Coalition Threading: Combining Traditional and Non-Traditional Parallelism to Maximize Scalability

Md Kamruzzaman   Steven Swanson   Dean Tullsen
Modern Architectures

Intel Core i7

AMD Phenom

IBM Power7

Performance does not always scale with core counts
Parallelization Wall

• Amdahl’s law
  – Serial part becomes dominant as you scale more

• Not all codes are parallelizable
  – Parallelization is hard

• Parallelization complexity
  – Load balancing
  – Synchronization overhead
  – Cache coherence

How can we utilize so many cores?
Non-traditional Parallelism (NTP)

• Providing parallel speedup without actually parallelizing the code
  – Uses multiple cores or hardware contexts but one computing thread

• Helper thread prefetching
  – ISCA ’99, ISCA ’01, ASPLOS ’02, PLDI ’02, ASPLOS ’11
Non-traditional Parallelism (NTP)

• Providing parallel speedup without actually parallelizing the code
  – Uses multiple cores or hardware contexts but one computing thread

• Helper thread dynamic recompilation
  – ISCA ’01, MICRO ’01, MICRO ’05, HPCA ’07
Non-traditional Parallelism (NTP)

• Providing parallel speedup without actually parallelizing the code
  – Uses multiple cores or hardware contexts but one computing thread

• Software data spreading
  – PLDI ’10
Serial vs. Parallel Code

Adding helper threads is free

Trading computing threads for helper threads
Serial vs. Parallel Code

First to investigate whether NTP is effective for parallel code
Applying NTP to Parallel Code

- Application has different phases and parallelization techniques impact them differently.

![Diagram showing parallelization and NTP effects on loops]

Application has different phases and parallelization techniques impact them differently.
Applying NTP to Parallel Code

- Application has different phases and parallelization techniques impact them differently.

```
<table>
<thead>
<tr>
<th></th>
<th>Loop 1</th>
<th>Loop 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallel</td>
<td>4x, 3x</td>
<td>2x, 4x</td>
</tr>
<tr>
<td>NTP + Parallel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Traditional Parallelization 3x
Applying NTP to Parallel Code

- Application has different phases and parallelization techniques impact them differently.

```
<table>
<thead>
<tr>
<th>Loop 1</th>
<th>Loop 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image.png" alt="Diagram" /></td>
<td><img src="image.png" alt="Diagram" /></td>
</tr>
</tbody>
</table>

NTP + Traditional Parallelization 3.5x
Applying NTP to Parallel Code

- Application has different phases and parallelization techniques impact them differently.

Coalition threading applies NTP only when useful 4x
Coalition Threading
Coalition Threading

- Combines **traditional** and **non-traditional** parallelization intelligently to maximize scalability

- Loop level granularity
  - Coalition threading decides on loop level about which techniques to apply

Traditional parallelization $\rightarrow$ Programmer specified pthread or openmp parallelization

Non-Traditional component $\rightarrow$ Inter-core prefetching (Kamruzzaman et al., ASPLOS ’11)
Inter-core Prefetching (ICP)

- Thread migration allows us to prefetch remotely, then access locally
- Software only technique
- ICP outperforms other techniques
  - SMT prefetching and shared cache prefetching
  - Software data spreading
Inter-core Prefetching (ICP)

- Execution proceeds **chunk by chunk**
  - Helper threads *prefetch* data for future chunks in *private caches*
Inter-core Prefetching (ICP)

- Execution proceeds **chunk by chunk**
  - Helper threads **prefetch** data for future chunks in **private caches**

```
Core - 0

$ $

Core - 1

$ $
```

Data already in the cache

- Threads swap cores
  - Prefetching chunk 3
  - Main execution chunk 2

Execution proceeds **chunk by chunk**
- Helper threads **prefetch** data for future chunks in **private caches**

```
Core - 0

$ $

Core - 1

$ $
```
Inter-core Prefetching (ICP)

- Execution proceeds **chunk by chunk**
  - Helper threads **prefetch** data for future chunks in **private caches**
Inter-core Prefetching (ICP)

- Execution proceeds **chunk by chunk**
  - Helper threads **prefetch** data for future chunks in **private caches**

```
Core - 0

Core - 1

Main execution chunk 4

Prefetching chunk 5

Threads swap cores
```

Helper threads lead the way for the main thread
Number of Helper Threads

One helper thread is enough to handle slower main thread

Multiple helper threads keep pace with the faster main thread
Distribution of Main and Helper Threads

High opportunity cost for parallel code
Distribution of Main and Helper Threads

Par-ICP with 1 helper thread works well for coalition threading

Loop 0  Loop 1  Loop 2  Loop 3  .................  Loop n

Which loops to apply par-ICP
Methodology
Coalition Threading Framework

• Source to source translator based on Rose

• Handles both openmp and pthread code

• The framework adds code for
  – Splitting the loop into chunks
  – Coordination of main thread and corresponding helper threads
  – Necessary synchronizations

• In this work, the prefetching code is manually constructed
Methodology

- Quad-socket 32-core AMD Opteron system
- 108 key loops from 11 benchmarks from NAS, SpecOMP, Parsec, and Graph500
- All 108 loops are pre-parallelized
- Hardware prefetcher turned on, “-O3” optimization using gcc
Interaction Between Traditional and Par-ICP
Impact of Par-ICP on Loops

Par-ICP wins for all thread counts

Par-ICP wins for large thread counts

Loop scales poorly or negatively

Par-ICP loses for all thread counts
Impact of Par-ICP on Loops

In 15% loops, par-ICP wins for all thread counts.
Impact of Par-ICP on Loops

In 30% loops, par-ICP wins for large thread counts
Impact of Par-ICP on Loops

Loop scales poorly, par-ICP accelerates each main thread
Impact of Par-ICP on Loops

Par-ICP loses for all thread counts in 54% loops
Impact of Par-ICP on Loops

Coalition threading tries to apply the best parallelization technique on each loop

All four loops are from the same application
Par-ICP is effective for 39% loops.
Decision for some loops is more critical in coalition threading.
Heuristics for Coalition Threading
Linear Classifier Heuristic

• Linear classifier uses a hyperplane to best separate a set of n-dimensional points into two classes

• We experiment with 12 loop characteristics
  – Measured by profiling serial execution

• Factors that impact scalability of parallel execution
  – Parallelization overhead (PO)

• Factors that impact effectiveness of ICP
  – Chunk data reuse (CDR)
  – Cache misses per instruction
Linear Classifier Heuristic
• Parallelization overhead impedes main thread progress but does not impact helper threads
• Coherence activity increases with CDR
• Too many misses imply need for multiple helper threads
Linear Classifier Heuristic

- LC7 heuristic uses 7 key characteristics
- 10-fold cross validation accuracy 84%
Prediction accuracy improves with larger training sets

LC7 Prediction Accuracy

- 14% loops **GAIN** more than 50% performance using par-ICP
- 23% loops **LOSE** more than 30% performance using par-ICP

- Prediction accuracy improves with larger training sets
Performance of LC7 Prediction

- LC7 classifies critical loops better
- 98% accuracy when performance impact is at least 25%
Impact on Application Level
Application Level Scalability

Average Speedup

Total number of threads used
• The benefit by coalition threading is higher in large thread counts.
Application Level Scalability

- Blind par-ICP gradually catches up traditional threading.
**Application Level Scalability**

- LC7 based heuristic CT performs as good as oracle CT
- 13% improvements on average for 32 threads
• Naive par-ICP improves SP and streamcluster by 21% and 40% successively
Impact of Different Parallelization Techniques – 32 Threads

- Heuristic coalition threading provides 17% speedup on average, and as high as 51%
• LC7 heuristic based coalition threading provides only 0.7% less speedup on average than an oracle
• Incorrectly classified loops have nominal impact
Conclusion

- Coalition threading combines **non-traditional parallelism** with traditional threading
- **Inter-core prefetching** is effective for parallel loops and par-ICP provides better scalability in 39% cases
- Linear classifier based heuristic can classify **98%** loops when the decision is critical
- Heuristic coalition threading works only **0.7% less well** than the oracle
- Application level speedup is **17%** on average and as high as **51%**
Thank You

Questions ?
Backup Slides
Impact of Par-ICP on Loops

Loop scales poorly, par-ICP accelerates each main thread