SCORE: Scalability-Oriented Optimization

Emery Berger (UMass Amherst)
Stephen Freund (Williams College)

Charlie Curtsinger (UMass Amherst, Grinnell)
Zander Majercik, Yitong Tseo (Williams)
SCORE Architecture

• Competing goals for concurrent programming:
  – correctness
  – performance
• SCORE: automatically improve scalability without compromising correctness.

Dynamic Analysis
- Causal Profiling
- Bottleneck Identification
  remediation candidates

Code Optimization
- Code Transformation and Robustification
  bottleneck targets & proposed remediations
  transformed executable

Runtime System
- Performance Monitoring
- Remediation Supervision
  new/revised remediations
SCORE Architecture

**Dynamic Analysis**
- Causal Profiling
- Bottleneck Identification
  - remediation candidates

**Code Optimization**
- Code Transformation and Robustification
  - transformed executable
  - new/revised remediations

**Runtime System**
- Performance Monitoring
- Remediation Supervision

**Examples**
- replace mutex with R/W lock or optimistic control
- remove synchronization
- change lock granularity
- rebalance thread pools
SCORÉ Architecture

- **Causal Profiling**
  - Finds where performance tuning will help
  - For Linux [Charlie Curtsinger]
  - For JVM [Freund]
Profilers (eg: gprof, perf)

- Serial Execution
  - runs for a long time or runs many times

- Parallel Execution
Where Will Performance Tuning Help?

The program runs **this much** faster if you can speed up **?** by **this much**.
Virtual Speedups

Each time B runs, we **pause** all other running threads.
Virtual Speedups

Each time B runs, we **pause** all other running threads.
Virtual Speedups

Each time B runs, we pause all other running threads.

After accounting for the size of the delay we inserted, we can measure the expected program speedup.
Virtual Speedups

Each time B runs, we pause all other running threads.
Virtual Speedups

Each time A runs, we pause all other running threads.
Virtual Speedups

Each time \textcolor{blue}{A} runs, we \textbf{pause} all other running threads.
Virtual Speedups

Each time A runs, we pause all other running threads.

No program speedup.
Coz Causal Profiler for Linux

• Measures
  – total running time, or
  – latency/throughput between user-inserted progress points.

• Sampling-based
  – performs many short experiments
  – random source line / amount for virtual speedup
  – reports most promising lines for optimization

• github.com/plasma-umass/coz
We can “optimize” a pipeline stage by reassigning threads to that stage. Moving six threads from extraction to ranking yielded a **21% speedup**.
## Coz Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Speedup</th>
<th>Diff Size</th>
<th>Change Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>memcached</td>
<td>9.39%</td>
<td>-6, +2</td>
<td>removed unnecessary locks</td>
</tr>
<tr>
<td>sqlite</td>
<td>25.60%</td>
<td>-3, +3</td>
<td>removed DIY vtable implementation</td>
</tr>
<tr>
<td>blackscholes</td>
<td>2.56%</td>
<td>-61, +4</td>
<td>manual common subexpression elimination</td>
</tr>
<tr>
<td>dedup</td>
<td>8.95%</td>
<td>-3, +3</td>
<td>fixed degenerate hash function</td>
</tr>
<tr>
<td>ferret</td>
<td>21.27%</td>
<td>-4, +4</td>
<td>rebalanced pipeline thread allocation</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>37.50%</td>
<td>-1, +0</td>
<td>removed custom barrier with high contention</td>
</tr>
<tr>
<td>streamcluster</td>
<td>68.40%</td>
<td>-1, +0</td>
<td>removed custom barrier with high contention</td>
</tr>
<tr>
<td>swaptions</td>
<td>15.80%</td>
<td>-10, +16</td>
<td>reordered loop nests</td>
</tr>
</tbody>
</table>
Summary

• SCORE: automatically improve scalability without compromising correctness.

• Causal profiling:
  – predicts program speed up based on virtual speedup experiments (~20% overhead)
  – has identified optimization opportunities
  – provides accurate predictions

• Currently integrating dynamic synchronization and remediation analysis.