Collaborative Parallelization Framework

Ziyang Xu, Greg Chan,
Sotiris Apostolakis, and
David I. August

Liberty Research Group
Princeton University
Automatic parallelization is great but...

Figure 7: Enabling effect of Privateer at 24 worker processes. [1]

15x on 24 cores

2.5x on 6 cores

Figure 10: Overall speedup (Benchmarks in the legend are ordered from highest to lowest speedup) [2]

42x on 120 cores

Problem 1: Each Technique is Applicable to a Subset of Programs

No benchmarks appear in all evaluations

Figure 7: Enabling effect of Private at 24 worker processes.

Figure 10: Overall speedup (Benchmarks in the legend are ordered from highest to lowest speedup)

Figure 9: Speedups achieved by HELIX on a real system

Table 1: Percentage sequential code coverage of various transformations – Last column shows the Abort frequencies in the benchmarks. Coverages higher than 20% are highlighted. (CS: control speculation for uncounted loop; RE: reduction expansion; SF: speculative fission; PM: prematerization; IDE: infrequent dependence isolation. LDP: [1] More long distance dependence. AF: abort frequency.)

Problem 2: Getting Optimizations to Work Together in Current Compiler Design is too Complex

Monolithic Coordinating Different Transformations

Modular Without Much Coordination
Collaborative Parallelization Framework

Insight:
• Existing optimizations know how other optimizations can help
• Listen to the high value transformations, like DOALL - call them Critics
• Have low value, enabling transformations respond - call them Remediators
• Have a standard Remediation and Critic interface
Insight:
The Criticisms are easily expressed as dependences on the program dependence graph to remove.
Conclusion

• Combine techniques into one framework
  • High value transformations as Critics
  • Enabling transformations as Remediers

• Enable collaboration
  • Use Orchestrator to centralize decision-making
  • Standard interface

• Opportunity
  • Treat programmers as a Remedier and bring them into the collaboration
Thank you

Questions?

Liberty Research Group
Princeton University
Orchestrator:
• Coordinate other parts
• Generate final transformation
• Centralize decision-making

Insight: Existing Optimizations Know How Other Optimizations Can Help
Critic:
• Address dependences
• Give potential speedup
Remediator:
• Express transformation effect instead of apply them
• Give estimated cost
**Programmer as a Remediator**

- **Programmer Remediator:**
  - Translate criticisms to high-level yes/no questions
  - High-probability assumptions

Assume expected answer, ask programmer only when remedy part of final parallelization plan.
How to compose?

$E_1$, $E_2$, $P_1$

$E_3$, $E_1$, $P_2$

$E_i$, $E_j$, $E_k$, $P_n$

$E_i$: Enabling Transformations
(e.g., Memory Speculation)

$P_i$: Parallelization Techniques
(e.g., DOALL, PS-DSWP)
## Comparison

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Parallelization Technique</th>
<th>Speculation</th>
<th>User Feedback</th>
<th>Automatic Decision Process</th>
<th>Geomean Speedup/Cores</th>
<th>Best Speedup Overall (# of benchmarks)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster SpecDOALL</td>
<td>DOALL</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>43.8x/120</td>
<td>6</td>
</tr>
<tr>
<td>Privateer</td>
<td>DOALL</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>11.4x/24</td>
<td>3</td>
</tr>
<tr>
<td>DSMTX</td>
<td>DOALL, PS-DSWP</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>49x/128</td>
<td>8</td>
</tr>
<tr>
<td>SMTX</td>
<td>DOALL, PS-DSWP</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>13.36x/24</td>
<td>4</td>
</tr>
<tr>
<td>HELIX</td>
<td>HELIX</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>2.25x/6</td>
<td>9</td>
</tr>
<tr>
<td>CommSet</td>
<td>DOALL, PS-DSWP</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>5.7x/8</td>
<td>7</td>
</tr>
</tbody>
</table>