Performance Study of Multicore Library on Grid’5000
PaSTeL and Expo

Brice Videau (MESCAL) and Érik Saule (MOAIS)

Supervision: Jean-François Méhaut and Olivier Richard

June 3rd, 2008
Context and Objectives
Context and Objectives

Context: Evolution in Processor Architecture

Toward multi-core architectures

- In order to limit thermal output:
  - Decrease in frequency
  - Increase the number of cores to make up for the performance loss
- The trend is going strong
- In a near future every computer will be multi-core

Figure: Intel prototype using 80 cores.
Problematic: What Can Be Expected of Modern Architectures?

As the number of cores increase, the available work per core decrease.

Moreover other parameters increase complexity:

- Different hardware
- Different programming paradigms

How small a task can be parallelized on these architectures using these paradigms?

What are the limiting factors?
Objectives: Build an Efficient and Tunable Parallel Library

Two parts: Runtime and Algorithm

Runtime:
- Competitive
- Tunable
- Simple

Algorithm:
- Comparable to others
- Allow the design of a use case
Objectives: In Depth Study of the Parallel Library

Study the performance of the parallel library:

- Variation of internal parameters
- Variation of hardware

Comparison with other parallel libraries.
Problematic - 2: Large Scale experiments

Performance study are long especially when many parameters vary:
⇒ Use Grid’5000 to parallelize experiments.

Grid’5000: cluster of cluster distributed among France

But:
- Many varying parameters
- Many machines involved
- Many results generated
Objectives - 2: Experiment Engine

Build a middleware to conduct experiments on light-weight grids.

We need:

- Fine monitoring of processes: stdout, stderr, return status
- Must not be (too) intrusive
- Fine control of resources: quality, hardware, software
- Native archiving mechanism
Talk Outline

1. Context and Objectives
2. PaSTeL
3. Expo
4. Conclusions and Perspectives
PaSTeL
Overview

Architecture

Algorithms and threads:
- One (or several) main threads
- Execute sequential program(s)
- Work on and gather results of parallel algorithms

Execution model: work-stealing
- \( n \) threads waiting to steal a main thread
- Have to be very reactive

PaSTeL is modelled and on simple algorithms its performance can be predicted
Reference values are obtained at runtime

Figure: Threading model
A subset of the STL

STL: Standard Template Library

A collections of algorithms in C++

Parallelized versions exist, thus it is a possible reference.

We Implemented a subset of the STL in PaSTeL
Example

Example use of PaSTeL:

```cpp
#include <pastel_algorithm>
int main (void)
{
    float array[SIZE];
    /* ..... */
    //std::sort(array, array + SIZE);
    pastel::sort(array, array + SIZE);
    /* ..... */
}
```
Methodology

Figures:
- Comparison between STL, MCSTL, TBB and PaSTeL
- Fixed number of threads
- Variable data sizes

Libraries:
- STL: Standard Template Library : sequential
- MCSTL: Multi-Core Standard Template Library
- TBB: Threading Building Blocks
## Platforms

<table>
<thead>
<tr>
<th>System</th>
<th>Processor Type</th>
<th>Cores</th>
<th>Frequency</th>
<th>Cache Size</th>
<th>Memory</th>
<th>System</th>
<th>Compiler</th>
<th>Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Idkoiff</strong></td>
<td>8 * AMD Opteron 875</td>
<td>16</td>
<td>2.2GHz</td>
<td>1024kb per core</td>
<td>32 GB</td>
<td>Linux 2.6.23</td>
<td>gcc 4.2.3</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Nedni</strong></td>
<td>Intel Core 2 Duo</td>
<td>2</td>
<td>1.8GHz</td>
<td>2048kb shared</td>
<td>2 GB</td>
<td>Linux 2.6.23</td>
<td>gcc 4.2.3</td>
<td>2</td>
</tr>
</tbody>
</table>
Performance Study of Multicore Library on Grid'5000

Context and Objectives

min_element : nedni

min_element algorithm on an array of int on an Intel Core2 Duo

STL
TBB
PaSTeL
PaSTeL evaluation

size

time (in cycles)
min_element : idkoiff

min_element algorithm on an array of int on 4 AMD Opteron 875

- STL
- TBB
- PaSTeL
- PaSTeL evaluation

Performance Study of Multicore Library on Grid'5000
Expo: an Experiment Engine

Expo aims at being:
- Tailored for lightweight grids...
- But portable
- Easy to use...
- But powerful

Expo:
- Supervises experiments
- Archives experiments
- Is modular
- Is fault tolerant
Expo Modular Architecture

Different modules in Expo
- Dedicated Language
- Experiment Module
- Reservation Module
- Command Execution Module
- + Parallel Launcher
Expo Dedicated Language: Overview

Based on Ruby, an interpreted object oriented language. The Expo Language presents different abstractions to the user:

- Resources and Resource Sets
- Tasks and Parallel Tasks
- Synchronous and Asynchronous Tasks
- Reservations
- Parallel Sections and Barriers
Simple Use Case

1. require ’expo_g5k’
2. oargridsub :res => ”gdx:nodes=10,helios:nodes=10”
3. check $all
4. ptask $all.gateway, $all, ”date”
5. id, res = ptask $all[”gdx”].gateway, $all[”gdx”], ”sleep 1”
6. res.each { |r| puts r.duration }
7. puts ”moyenne : ” + res.mean_duration
Conclusions and Perspectives
Conclusions

PaSTeL:
- Efficient parallelization possible on small datasets
- Several algorithms take advantage of the parallel implementation
- Extremely reproducible results
- Code is still simple enough to be easily modified and modelled

Expo:
- Can already be used on grids with ssh access
- Concise language
- Powerful resource management
Perspectives

Use those two tools to study and model modern architectures

PaSTeL:
- Implement other models than work-stealing
- Use other threading libraries

Expo:
- Improve the DSL with feedback from the new study
- Port Expo to other grid architectures
Annexe

\[(n-1)l_s\]

\[2l_s\]

\[l_s\]

\[t_n\]

\[t_3\]

\[t_2\]

\[t_1\]

\[0\] \[t_1\] \[l_e\] \[T\]