ParaFormance™: An Advanced Refactoring Tool for Parallelising C++ Programs – Part 3

Chris Brown, Vladimir Janjic, Kevin Hammond
University of St Andrews, Scotland
@chrismarkbrown
@rephrase_eu
1. Introduction and overview to Safety Checking
2. Live Demonstration of Safety Checking
3. Follow along interactively
4. Ant Colony
5. Wrap up
6. Questions and Answers
What is Safety Checking?

1. Introduces parallelism into an application semi-automatically
2. Refactors a sequential portion of code into a parallel version
3. Introduces all parallel ‘business logic’
Why?

- Huge saving in effort over manual
- Difficult to get right!
- Can work on inserted code or existing code
- Without safety checking there is basically no way to know if your existing application will run without errors or not
Demonstration
Predicting Performance
Which Configuration?

Speedups for $\Delta(r) \parallel \Delta(p)$

Speedups for $\Delta(r) \parallel p$
Predicted vs. Actual Speedups

Speedups for denoise

- Pipe(G, D)
- Pipe(G, ParMap(D))
- Pipe(Farm(G), ParMap(D))
- Pred. Pipe(G, D)
- Pred. Pipe(G, ParMap(D))
- Pred. Pipe(Farm(G), ParMap(D))
Predicted vs. Actual Speedups

All measurements have been made on an 800 MHz 24 core, dual AMD Opteron 6176 architecture, running CentOS Linux 2.6.18-274.el5 and Erlang 5.9.1 R15 B01, averaging over 10 runs. Figure 5, left, compares the predicted (dashed) speedups against the actual (solid) speedups. The overall predicted speedup for denoise on 24 cores for the Pipe(Farm(G), ParMap(D)) version is 19.09.
Wrap up
## Comparison of Development Times

<table>
<thead>
<tr>
<th></th>
<th>Man. Time</th>
<th>Refac. Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Convolutions</td>
<td>24 hours</td>
<td>3 hours</td>
</tr>
<tr>
<td>Ant Colony</td>
<td>8 hours</td>
<td>1 hour</td>
</tr>
<tr>
<td>Basic N2</td>
<td>40 hours</td>
<td>5 hours</td>
</tr>
<tr>
<td>Graphical Lasso</td>
<td>15 hours</td>
<td>2 hours</td>
</tr>
</tbody>
</table>
The performance of Image Convolution on the xookik platform is shown in the graph. The x-axis represents the number of threads, while the y-axis shows the speedup. The graph compares different parallelization methods:

- **OpenMP**:
  - Sequential to parallel (s | m)
  - Parallel to parallel (m | m)
  - Parallel to sequential (m | s)

- **TBB**:
  - Sequential to parallel (s | m)
  - Parallel to parallel (m | m)
  - Parallel to sequential (m | s)

- **FF**:
  - Sequential to parallel (s | m)
  - Parallel to parallel (m | m)
  - Parallel to sequential (m | s)

The graph indicates that OpenMP and TBB provide similar speedups, with OpenMP slightly outperforming TBB in some cases. The best speedup is achieved with sequential to parallel (s | m) and parallel to sequential (m | s) methods. The performance improves as the number of threads increases, but there is a point of diminishing returns where adding more threads no longer significantly improves the speedup.
Image Convolution

Speedups for Image Convolution on *titanic*

- **OpenMP** $(s | m)$
- OpenMP $(m | m)$
- OpenMP $m$
- **TBB** $(s | m)$
- TBB $(m | m)$
- TBB $m$
- **FF** $(s | m)$
- FF $(m | m)$
- FF $m$

---

**The diagram shows speedups for Image Convolution on the *titanic* dataset using different parallelization methods.**

- The x-axis represents the number of CPU threads used.
- The y-axis represents the speedup.

**Key observations:**
- OpenMP $(s | m)$ and TBB $(s | m)$ show similar performance improvements.
- FF $(s | m)$ has slightly better performance compared to the others.
- The maximum speedups achieved are 12, 11, and 16 on different datasets.

---

**Parallel design patterns are provided as algorithmic skeletons.**

**Old:**

- In our system, other authors use rewriting/refactoring to support design space exploration.
- There has so far been only a limited amount of work on refactoring for parallel programs by using pattern rewriting rules.
- An early work in refactoring has been described in [22].
- A good survey (as of 2004) can be found in [17].
- There has so far been only a limited amount of work on refactoring for parallel programs by using pattern rewriting rules.
- An early work in refactoring has been described in [22].
- A good survey (as of 2004) can be found in [17].

**New:**

- In our system, other authors use rewriting/refactoring to support design space exploration.
- There has so far been only a limited amount of work on refactoring for parallel programs by using pattern rewriting rules.
- An early work in refactoring has been described in [22].
- A good survey (as of 2004) can be found in [17].
- There has so far been only a limited amount of work on refactoring for parallel programs by using pattern rewriting rules.
- An early work in refactoring has been described in [22].
- A good survey (as of 2004) can be found in [17].
The number of threads is used. This is due to the fact that a lot of speedups, and on each system, speedups tail off after a certain number. This indicates that it is not always the best idea to use more threads. Furthermore, both different libraries give approximately the same speedups, with no library achieving linear speedup. Finally, we can note that none of the libraries is able to achieve linear speedup as the number of threads increases, which gives slightly better speedup than other libraries. Furthermore, both different parallelizations give approximately the same speedups, with no library achieving linear speedup. Finally, we can note that none of the libraries is able to achieve linear speedup as the number of threads increases.

Figure 11, we can observe (similarly to the Image Convolution example) that speedups are similar for all parallel libraries.
Finally, we intend to investigate applications. This will add to the evidence that our approach is approach with these new skeletons on a wide range of realistic computer algebra and physics, and demonstrate the refactoring tion, further facilitating this approach. In addition, we also regarding parallel efficiency can be determined via optimisa-

of the refactoring tool is achieved, the best parametrisation architectures, including adding refactoring support for GPU particularly useful when porting the applications to different

predict the parallel performance from applying a particular

directions, including adding advanced performance models to

future we expect to develop this work in a number of new

achieved with the refactoring tool are approximately the same

as for full-scale manual implementations by an expert. In

without major performance losses: as desired, the speedups

supervised by the programmer. This can give significant sav-

code and simply points the refactoring tool towards them. The

Figure 3. Refactored Use Case Results in FastFlow

Comparison of Speedups for Ant Colony, BasicN2 and Graphical Lasso

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
<td>24</td>
</tr>
</tbody>
</table>

References

Road Map – Watch this space!

First Version Released

May 2017

Now
Conclusions

- Refactoring tool support:
  - Guides a programmer through steps to achieve parallelism
  - Warns the user if they are going wrong
  - Avoids common pitfalls
  - Helps with understanding and intuition
  - Reduces amount of boilerplate code
    - Allows programmer to concentrate on algorithm, rather than parallelism.
ParaFormance™ Needs You!

• Please join our mailing list and help grow our user community
  - news items
  - access to free development software
  - chat to the developers
  - free developer workshops
  - bug tracking and fixing
  - Sign up for tool demos/trials

• Subscribe at

https://mailman.cs.st-andrews.ac.uk/mailman/listinfo/paraformance-news
THANK YOU!

http://rephrase-ict.eu

@rephrase_eu