Parallel Computing Patterns for Grid Workflows

Cesare Pautasso, Gustavo Alonso
Department of Computer Science, ETH Zurich, Switzerland
{pautasso, alonso}@inf.ethz.ch – www.jopera.org
Grid Workflows for Large Scale eScience

How well can we do Parallel Computing with Workflows?

Service Oriented Grid

Large Scale eScience

Grid Workflows
Why Parallel Computing Patterns?

- Language primitives for modeling parallelism
  - Common classification
  - Unify different syntax/notations
  - Test of expressive power
- Efficient implementation for Grid workflows
  - Do all systems support all patterns?
  - What is the semantics of parallelism?
  - Impact on scheduling, data management, lineage tracking features
Overview

• Parallel Execution
  • Simple Parallelism
  • Data Parallelism
• Pipelined Execution
  • Best Effort
  • Blocking
  • Buffered
  • Superscalar
  • Streaming
Parallel Execution: Simple Parallelism

- Parallel split (Classical Control Flow Pattern)
- Independent tasks...
  - ...run in parallel (*strong semantics*)
  - ...may run in parallel if enough resources are available (*realistic implementation*)
  - ...are serialized non deterministically (*weak semantics*)
- Modeling:
  - Explicit or Implicit
  - Control flow or Data flow
  - Graph based or Block based (or both)
Modeling Simple Parallelism

- Data Flow, Graph Based, Implicit

Examples:
- SCIRun
- Kepler
- Triana
Modeling Simple Parallelism

- Control Flow, Graph Based

Example:
- YAWL
- JOpera
- GEL

Example:
- UML
Modeling Simple Parallelism

- Control Flow, Block Based, Explicit

Example:
- BPMN
- BPEL4WS
Parallel Execution: Data Parallelism

• SPMD: Run a copy of the same task over multiple data elements (in parallel)

• How to control the amount of parallelism?
  • Static (Design-time) vs. Dynamic (Run-time)
  • Manual vs. Adaptive
  • Homogeneous vs. Heterogeneous partitions

• Modeling
  • Data Flow or Control Flow
  • Graph Rewriting, Block based
  • First-Order Functions (Map)
Modeling Data Parallelism

- Data Flow, Graph Rewriting

Examples:
- Triana
- Taverna
- JOpera

- Static or Dynamic
Modeling Data Parallelism

- Data Flow, First-Order Functions

Example: Kepler
Modeling Data Parallelism

- Control Flow, Graph Based

$\ll ParallelLoop \gg$

$T^*$

$Next$

$T$

Examples:
- Teuta
- UML
Modeling Data Parallelism

- Control Flow, Block Based

Examples:
- WS-BPEL
- AGWL
- Karajan
- GEL
Overview

• Parallel Execution
  • Simple Parallelism
  • Data Parallelism

• Pipelined Execution
  • Best Effort
  • Blocking
  • Buffered
  • Superscalar
  • Streaming
Parallel Execution: Pipelined Execution

- Stream multiple data elements sequentially through a sequence of tasks
Modeling Pipelined Execution

• Syntax very similar, but semantics changes a lot!

• How to deal with non uniform task duration?
  • Best Effort
  • Blocking
  • Buffering
  • Superscalar
  • Streaming
Best Effort Pipelined Execution

- Drop data elements on pipeline collisions
- Advantages:
  - Simplified implementation
  - Some applications may tolerate data loss
- Problem:
  - Downsampling is non deterministic
Blocking Pipelined Execution

- Tasks are blocked if successors are busy
- Advantages:
  - Avoid data loss in the pipeline
- Problem:
  - Pipeline speed limited by slowest task
  - Data may be lost before it enters the pipeline
Buffered Pipelined Execution

- Tasks are decoupled by buffers

**Advantages:**
- Collisions are prevented
- Best applied to tasks having variable speed

**Problem:**
- Buffer capacity is limited (Blocking still needed)
Superscalar Pipelined Execution

- If a task is busy, create another instance

- Advantage:
  - Data loss avoided without blocking

- Problem:
  - Data elements may overtake one another
  - Where to enforce synchronization?
Streaming Pipelined Execution

- Tasks exchange data while running

Advantages:
- Suitable for a distributed (P2P) engine

Problems:
- Shifts complexity from the workflow engine to the tasks
- Tasks exchange data while running
- Workflow/Task interface more complex
Conclusions

- Applying parallel computing techniques to Grid workflows has become a necessity for large scale eScience applications.

- Not all Grid workflow languages/systems we surveyed support all patterns:
  - Simple Parallelism & Static Data Parallelism supported by all
  - Dynamic Data Parallelism still a challenge (for some)
  - Pipelining implemented with many different semantics

- Let us know how your Grid workflow language/tool supports these patterns!
Parallel Computing Patterns for Grid Workflows

Cesare Pautasso, Gustavo Alonso
Department of Computer Science, ETH Zurich, Switzerland
pautasso@inf.ethz.ch – www.jopera.org