A Reconfigurable Component Model for HPC

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

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

Goal: run the biggest possible applications
- from astrophysics, meteorology, industry...
- up to millions of years of sequential computing time

using cutting edge hardware
- very parallel

Challenge: scalability

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

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

Examples:
- CCA
- L2C (Avalon team)

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

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

Not supported by HPC component models
- reconfiguration needed

Goal of this talk:
HPC reconfigurable component model
reconfigurable
scalable (distributed, concurrent reconfiguration)
good SE properties (reuse, separation of concerns)
Plan of the Talk

Context and related works
- Related work
- Our proposition

Presentation of the model, DirectMOD
- Model elements
- Putting it all together

Implementation: DirectL2C
- Code
- Performance

Extensions of DirectMOD
- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives
Plan of the Talk

Context and related works
- Related work
- Our proposition

Presentation of the model, DirectMOD
- Model elements
- Putting it all together

Implementation: DirectL2C
- Code
- Performance

Extensions of DirectMOD
- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives
Reconfigurable Component Models

From the literature:

<table>
<thead>
<tr>
<th></th>
<th>Examples</th>
<th>Locking and representation</th>
<th>Scalable?</th>
<th>Reconf SE properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>No reconfiguration support</td>
<td>CCA, L2C</td>
<td>none</td>
<td>up to the user</td>
<td>poor</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

A Reconfigurable Component Model for HPC
Our proposal: main ideas

Let users define locking units (*domains*)
- custom granularity / distribution → performance

Separate locking units from transformations
- transformation on standalone representation
- assembly-level explicit mapping to domain contents (using *transformation adapters*)

<table>
<thead>
<tr>
<th></th>
<th>Locking units</th>
<th>Representation for transformations</th>
<th>Scalable?</th>
<th>Reconf SE properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain-based</td>
<td>user-defined</td>
<td></td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Transformation adapters</td>
<td></td>
<td>standalone</td>
<td></td>
<td>good</td>
</tr>
</tbody>
</table>
Plan of the Talk

Context and related works
- Related work
- Our proposition

Presentation of the model, DirectMOD
- Model elements
- Putting it all together

Implementation: DirectL2C
- Code
- Performance

Extensions of DirectMOD
- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives
A Formal Model

DirectMOD:
- formal model
  - full syntax
  - transformation semantics

Benefits
- unambiguous specification
- tech-agnostic
- runtime representation

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

A Reconfigurable Component Model for HPC
DirectMOD Transformations
DirectMOD Transformations

- **Insert**
- **Replace**

A Reconfigurable Component Model for HPC

STAARS
DirectMOD Transformations

Insert

Monitor

Compute

state

Monitor

Compute

Monitor

Compute

Compute

Compute

A Reconfigurable Component Model for HPC
DirectMOD Transformations

A Reconfigurable Component Model for HPC
DirectMOD Transformation Adapters

- special kind of port
- reference to transformation and application subassembly
Putting Everything Together

A Reconfigurable Component Model for HPC

STAARS
DirectMOD Programming Model

- Locking/synchro specialist
  - Locking algorithms

- Transformation programmers
  - Assembly transformations

- Component programmers
  - Components

End user

Reconfigurable assembly
Plan of the Talk

Context and related works
- Related work
- Our proposition

Presentation of the model, DirectMOD
- Model elements
- Putting it all together

Implementation: DirectL2C
- Code
- Performance

Extensions of DirectMOD
- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives
A C++/MPI Implementation

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

**L2C provides**
- C++ components with zero overhead
- basic component operations
- assembly deployment
  - distributed

**DirectL2C provides**
- transformation parsing and execution
- 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>
### Evaluation: Easy to Write?

**Implemented: ring assembly**
- insert new component
- remove component

**Preliminary implementation**
- not yet fully optimized
- shortest possible code

<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
- complex synchronization code
Transformation Pseudocode

*Inputdata:* portName, reconfPortName, resourceName

Direct::transformation insert();
insert.create("Cp", "newCp", resourceName);
//ports of created component have implicit names
insert.connect(portName, "newCpLeft");
insert.statePort(portName, "newCpRight");
insert.topoPort(reconfPortName, "newCpReconf");
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 clusters
- acceptable scalability up to 128 cores
Plan of the Talk

Context and related works
- Related work
- Our proposition

Presentation of the model, DirectMOD
- Model elements
- Putting it all together

Implementation: DirectL2C
- Code
- Performance

Extensions of DirectMOD
- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives
Some questions left unanswered

- **what about locking domains?**
  - looks difficult
  - poor reuse

- **what about transformations?**
  - how to write them? language?
  - what about genericity?
  - ... hierarchy?
Stopping Components

This assembly is running...  
... how to stop it?

- no control info
- performance?
- deadlocks?
- call stacks

STAARS

A Reconfigurable Component Model for HPC
Locking use ports

**Approach: locking use ports**

- no more calls once locked
- mutex-like behaviour

- in what order?
- deadlocks?
Adding metadata

First step: add metadata

- minimal
- call dependencies
- termination dependencies
Second Step: Locking Order Algorithm

Input:

a) A → B → C → D → E → F

b) A → C → E
   B → D → F

Output: F, A, C, B, D, E
It Works for Stencil Applications

- **Stencil applications**
  - Simple control
  - Concurrent locking of connections

- **Useful automation: connection variants**
  - Local, distributed
  - Proxies
  - 1-to-n connections
1/2/3D grid benchmark w/ locking only
versus component-wise approach
High-level transformation language(s)

- **High-level languages**
  - hierarchy
  - genericity
  - decent compact language for humans (ie, not XML)

- **Transformation to low-level**
  - off-line process
  - optimization during transformation
High-level transformation language(s)

A0

\[ V(A_0, \Sigma) = \{ \text{im.}(\text{Master-Worker}) \} \]

A1

\[ V(A_1, \Sigma) = \{ \]
\[ \text{sp.}(\text{Master, Master A}) \]
\[ \text{sp.}(\text{Master, Master B}) \]
\[ \text{sp.}(n \text{ workers, 2 workers}) \]
\[ \text{sp.}(n \text{ workers, 1 worker}) \]
\[ \text{sp.}(n \text{ workers, 2 identical workers}) \]
\[ \text{sp.}(n \text{ provides, 2 provides}) \]
\[ \text{sp.}(n \text{ provides, 1 provide}) \]
\[ \text{sp.}(1\text{-to-n use/provide, 1\text{-to-2 use/provide})} \]
\[ \text{sp.}(1\text{-to-n use/provide, use/provide}) \]
\[ \} \]

A2

\[ V(A_2, \Sigma) = \{ \]
\[ \text{sp.}(\text{Master, Master A}) \]
\[ \text{sp.}(\text{Master, Master B}) \]
\[ \text{im.}(2 \text{ workers}) \]
\[ \} \]

A3

\[ V(A_3, \Sigma) = \{ \]
\[ \text{sp.}(\text{Master, Master A}) \]
\[ \text{sp.}(\text{Master, Master B}) \]
\[ \text{sp.}(\text{Worker, Worker A}) \]
\[ \text{sp.}(\text{Worker, Worker B}) \]
\[ \]
High-level transformation language(s)

V(A4,Σ) = {
sp(Master, Master A)
sp(Master, Master B)
sp(Worker, Worker A)
sp(Worker, Worker B)
}\(\text{im.(1-to-2 use/provide)}\)
Plan of the Talk

Context and related works
- Related work
- Our proposition

Presentation of the model, DirectMOD
- Model elements
- Putting it all together

Implementation: DirectL2C
- Code
- Performance

Extensions of DirectMOD
- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives
Conclusion and Perspectives

Presented DirectMOD

- a reconfigurable component model
- formal
- distributed
- + implementation (DirectL2C)
- + efficient locking for stencil applications

Perspectives

- improve evaluation & implementation
- ongoing work on a complex benchmark
- experiments on a large platform (DARI allowance)
- + work on directL2C
- transformation specification
- genericity
- compact language