Efficient Discovery of Loop Nests in Execution Traces

Qiang Xu  
CGG Veritas Inc.

Jaspal Subhlok  
Nathaniel Hammen  
University of Houston

MASCOTS 2010
Loop Nest Discovery: What?

Execution trace of events represented as a string symbols converted to loop nest, e.g.,

\[
\text{abababababab} \rightarrow (ab)^4
\]
\[
\text{aabaabaabcaabaabaabc} \rightarrow ((a^2b)^3c)^2
\]

Communication trace of NAS LU Benchmark with 32K symbols

\[
(V)^3(W)^2X(W)^3(QJYQWHHYQEYQCY)^2RU((NKID)^{160}(MLGB)^{160}QJYQHYQEYQCY)^{249}(NKID)^{160}RQJYQHYQEYQCYRSRPYAOF(T)^3
\]

Each letter represents a unique MPI operation
Loop Nest Discovery: Why?

Summarized trace ideal for re-execution for performance prediction

vs. tables (gzip) or grammar (Sequitur)

Context: Performance Skeletons
Definition: short running program representing an application-- skeleton execution time is predictor of application execution time

Building: Identify the loop structure in execution trace and generating code to mimic loop body execution
Rest of the Talk

- Discovery of “optimal” loop nest
- Fast Greedy loop nest discovery
- Performance results
- Theoretical “near optimality” of Greedy algorithm
“Optimal” Loop Nest Discovery

Approach: recursively discover the longest span (outermost) loop in a trace

Challenge: outermost loop can span millions of symbols – simple algorithms are $O(n^3)$

Common heuristics can prevent long span loops from being discovered

- Compressing a trace window (e.g., first 100K symbols…) at a time
- Reduction of shorter span repeat patterns

This work employs “Crochemore’s algorithm”
Discovering a loop: Terminology

**Repeats** → A set of repeating substrings in a string

Types of Repeats: Consider substring “abca”

**Split Repeats:**  abcaxyzabca

**Overlapping Repeats:**  xyabcaabca

**Tandem Repeats:**  yzabcaabcaab

Loops associated only with Tandem Repeats…
Discovering a loop: PM-Repeats

Loops are Tandem Repeats that are Primal and Maximal aka PM-Repeats

Primal: Not composed of tandem repeats of another substring
Maximal: No identical substring precedes/succeeds

Example string: abababababab
Only PM-repeat → (ab)^4
Not PM-repeat → (abab)^2 -- Not Primal
Not PM-repeat → (ab)^3ab -- Not Maximal

Loop Discovery = Find longest span PM-Repeat
Chrochemore’s algorithm finds all Repeats of all types (split, overlap, tandem) and lengths (levels) in a string with “successive refinement”. Example:

a b a a b a b a a b a a b
1 2 3 4 5 6 7 8 9 10 11 12 13

Level 1: \{1,3,4,6,8,9,11,12\} \{2,5,7,10,13\}

\[ \begin{array}{ll}
\text{Level 2:} & \{1,4,6,9\} \{3,8,11\} \{2,5,7,10\} \\
& \text{ab} \quad \text{aa} \quad \text{ba} \\
& \text{....} \quad \text{.....} \\
\text{Level 7:} & \{1\} \quad \{6\} \quad (\text{no repeats } \geq 7) \\
& \text{abaabab} \quad \text{abaabaa}
\end{array} \]
Loop Nest Discovery Procedure

1. **Repeats Discovery:** All repeats by Crochemore’s algorithms \( O(n \log n) \)
2. **Loop Identification:** Separate out PM-Repeats: (Primal, Maximal, Tandem) \( O(n^2) \)
3. **Loop filtering:** Identify longest span non overlapping PM-Repeats. \( (O(n^2)) \) but very fast in practice

Above steps are recursively repeated for loop bodies to discover full loop nest
Compression by loop nest discovery for communication traces of NAS MPI Benchmarks

<table>
<thead>
<tr>
<th>Bench-mark Name</th>
<th>Trace Length</th>
<th>Compressed Trace Length</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT (B/C)</td>
<td>17106</td>
<td>85</td>
<td>201</td>
</tr>
<tr>
<td>SP (B/C)</td>
<td>26888</td>
<td>162</td>
<td>166</td>
</tr>
<tr>
<td>CG (B/C)</td>
<td>41954</td>
<td>38</td>
<td>1104</td>
</tr>
<tr>
<td>MG (B)</td>
<td>8909</td>
<td>1006</td>
<td>8.9</td>
</tr>
<tr>
<td>MG (C)</td>
<td>10047</td>
<td>1118</td>
<td>9.0</td>
</tr>
<tr>
<td>LU (B)</td>
<td>203048</td>
<td>60</td>
<td>3384</td>
</tr>
<tr>
<td>LU (C)</td>
<td>323048</td>
<td>60</td>
<td>5384</td>
</tr>
<tr>
<td>Average</td>
<td>71695</td>
<td>281</td>
<td>1173</td>
</tr>
</tbody>
</table>
# Loop Discovery Performance Results

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Trace Length</th>
<th>Loop Discovery Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>BT (B/C)</td>
<td>17106</td>
<td>311</td>
</tr>
<tr>
<td>SP (B/C)</td>
<td>26888</td>
<td>748</td>
</tr>
<tr>
<td>CG (B/C)</td>
<td>41954</td>
<td>2022</td>
</tr>
<tr>
<td>MG (B)</td>
<td>8909</td>
<td>113</td>
</tr>
<tr>
<td>MG (C)</td>
<td>10047</td>
<td>145</td>
</tr>
<tr>
<td>LU (B)</td>
<td>203048</td>
<td>44205</td>
</tr>
<tr>
<td>LU (C)</td>
<td>323048</td>
<td>113890</td>
</tr>
</tbody>
</table>

- Processing time over 30 hours for 323K record trace!
- Motivated a Greedy Approach
**Greedy Loop Discovery**

*Poor Performance of “Optimal” loop discovery due to Repeats Discovery and Loops Discovery over very long strings.*

Greedy Approach reduces PM-Repeats as they are discovered – not wait for the longest span PM-Repeats

Optimal: \( \text{aaaabbbccaaaabbbcc} \rightarrow (\text{aaabbbcc})^2 \rightarrow (a^4b^3c^2)^2 \)

Greedy: \( \text{aaaabbbccaaaabbbcc} \rightarrow a^4b^3c^2a^4b^3c^2 \rightarrow (a^4b^3c^2)^2 \)

**Plus:** Much faster as strings processed are much shorter  
**Con:** Long span loops may not be identified correctly

Early reduction is still only for PM-Repeats. Impact on discovery of long span loops is “rare and bounded”
Greedy Experimental Results

- Loops discovered almost identical – one minor difference
- Performance improved multiple orders of magnitude

<table>
<thead>
<tr>
<th>Benchmark Name</th>
<th>Trace Length</th>
<th>Loop Discovery Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Optimal</td>
</tr>
<tr>
<td>BT (B/C)</td>
<td>17106</td>
<td>311</td>
</tr>
<tr>
<td>SP (B/C)</td>
<td>26888</td>
<td>748</td>
</tr>
<tr>
<td>CG (B/C)</td>
<td>41954</td>
<td>2022</td>
</tr>
<tr>
<td>MG (B)</td>
<td>8909</td>
<td>113</td>
</tr>
<tr>
<td>MG (C)</td>
<td>10047</td>
<td>145</td>
</tr>
<tr>
<td>LU (B)</td>
<td>203048</td>
<td>44205</td>
</tr>
<tr>
<td>LU (C)</td>
<td>323048</td>
<td>113890</td>
</tr>
</tbody>
</table>
Algorithmic Results for Greedy

Greedy early reductions can impact discovery of outer loops, however…

**Lemma 1:** Reduction of one “inner loop” can impact the discovery of an “outer loop” only as follows:

1. Loop discovered may be a rotation of the optimal
2. Loop discovered may have up to 2 fewer iterations

Example

Optimal = \( xy(abcd)^{100}bc \)
Greedy = \( xya(bcda)^{99}bcdbc \)
What about the combined effect of many loop reductions at different levels?

(Hiding details and adding handwaving…)

**Lemma 2:** Result very similar to Lemma 1 still holds if the following rule is applied when identifying a new loop for reduction:

>If elements of a new loop can be “rotated” to give the elements of an existing loop, then do it.

Instead of: \( xy(abcd)^{100}(bcda)^{100} \)
Reduce to: \( xy(abcd)^{100}bcd(abcd)^{99}a \)
Conclusions

- New approach to discover loop nests inherent in an execution trace

- Optimal method and greedy method – a key contribution is near optimality of greedy method.

- Diverse application scenarios – implemented as part of skeleton based approach to predicting performance of parallel programs