A Reconfigurable Component Model for HPC

Vincent Lanore\textsuperscript{1}, Christian Pérez\textsuperscript{2}

\textsuperscript{1} : ENS de Lyon, \textsuperscript{2} : Inria
LIP laboratory, Avalon Team
France
High-Performance Computing

Goal: run the biggest possible applications

- eg, large simulations
- months/years of sequential computing time

using cutting edge hardware

- very parallel

Challenge: scalability

Tianhe-2
3,120,000 cores
source: top500.org

GyseLA 5D
years of sequential computing time
source: master omp, Paris13

A scientific mesh-based simulation
source: NTUA, school of mechanical engineering
**HPC Component Models**

**Examples: CCA, L2C**

**Typically:**
- low-level
  - C++/FORTRAN-level abstractions
- non-hierarchical
- distributed
  - eg, message passing, remote method call
- process abstraction

A jacobi solver assembly on 4 processes

MPI = Message Passing Interface
Problem: Dynamic HPC Applications

Applications with...
- dynamic communication topology
- dynamic data structure

Not supported by HPC component models
- reconfiguration needed

Examples
- Adaptive Mesh Refinement (AMR)
- dynamic load balancing

Goal of this talk:
HPC reconfigurable component model
Plan of the Talk

Context and related works
- Related work
- Our proposition

Presentation of the model, DirectMOD
- Assembly model
- Programming model

Implementation and evaluation
- Ring example
- DirectL2C
- Code and performance

Conclusions and perspectives
Reconfigurable Component Models

From the literature:

<table>
<thead>
<tr>
<th>No reconfiguration support</th>
<th>Examples</th>
<th>Locking and representation</th>
<th>Scalable?</th>
<th>Reconf SE properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>CCA, L2C</td>
<td>none</td>
<td>up to the user</td>
<td>poor</td>
<td></td>
</tr>
<tr>
<td>Global reconfiguration</td>
<td>global MAPE</td>
<td>global</td>
<td>no</td>
<td>good</td>
</tr>
<tr>
<td>Composite-level controllers</td>
<td>Fractal, SOFA</td>
<td>composite-level</td>
<td>sometimes</td>
<td>sometimes</td>
</tr>
</tbody>
</table>

Important parameters

- locking granularity
- scalability
- representation granularity
- ease of use

No model provides both

- scalable approach
- good SE properties for reconfigurable assemblies
- reuse
- separation of concerns
Our proposal: main ideas

Let users define locking units
- custom granularity / distribution → performance

<table>
<thead>
<tr>
<th></th>
<th>Locking and representation</th>
<th>Scalable?</th>
<th>Reconf SE properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-based</td>
<td>user-defined (domains)</td>
<td>yes</td>
<td>???</td>
</tr>
</tbody>
</table>

To improve SE properties
- separate locking from representation and transformation

<table>
<thead>
<tr>
<th></th>
<th>Locking and representation</th>
<th>Scalable?</th>
<th>Reconf SE properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-based + separation lock/transfo</td>
<td>user-defined + separated</td>
<td>yes</td>
<td>good</td>
</tr>
</tbody>
</table>
A Formal Model

DirectMOD: formal model
- full syntax
- transformation semantics

In addition:
- resource model (see paper)
- call stack operational semantics (see paper)

Benefits
- unambiguous specification
- tech-agnostic
- runtime representation
DirectMOD Assembly Model

Elements
- components
- ports

Relations
- point-to-point references
- owner (component-port relation)
DirectMOD Domains

New element: domains

- manage a subassembly
- unit of locking
- unit of internal representation
- reconfigure their contents
DirectMOD Domains

New element: domains

- manage a subassembly
- unit of locking
- unit of internal representation
- reconfigure their contents
DirectMOD Transformations

origin → transformation → topology

destination

application

Monitor
Compute

Monitor
Compute

Monitor
Compute

Compute

A Reconfigurable Component Model for HPC

Montréal

CBSE'15
DirectMOD Transformation Adapters

- special kind of port
- reference to transformation and application subassembly
DirectMOD Programming Model

- Locking/synchrone specialist
- Transformation programmer
- Component programmer

- Locking algorithms
- Assembly transformations
- Components

End user

Reconfigurable assembly
A C++/MPI Implementation

DirectL2C
- DirectMOD implementation
- extension of L2C
- uses traditional HPC tech
  - C++
  - MPI (Message Passing Interface)
  - threads

Provides
- remote basic reconfiguration operations
- transformation parsing and execution
- helper classes for locking
- interface and locking APIs

<table>
<thead>
<tr>
<th></th>
<th>Components</th>
<th>Files</th>
<th>mLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2C</td>
<td>37</td>
<td>79</td>
<td>4570</td>
</tr>
<tr>
<td>DirectL2C</td>
<td>3</td>
<td>10</td>
<td>1118</td>
</tr>
</tbody>
</table>
DirectL2C in Practice

Component

- fonctionnal code (C++, helper class)
- interface declaration (L2C API + extension)
- locking instrumentation (conforming to locking API)

Domain

- locking code (C++, using locking API)

Transformation

- transformation description (high level API)

Starting assembly

- assembly description (L2C assembly descriptor)

DirectL2C launcher
Evaluation: Easy to Write?

Implemented: ring assembly
- insert new component
- remove component

Preliminary implementation
- not yet fully optimized
- shortest possible code

Ring assembly LOC

<table>
<thead>
<tr>
<th>Function</th>
<th>C++ LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformation</td>
<td>8</td>
</tr>
<tr>
<td>Non-functional sync</td>
<td>20</td>
</tr>
<tr>
<td>Code instrumentation</td>
<td>13</td>
</tr>
<tr>
<td>L2C overhead</td>
<td>7</td>
</tr>
<tr>
<td>DirectL2C overhead</td>
<td>6</td>
</tr>
<tr>
<td>Functional code</td>
<td>31</td>
</tr>
<tr>
<td>Other</td>
<td>3</td>
</tr>
<tr>
<td><strong>TOTAL</strong></td>
<td><strong>88</strong></td>
</tr>
</tbody>
</table>

- short and easy transformation code
- large and difficult synchronization code
Evaluation: scalable?

Testing our ring assembly

- one component per core at startup
- fixed number of *insert* and *remove* transformations per starting component
- one domain per component (fully distributed)

Preliminary experiments on Grid'5000

- acceptable scalability up to 128 cores
Conclusion and Perspectives

Presented DirectMOD

- a reconfigurable component model
- formal
- +implementation (DirectL2C)
- preliminary evaluation
  - short and easy to write code
  - scalability up to 128 cores

Perspectives

- improve evaluation
  - ongoing work on a complex benchmark
  - experiments on large platform (eg, Curie)
- ease domain development
  - ongoing work on automating certain locking patterns
- transformation specification
  - genericity
  - compact language