A Low Level Component Model enabling Performance Portability of HPC Applications

Julien Bigot (INRIA/CEA)
Christian Perez (INRIA)
Vincent Pichon (EDF R&D)
Zhengxhion Hou (INRIA)
Avalon Research Team, LIP, Lyon, France

Context

Scientific Applications

- Cluster (GPU/MC/…)
- Grids (EGEE)
- Super-computer (Exascale)
- IaaS (Cloud)
Programming a parallel applications

- (High level) parallel languages
  - HPF, PGAS, ...
  - Not yet mature

- Platform oriented models
  - Multi-core ↔ Threads, OpenMP
  - GPU ↔ Cuda, OpenCL, OpenAPP
  - Multi-node ↔ MPI
  - Many versions of the same code
  - Difficult to maintain all versions synchronized
  - Difficult to keep specific machine optimizations
  - Low code reuse
Proposed Approach Overview

- Separation of concerns
  - Machine specific code from re-usable code
- Make explicit points of configuration
  - Need a configurable representation of an application
- Generate machine specific version
  - Need a process

- Component model as an application description model to adapt to a particular machine
Content

- Context
- Overview of component models
- L2C: Low Level Component
- Jacobi & L2C
- Experimental evaluation
- Conclusion & Perspectives
Software Component

- Technology that advocates for composition
  - Old idea (late 60’s)
  - Assembling rather than developing

- Many types of composition operator
  - Spatial, temporal, ….

- Assembly of component
  - Primitive & composite components

- Many models
  - CCA, Salome, CCM, Fractal, GCM, OGSi, SCA, …
CCA Example

Dashed lines indicate alternate connections

Create different applications in "plug-and-play" fashion

From CCA Tutorial, http://www.cca-forum.org/tutorials/
Limitation of Existing HPC Component Model

- Pre-defined set of interactions
  - Usually function/method invocation oriented
  - How to incorporate other interactions, eg MPI?
- Provide communication abstraction
  - Language interoperability (~IDL)
  - Network transparency
  - Potential overhead when not needed
  - Limited data types systems
    - Babel SIDL, OMG IDL, …
- Programming model vs execution model
Content

- Context
- Overview of component models
- L2C: Low Level Component
- Jacobi & L2C
- Experimental evaluation
- Conclusion & Perspectives
Low Level Component Model

- A minimalist component model for HPC
  - Component creation/deletion & connection
  - An (optional) launcher
- No L2C code between component @ runtime
- Support native interactions
  - C++, MPI, CORBA, (FORTRAN soon)
- Extensible
- LGPL, available at hlcm.gforge.inria.fr
A Minimal L2C Component

- A component is a class
  - HelloWorld
- A few macros make it a component
  - LCMP … LEND
- No library dependency (only header)

```cpp
class HelloWorld
{
public:
    HelloWorld()
    {
        cerr << "HelloWorld created" << endl;
    }

    ~HelloWorld()
    {
        cerr << "HelloWorld destroyed" << endl;
    }
};

LCMP(HelloWorld)
LEND
```
A Minimal L2C Assembly

- Assembly
  - Simple XML file
  - Set of instance

- Instance
  - Name (id)
  - Type (class name)

```
<lad>
  <instance id="hw" type="HelloWorld"/>
</lad>
```
Configuration properties

- A type
  - string, int, complex, double, ...
- A name
- Implementation
  - Public member variable
  - L_PROPERTY macro
- Value set in Assembly
C++ Interactions between components

- Method call interactions
  - Typed by an “interface”
- Provide the service
  - Inheritance
  - L_CPP_PROVIDE macro
- Use the service
  - Pointer import
  - L_CPP_USE macro
C++ Interactions in the Assembly

- Connection
  - Set USE property
  - Reference PROVIDE property

```xml
<lad>
  <instance id="serv" type="Server" />
  <instance id="client" type="Client" >
    <property id="greetservice"><propref instance="serv" property="greeter"/></property>
    <start property="go"/>
  </instance>
</lad>
```
class MpiClientProxy :
    virtual public Hello
{
    public:
        MPI_Comm comm;

        virtual void greet (string name)
        {
            unsigned long size = name.size()+1;
            MPI_Send(&size, 1, MPI_UNSIGNED_LONG, ...);
            MPI_Send(name.c_str(), size, MPI_CHAR, ...);
        }
};

LCMP(MpiClientProxy)
    L_CPP_PROVIDE(Hello, greeter);
    L_MPI_COMM(comm);
LEND
<lad><mpi>
  <process>
    <instance id="serv" type="Server"/>
    <instance id="serv_proxy" type="MpiServerProxy">...</instance>
  </process>
  <process>
    <instance id="client_proxy" type="MpiClientProxy"/>
    <instance id="client" type="Client">...</instance>
  </process>
  <communicator>
    <peer instance="client_proxy" property="comm"/>
    <peer instance="serv_proxy" property="comm"/>
  </communicator>
</mpi></lad>
Content

- Context
- Overview of component models
- L2C: Low Level Component
- Jacobi & L2C
- Experimental evaluation
- Conclusion & Perspectives
Jacobi computation

For iter = 0 to Niter
  For y = 0 to ymax
    For x = 0 to xmax
      tab[iter][x][y] = ...

run(size, niter)
compute(array)
Jacobi Parallel Computation

- Domain decomposition
  - 1 thread, 1 subdomain
- Frontier depends on neighbor value
  - Data overlap
- Overlap Consistency
  - Shared memory + synchronization
    - (Not in this talk: distinct memory regions)
  - Distributed memory + message passing
Thread Jacobi Parallelization

- 1 shared array
- Barrier after each iter

For iter = 0 to Niter
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = …
    - Barrier
MPI Jacobi Parallelization

- 1 local array per thread
- Send/receive at each iter

- For iter = 0 to Niter
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = ...
  - SendReceive
Hierarchic Parallelization

- Multi nodes
  - MPI
- Multi core
  - Threads

- For iter = 0 to Niter
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = ...
  - Local Barrier
  - SendReceive

Diagram:
- Main
- MPI iter
- Thread iter
- XY
The 4 connector way

- 1 connector instance
  - 1 domain
- 1 DataExchange/side
  - Implementation agnostic interface

- For iter = 0 to Niter
  - Wait for frontier
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = …
  - Data update T/B/L/R

```
class DataUpdate
{
  public:
    virtual void exchange ( ArraySlice in,
                            ArraySlice out
                        ) = 0;
};
```
The 4 connector way: Threads

```c
void exchange ( ArraySlice in, ArraySlice out )
{
    barrier(2);
}
```
The 4 connector way: MPI

void exchange ( ArraySlice in, ArraySlice out )
{
    MPI_SendReceive(in, out);
}
The 4 connector way: Hierarchy
Content

- Context
- Overview of component models
- L2C: Low Level Component
- Jacobi & L2C
- Experimental evaluation
- Conclusion & Perspectives
Experimental platform: Grid’5000

- Griffon cluster
  - Intel Xeon L5420 2.5 GHz
    - 4 cores per CPU
    - 2 CPU per node
  - 92 nodes
  - 16 GB RAM
- Infiniband-20G network
Performance vs Native Codes

No overhead of using component in Jacobi!
Iteration Time

Overhead coming from using too much threads on this machine!

- Limited memory bandwidth
Impact of #Threads per Node

![Graph showing speedup and efficiency vs. number of threads for different configurations: No cmp-1 node, Cmp-driver-1 node, No cmp-8 nodes, Cmp-driver-8 nodes. The graphs illustrate the performance impact of varying the number of threads per node.]
Speedup & Efficiency

The graph illustrates the speedup and efficiency of different configurations. The x-axis represents the number of threads or processes, and the y-axis shows the speedup and efficiency percentages. Different configurations include 2 threads, 2 processes, 4 threads, 4 processes, 8 threads, 8 processes, 8*2=16, 8*8=64, 16 nodes, 32 nodes, and 64 nodes.

The line colors indicate different categories: NOCMP, DRIVER, and CONNECTORS. The graph shows varying performance across these categories and configurations.
Software Complexity

### Number of Lines

<table>
<thead>
<tr>
<th>Jacobi Version</th>
<th>Native</th>
<th>Driver</th>
<th>Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>161</td>
<td>239</td>
<td>388</td>
</tr>
<tr>
<td>Multithreaded</td>
<td>338</td>
<td>386</td>
<td>643</td>
</tr>
<tr>
<td>MPI</td>
<td>261</td>
<td>285</td>
<td>446</td>
</tr>
</tbody>
</table>

### Code Reuse

<table>
<thead>
<tr>
<th>Code Reuse vs Seq (%)</th>
<th>Driver</th>
<th>Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread</td>
<td>26%</td>
<td>31%</td>
</tr>
<tr>
<td>MPI</td>
<td>32%</td>
<td>87%</td>
</tr>
<tr>
<td>MPI+Thread</td>
<td>-</td>
<td>100%</td>
</tr>
</tbody>
</table>
## Detailed Component SLOC

<table>
<thead>
<tr>
<th>Assembly Version</th>
<th>Cpt Name</th>
<th>SLOC</th>
<th>Seq. Reused</th>
</tr>
</thead>
<tbody>
<tr>
<td>Driver &amp; Connector</td>
<td>JacobiCore</td>
<td>25</td>
<td>Yes</td>
</tr>
<tr>
<td>Driver &amp; Connector</td>
<td>DataInitializer</td>
<td>68</td>
<td>Yes</td>
</tr>
<tr>
<td>Driver &amp; Connector</td>
<td>Main</td>
<td>105</td>
<td>Yes</td>
</tr>
<tr>
<td>Driver</td>
<td>SeqDriver</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>Driver</td>
<td>MPIDriver</td>
<td>155</td>
<td></td>
</tr>
<tr>
<td>Driver</td>
<td>ThreadDriver</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Connector</td>
<td>XP</td>
<td>71</td>
<td>Yes</td>
</tr>
<tr>
<td>Connector</td>
<td>JacobiNiter</td>
<td>187</td>
<td>Yes</td>
</tr>
<tr>
<td>Connector</td>
<td>ThreadXP</td>
<td>186</td>
<td></td>
</tr>
<tr>
<td>Connector</td>
<td>ThreadConnector</td>
<td>140</td>
<td></td>
</tr>
<tr>
<td>Connector</td>
<td>MPIConnector</td>
<td>58s</td>
<td></td>
</tr>
</tbody>
</table>
Software Complexity

- **Cyclomatic complexity**
  - *It directly measures the number of linearly independent paths through a program's source code.* Wikipedia

<table>
<thead>
<tr>
<th>Jacobi Version</th>
<th>Native</th>
<th>Driver</th>
<th>Connector</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential</td>
<td>28</td>
<td>32</td>
<td>8</td>
</tr>
<tr>
<td>Multithreaded</td>
<td>76</td>
<td>41</td>
<td>26</td>
</tr>
<tr>
<td>MPI</td>
<td>55</td>
<td>22</td>
<td>13</td>
</tr>
</tbody>
</table>
Conclusion & Perspectives

- Component model as a way to master versions
  - Application adaptation => assembly modification
- L2C
  - A simple, efficient, and extensible model
    - C++, MPI, CORBA, (FORTRAN 2008 soon)
- L2C assembly complex to write (XML)
  - Shall be generated by a higher model
  - HLCM: A high level component model
- Extensive memory bandwidth more machines
  - Curie (BULL) & Juquene (BG/Q)
- Design configuration algorithms