Towards a Holistic Approach to Auto-Parallelization

Integrating Profile-Driven Parallelism Detection and Machine-Learning Based Mapping

Georgios Tournavitis, Zheng Wang, Björn Franke and Michael F.P. O’Boyle

MPSoCs 2009, St. Goar, Germany.
Overview

- Introduction & motivation
- Profile-driven parallelization
- ML-based mapping
- Empirical evaluation
- Conclusions & future work
Introduction

• Performance on multi-core architectures requires extraction of thread-level parallelism
  - Legacy code is sequential
  - Majority still thinks/programs sequentially
  - Parallel programming is tedious and error-prone

• Auto-parallelization
  - 30 years of research
  - Fails to deliver performance beyond niche settings
State-of-the-art

• State-of-the-art auto-parallelizer

• Array-based scientific applications

• On average **no speedup!!!**
State-of-the-art

- Wide performance gap
- Why?

NAS NPB 2.3 OMP-C and SPEC CFP2000
2 Quad-cores (8 cores in total) Intel Xeon X5450 @ 3.00GHz
Intel icc 10.1 -O2 -xT -axT -ipo
State-of-the-art

- Wide performance gap
- Why?

NAS NPB 2.3 OMP-C and SPEC CFP2000
2 Quad-cores (8 cores in total) Intel Xeon X5450 @ 3.00GHz
Intel icc 10.1 -O2 -xT -axT -ipo
State-of-the-art

- Wide performance gap

1. Static parallelism detection is **conservative** due to lack of information

2. Lack of **automatic** and **portable** approach to parallelism mapping

NAS NPB 2.3 OMP-C and SPEC CFP2000
2 Quad-cores (8 cores in total) Intel Xeon X5450 @ 3.00GHz
Intel icc 10.1 -O2 -xT -axT -ipo
Problem statement

Parallel loops
Problem statement

- Parallel loops
- Statically provable parallel
Problem statement

- Unexploited
- Statically provable parallel
Problem statement

Unexploited

Statically provable parallel
Problem statement

- Unexploited
- Statically provable parallel

Profitable
Overview of holistic approach

1. Sequential code in C
2. Profiling-driven analysis
3. Code with OpenMP annotations
4. Machine-Learning based mapper
5. Code with profitable loops
Overview of holistic approach

Sequential code in C

Profiling-driven analysis

Code with OpenMP annotations

Machine-Learning based mapper

Code with profitable loops
Overview of holistic approach

- Sequential code in C
- Profiling-driven analysis
- Code with OpenMP annotations
- Machine-Learning based mapper
- Code with profitable loops
Profiling-driven parallelism detection

Keypoint: restrictions of static analysis can be overcome using precise, dynamic information

How?

• Instrument the middle-level IR
  - Straightforward correlation
  - High-level information (loops, induction/reduction vars)
  - Avoid obfuscation due to ISA
  - Execute natively
Profiling-driven parallelism detection

• Dynamically reconstruct precise view of control and data flow
  - Identify parallel loops
  - Identify privatizable data
  - Use hybrid static/profiling-based approach to uncover parallel reductions

• Unsafe so user validates correctness
Overview

Sequential code in C → Profile-driven analysis → Code with OpenMP annotations → Machine-Learning based mapper → Code with profitable loops
ML-based mapping

- How do we select/map profitable loops?
- Traditional compilers use hard-wired heuristics
- ML provides portable and automatic modeling of compiler/runtime/architecture
ML-based mapping

How good heuristics can be?

![Graph showing the relationship between Number of Instructions and Number of Iterations. Points represent whether the loops should be parallelized or not, with blue dots indicating parallelization and pink dots indicating no parallelization. There is a dashed line indicating the threshold for parallelization.]
ML-based mapping

- Predictive modeling based on Support Vector Machine (SVM)
  - Decide (i) profitability, (ii) loop scheduling
  - Construct hyperplanes in **transformed higher-dimensional** space
  - Non-linear & multiclass extensions

\[ f' = f \cdot X - b \]
ML-based mapping

- **Off-line** learning
- Predict using **smallest** input

**Features**

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR instruction count</td>
<td>Data access count</td>
</tr>
<tr>
<td>IR Load/Store count</td>
<td>Instruction count</td>
</tr>
<tr>
<td>IR Branch count</td>
<td>Branch count</td>
</tr>
<tr>
<td>Loop iteration count</td>
<td></td>
</tr>
</tbody>
</table>

**Diagram**

1. New program
2. Feature extraction
3. Trained model
4. Profitable or not
5. Scheduling policy
Final approval

- Profile-driven parallelization cannot guarantee correctness for a new input
Empirical evaluation
Benchmarks

• 2 sets of applications
  - NAS Parallel Benchmarks 2.3
  - SPEC FP2000 (subset in C)

• 2 versions
  - Sequential
  - Manually parallelized using OpenMP by expert programmers

• All input datasets
Performance on Intel

- Intel icc **fails** to deliver any improvement
- For small datasets even **slowdown**

---

NAS NPB 2.3 OMP-C and SPEC CFP2000
2 Quad-cores (8 cores in total) Intel Xeon X5450 @ 3.00GHz
Intel icc 10.1 -O2 -xT -axT -ipo
Performance on Intel

- Intel icc **fails** to deliver any improvement
- For small datasets even **slowdown**

NAS NPB 2.3 OMP-C and SPEC CFP2000
2 Quad-cores (8 cores in total) Intel Xeon X5450 @ 3.00GHz
Intel icc 10.1 -O2 -xT -axT -ipo
Performance on Intel

- Profile-driven: **3.34** average speedup
- **96%** of hand-parallelized
Performance on Cell

- On average **manual** parallelization delivers **slowdown**
- OpenMP overheads/tradeoff radically different
Performance on Cell

- On average **manual** parallelization delivers **slowdown**
- OpenMP overheads/tradeoff radically different
Performance on Cell

• Profile-driven + ML: avoids slowdowns
• On average speedup

NAS NPB 2.3 OMP-C and SPEC CFP2000
Dual Socket, QS20 Cell Blade
IBM xlc ssc v0.9 O5 -qstrict -qarch=cell -qipa=partition=minute
Conclusions

• Static-only parallelization approaches **fail**

• Profiling is effective in uncovering parallel loops

• ML mapping adapts across architectures

• Profiling information + ML combined outperforms each approach in isolation

• Performance of holistic approach close to manual parallelization
Future work

• Consider low-overhead TLS/TM to ensure correctness
• Study on a wider set of benchmarks
• Extend to partially sequential loops
Thanks!!!

Acknowledgments:

▶ MilePost Consortium
- 3 year STREP project funded by the EU approx 1.6 M Euro
- Started July 2006. Tools for embedded systems

▶ Industry
- IBM: gcc and machine learning
- CAPS-Enterprise: embedded libraries and search technology
- ARC: Reconfigurable IP and compilation

▶ Academia
- Edinburgh: embedded optimising compilers + machine learning
- INRIA: adaptive and continuous compilation