.
  – i.e. mutual exclusion, producer-consumer.

• **Very vague definition.**
They all start from

- Decomposing a sequential problem to expose concurrency.
- Data decomposition
  - Concurrency comes from working on different data elements simultaneously.
- Functional decomposition
  - Concurrency comes from working on independent functional tasks simultaneously.
It’s Parallel Programming 101

• Why can’t we advance from here?
• Almost all of the patterns discussed are either intuitive, or covered in introductory courses.
• Do these patterns capture most of parallel programs today?
• Can these patterns be used to exploit full parallelism?
Why so trivial?

• Only two ways to decompose:
  – Data and Functional.

• Over-simplified just for
  – Intuitiveness & “natural representation”
  – Fitting the problem to “imperative language”

• By defining functional task:
  – We have chosen to reuse a variable.
  – It’s a conscious choice, and brings new constraint.
  – Removing this does not change the problem definition.
**Same Effect**

In many cases, different decomposition leads to same computation.

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

$\times$

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

=  

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

**Owner-Compute**

Implies / Equivalent to

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

$\times$

<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>
</tbody>
</table>

=  

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

**Row-major Decomposition**
Exposing Better Parallelism

• Loosen the "natural representation".
• Free the assumptions
  – Variable reuse
  – Thinking in terms of imperative language
Non-Trivial Parallelism I: Molecular Dynamics

Repeated data of molecules can be processed better in periodic domain. (but little bit harder to represent)

*Decomposition is done in two domains. Spatial + Periodic
Non-Trivial Parallelism II: Climate Modeling

Physics equation called “Spectral Transform Method” needs spectral domain to work on.

\[
\psi(\lambda, \mu) = \sum_{m=-M}^{M} \sum_{n=|m|}^{N(m)} \psi_n^m P_n^m(\mu)e^{im\lambda},
\]

Spectral Domain
Direction

• Capture fundamental decomposition in
  – Systematic
  – Constraint-Free Ways.
• Document more non-trivial patterns
• Map patterns to programming environments
• Structural Patterns
Any Questions?