Evaluation of an HPC Component Model on Jacobi and 3D FFT Kernels

Christian Perez

Joint work with J. Bigot (CEA), V. Lanore (Inria), J. Richard (Orléans)

Avalon Research Team, LIP, Lyon, France

Clusters, Clouds, and Data for Scientific Computing (CCDSC’14)
La Maison des Contes, France, September 2nd-5th 2014
Context

Scientific Applications

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

- (High level) parallel languages
  - 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
    - Different algorithms!
  - 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
A global view of software engineering evolution

- **1980**: Procedural technology
  - Procedures, Pascal, C, ...
  - Procedural refinement

- **1995**: Object technology
  - Objects, Classes, Smalltalk, C++, ...
  - Object composition

- **2000**: Component technology
  - Packages, Frameworks, Patterns, ...
  - Component composition

- **Model technology**
  - Models, Metamodels, UML, OCL, MOF, XMI, SPEM, CWM
  - Model transformation

*From Jean Bézivin presentation, ATLAS group (Inria & LINA), Nantes, France*
OVERVIEW OF COMPONENT MODELS
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 (but in HPC)
  - CCA, Salome, CCM, Fractal, GCM, OGSi, SCA, ...
Common Component Architecture (CCA) Example

Dashed lines indicate alternate connections.

GoPort

IntegratorPort

Driver

IntegratorPort

MonteCarloIntegrator

RandomGeneratorPort

RandomGenerator

IntegratorPort

MonteCarloIntegrator

RandomGeneratorPort

RandomGenerator

IntegratorPort

MidpointIntegrator

FunctionPort

NonlinearFunction

FunctionPort

LinearFunction

FunctionPort

PiFunction

FunctionPort

RandomGeneratorPort

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

From CCA Tutorial, http://www.cca-forum.org/tutorials/
Component in Parallel Computing

- Memory sharing between components
  - CCA & CCM Extensions
- Parallel components
  - CCA, SCIRun2, GridCCM
- Collective communications
  - CCM Extension
- Parallel method calls
  - SCIRun2, GridCCM
- Master / worker support
  - CCA & CCM Extensions
- Some algorithmic skeletons in assemblies
  - STKM

Two type of features
- Component implementations
  - ≈ skeletons
- Component interactions
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 abstractions**
  - Language interoperability (~IDL)
  - Network transparency
    - Potential overhead when not needed
    - Limited data types systems
      - Babel SIDL, OMG IDL, …

- Programming model vs execution model
Programming model vs Execution model

- **L2C: Execution model**
  - Performance oriented
  - Close to hardware
  - Not so easy to make use

- **HLCM: « Programming » model**
  - Assembly oriented
  - Abstract hardware
  - Shall be “easy” to make use
OVERVIEW OF L2C LOW LEVEL COMPONENT
Low Level Component Model

- A minimalist component model for HPC
  - Component creation/deletion, configuration, and connection
  - An (optional) launcher
- No L2C code between components @ runtime
- Support native interactions
  - C++, MPI, CORBA, FORTRAN (2008)
- Extensible
- LGPL, available at hlcm.gforge.inria.fr
L2C: Connector Overview

- **C++/FORTRAN Interactions**
  - Use/Provide relationships
  - No language interoperability
    - Outside L2C goals

- **MPI Interactions**
  - Connector ~ communicator
L2C AND JACOBI
Jacobi Sequential Computation

Iter = N

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

run(size, niter)
compute(array)

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

- **1 shared array**
- **Barrier after each iteration**

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
The 4 connector way

1 connector instance
- 1 domain

1 DataExchange/side
- Implementation agnostic interface

For \( \text{iter} = 0 \) to \( N_{\text{iter}} \)
- Wait for frontier
- For \( y = 0 \) to \( y_{\text{max}} \)
  - For \( x = 0 \) to \( x_{\text{max}} \)
    - \( \text{tab}[\text{iter}][x][y] = \ldots \)
  - Data update T/B/L/R

```cpp
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);
}
```
void exchange ( ArraySlice in, ArraySlice out )
{
    MPI_SendReceive(in, out);
}
The 4 Connector Way: Hierarchy
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
Iteration Time

Overhead coming from using too much threads on this machine!

- Limited memory bandwidth
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>
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>
L2C AND 3D FFT
1D MPI 3D FFT Assembly
(2 nodes)
1D MPI 3D FFT Assembly
Homogeneous Experiments

1024^3 3D FFT – 1D Decomposition – Curie (Thin node)
Heterogeneous Experiments

256^3 3D FFT – 1D Decomposition : Edel+Genevi Cluster (Grid’5000)
2D MPI 3D FFT Assembly
(2 nodes)
Homogeneous Experiments

1024^3 3D FFT – 2D Decomposition – Curie (Thin node)
## Number of Lines & Reusability

<table>
<thead>
<tr>
<th>Version</th>
<th>C++ Lines of Code</th>
<th>% Reused Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2C 1D 2t xz</td>
<td>927</td>
<td>-</td>
</tr>
<tr>
<td>L2C 1D 1t xz</td>
<td>929</td>
<td>77%</td>
</tr>
<tr>
<td>L2C 1D 2t yz</td>
<td>929</td>
<td>100%</td>
</tr>
<tr>
<td>L2C 1D 2t yz blk</td>
<td>1035</td>
<td>69%</td>
</tr>
<tr>
<td>L2C 1DH 1t yz</td>
<td>983</td>
<td>80%</td>
</tr>
<tr>
<td>L2C 1DH 2t yz blk</td>
<td>1097</td>
<td>72%</td>
</tr>
<tr>
<td>L2C 2D 3t</td>
<td>1067</td>
<td>87%</td>
</tr>
<tr>
<td>L2C 2DH 3t</td>
<td>1146</td>
<td>69%</td>
</tr>
</tbody>
</table>
Conclusion & Perspectives

- Component model as a way to handle versions
  - Application adaptation => assembly modification

- L2C
  - A simple, efficient, and extensible model

- Towards component + task graph (L2C+StarPU)

- Efficient reconfiguration of component (on going)

- L2C assembly complex to write
  - Shall be generated by a higher model
  - HLCM: A high level component model

- Transformation algorithms from “HLCM” to “L2C”
Component and Task Graph

- **Component**
  - Good for describing application structure

- **Task graph**
  - Efficient to handle task dependencies

- **Towards a Component+Task graph model**
  - Runtime/Avalon/CEA PhD starting Nov 1.
  - Superseding L2C+StarPU
  - Gysela5D as a target application