Static 2D FFT Adaptation Through a Component Model Based on Charm++ (preliminary results)

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

\textsuperscript{1} ENS de Lyon, \textsuperscript{2} Inria
LIP, Avalon team

06/14/2013 – 9\textsuperscript{th} JLPC workshop
Context: HPC

Applications are used:
- on various architectures;
- with various input data and parameters.

Challenge: adaptation to improve performance.

Adaptation:
- **To what?** To architecture, to input parameters, to reservation size...
- **When?** At compile-time, at launch-time, at runtime...
- **How?** Parameter tweaking, low-level optimization, algorithmic changes, application structure changes...
Adaptation

Our focus:

- algorithmic-level adaptation;
- application structure adaptation.

How to implement as a developer?

Component models deal with application structure.

Goal of this presentation:

- illustrate adaptation challenges with the FFT example;
- evaluate the component approach for adaptation.
Plan

- Distributed FFT
  - Algorithms
  - Performance analysis
- Gluon++: a Charm++ Component Model
  - Overview
  - 2D FFT in Gluon++
- Evaluation
  - Performance
  - Software engineering
- Conclusions & Perspectives
Fast Fourier Transform

The Fast Fourier Transform (FFT):

- important tool in engineering and physics;
- used in many HPC applications.
  - notably in large-scale numerical simulations
    ⇒ distributed FFT
A widely-used Distributed FFT Algorithm

Two repeating steps:

- local FFT;
- matrix transposition (complete exchange).

1) FFT

2) transpose

3) FFT

4) transpose
Let \( N \) be the matrix size and \( p \) the number of cores.

Distributed FFT performance:

- High \( N/p \) \( \Rightarrow \) local FFT is dominant;
  - affected by node architecture, memory bandwidth...
  - well-known problem, e.g. FFTW [3].
- Low \( N/p \), high \( p \) \( \Rightarrow \) transposition is dominant;
  - affected by network latency, topology, bandwidth, memory bandwidth...
Linear Exchange (LEX)

Characteristics:
- variants: PEX, BEX;
- minimal data sent and copied in memory;
- $O(p^2)$ messages;
- good with large $N/p$. 

1) send

2) receive
Recursive Exchange (REX)

Characteristics:
- $O(p \times \log(p))$ messages;
- messages $N/2$ times larger;
- good with small $N/p$ and large $p$. 
Matrix transposition:

- select BEX/PEX/LEX or REX depending on $N$ and $p$;
- many more variants, *e.g.* from MPI [1,2].

Matrix decomposition: *e.g.* 3DFFT $\rightarrow$ slab or pencils [4].

Local FFT: *e.g.* FFTW codelets [3].

Such adaptations rely on **variant selection.** Existing solutions $\rightarrow$ specialized frameworks;

As a developer how to:

- develop and maintain variants;
- select variants (manually or automatically).
Plan

- Distributed FFT
  - Algorithms
  - Performance analysis

- Gluon++: a Charm++ Component Model
  - Overview
  - 2D FFT in Gluon++

- Evaluation
  - Performance
  - Software engineering

- Conclusions & Perspectives
Component Models

Components = black boxes that interact through ports
Application = assembly of component instances
Charm++
devolved in the Parallel Programming Laboratory at the University of Illinois

- Message-passing object-oriented language;
  - objects: “chares”;
  - distant asynchronous method calls through proxies.

- Platform-independent;
  - mapping chares to PEs;
  - chare arrays and groups (1 chare/PE).

- Performance;
  - latency tolerance;
  - dynamic load balancing.
Gluon++

developed by Julien Bigot in the Avalon team (Inria, LIP)

Assembly in separate file:
- instance list;
- placement on PEs;
- parameters.

gluon_loader
- loads required components only;
- resulting application is “component-free”.
Gluon++

developed by Julien Bigot in the Avalon team (Inria, LIP)

Assembly in separate file:
- instance list;
- placement on PEs;
- parameters.

gluon_loader
- loads required components only;
- resulting application is “component-free”.

developed by Julien Bigot in the Avalon team (Inria, LIP)
FFT2D in Gluon++

**Code reuse:** 2D matrix transposition from 1D FFT in gluon++.

**Local FFT:** FFTW (in Component Algo).
**FFT2D in Gluon++**

**Code reuse:** 2D matrix transposition from 1D FFT in gluon++.
**Local FFT:** FFTW (in Component Algo).
Plan

• Distributed FFT
  – Algorithms
  – Performance analysis

• Gluon++: a Charm++ Component Model
  – Overview
  – 2D FFT in Gluon++

• Evaluation
  – Performance
  – Software engineering

• Conclusions & Perspectives
Grid'5000
Griffon cluster

8-core nodes; 1PE/core. Infinibband network.

“Weak scaling”: $N/p$ is constant. High $N/p$: $p \times 500kB$ per proc.
Grid'5000
Griffon cluster
8-core nodes; 1PE/core. Infiniband network.
“Weak scaling”: \( N/p \) is constant. \( N/p=1 \).
Evaluation: Software Engineering

Component development:
- raw Charm++ programming plus a few macro calls;
- LEX/PEX/BEX → **reuse** of existing component + copy/paste/ rename + a few lines of code (<1 hour);
- REX → from scratch, a few days;

Component assembly:
- 20-line XML file; 4/5 lines per component;
- variant selection → one word;
- set attribute values;
- no recompilation.

Making a new component:
- (write new charm interface file);
- write new .cpp file
- compile component into .so;
- ready to use in assembly.

Component development and compilation → fully independent
Thanks to Charm++:
- easy component programming;
- performance.

Thanks to components:
- independent component development;
- easy assembly.

Gluon++ is a suitable solution for component design, concrete assembly and execution.

Remaining problem: how to generate gluon assembly?
→ to optimize performance;
→ while preserving component-independence.

Possible solution: generate from a high-level model.
Plan

• Distributed FFT
  – Algorithms
  – Performance analysis

• Gluon++: a Charm++ Component Model
  – Overview
  – 2D FFT in Gluon++

• Evaluation
  – Performance
  – Software engineering

• Conclusions & Perspectives
Conclusions and future work

Challenge:
- adaptation for HPC; variant selection;
- developer perspective.

Proposed answer: Gluon++
- Charm++;
- Components.

First external user of Gluon++.

Evaluation with 2D FFT:
- good performance on Grid'5000;
- easy variant development and selection.

Perspectives:
- more experiments (BlueWaters? Curie?);
- HLCM;
- 3D implementation;
  - slab/pencil decomposition;
  - comparison with Charm++ 3D FFT.
References


