Efficient Data Race Detection for Async-Finish Parallelism

Raghavan Raman  Jisheng Zhao  Vivek Sarkar
Rice University

Martin Vechev  Eran Yahav
IBM T. J. Watson Research Center
Structured Parallelism

• Many algorithms expressible in structured parallel languages
  – Renewed interest: Java fork-join, MIT’s Cilk, IBM’s X10, DPJ (UIUC), TPL (Microsoft), HJ (Rice)

• Benefits of structured languages
  – Eliminates certain kinds of deadlocks
  – Enables simpler analysis of concurrency
  – Amenable to schedulers with guaranteed space and time bounds
Structured Parallelism

- Programs contain a massive number of tasks
  - Most tasks access disjoint memory locations

- Current data-race detection techniques
  - Focus on unstructured lock-based concurrency
  - Poor space complexity
    - Size of a Vector Clock is proportional to number of threads
  - Inefficient when applied to huge number of tasks
Structured Parallelism: Cilk

• Cilk language constructs
  – spawn
  – sync
  – Induces fully-strict computation graphs (also called Series-Parallel Dags)

• SP-bags: Cilk data-race detection algorithm
  – **Key idea:** Exploits structure for checking
  – Good space complexity – O(1) per memory location, independent of number of tasks
  – Serial algorithm – runs on a single worker thread but reports all possible data races for a given input
SP-bags Revisited

- SP-bags: targets spawn-sync
  - Limitation: Not directly applicable to other constructs

- async-finish
  - More general set of computation graphs than spawn-sync
  - Used as a basis in research projects like IBM X10, Rice HJ, UCLA FX10

- Can SP-bags be extended to async-finish?
Main Contributions

- ESP-bags algorithm: Extending SP-bags to async-finish
- Implementation in tool - TaskChecker
- Static compiler optimizations to reduce runtime overhead
- Evaluation on 12 benchmarks
  - Average slowdown is 3.05x (Geometric Mean)
    - Average slowdown for SP-bags is 7.05x
    - Average slowdown for FastTrack is 6.19x
Task Parallel Extensions

- **async <stmt>:**
  - Creates a new task that can execute <stmt> in parallel with the parent task

- **finish <stmt>:**
  - Blocks and waits for all tasks spawned inside finish scope to complete

- **isolated <stmt>:**
  - Each task must perform an isolated statement in mutual exclusion with any other isolated statements
  - Also called as `atomic’ in X10
Example – Parallel Depth-First Search Spanning Tree

class V {
  V [] neighbors;
  V parent;
  ...
  boolean tryLabeling(V n) {
    isolated if (parent == null) parent = n;
    return parent == n;
  } // tryLabeling

  void compute() {
    for (int i=0; i<neighbors.length; i++) {
      V child = neighbors[i];
      if (child.tryLabeling(this))
        async child.compute(); // escaping async
    }
  } // compute

  void DFS() { // Compute a DFST with “this” as root
    parent = this; // Only the root has parent = itself
    finish compute();
  } // DFS
} // class V
ESP-bags: Extended SP-bags

• Attach two ‘bags’, S and P, to every task instance

• Also attach a P-bag to every finish instance
  – Different from SP-bags

• Each bag holds a set of task ids
  – Task ids are created dynamically

• Attach meta-data to memory locations
  – Each memory location has two fields:
    • reader task id
    • writer task id
  – Can be restricted to two fields per object for object-based race-detection
ESP-bags: Basic Operation

• A serial algorithm:
  – Performs a sequential depth-first execution of the parallel program on a single processor

• Invariant:
  – A task id will always belong to at most one bag at a time

• Space Overhead:
  – Bags represented using a disjoint-set data structure
Contents of S and P bags

• When a statement S in task A is executed, ...

• S-bag of task A
  – holds the task ids of descendent tasks (of A) that always precede the statement S in A

• P-bag of task A
  – holds the task ids of descendent tasks (of A) that may execute in parallel with the statement S in A
Example

\[
\begin{align*}
S_{T_1} &= \{ T_1 \} \\
\mathcal{P}_{T_1} &= \{ \} \\
S_{T_1} &= \{ T_1 \} \\
\mathcal{P}_{T_1} &= \{ T_2 \} \\
S_{T_1} &= \{ T_1, T_3 \} \\
\mathcal{P}_{T_1} &= \{ T_2, T_3 \} \\
S_{T_1} &= \{ T_1, T_2, T_3 \} \\
\mathcal{P}_{T_1} &= \{ \} \\
\end{align*}
\]
Updating the Bags

• Creation of task A (async A)
  – $S_A = \{ A \}$
  – $P_A = \{ \}$

• Execution returns from task A to parent task/finish B
  – $P_B = P_B U S_A U P_A$
  – $S_A = \{ \}$
  – $P_A = \{ \}$

• Start of finish block F
  – $P_F = \{ \}$

• End of finish block F by task B
  – $S_B = S_B U P_F$
  – $P_F = \{ \}$
Checks Performed When Accessing the memory locations

• Read location L by task t
  – If L.writer is in a P-bag, then Data Race
  – If L.reader is in a S-bag, then L.reader = t
  – Otherwise, do nothing

• Write Location L by Task t
  – If L.writer or L.reader is in a P-bag, then Data Race
  – L.writer = t
Example

Reduced version of the example in Figure 2 in the paper
1 final int [] A, B;
2 ...
3 finish {  F1
4    for (int i = 0; i < size; i++) {
5      ...
6        async {  T2
7          B[i] += i;
8          ...
9      } // async
10     finish {  F2
11        async {  T3
12          B[i] = A[i];
13      } // async
14      ...
15     } // finish
16   } // for
17 } // finish

Data Race

Reduced version of the example in Figure 2 in the paper
Performance Optimizations

• ESP-bags Algorithm:
  – Instrument every memory access
  – Some checks are redundant and can be removed

• Read/Write Check Optimization

```javascript
... async {
    ReadCheck (p.x);
    ... = p.x;
    ...
    WriteCheck (p.x);
    p.x = ...;
}
```
More Performance Optimizations

• Check Elimination in Sequential Code Regions

• Read-only Check Elimination in Parallel Regions

• Eliminating Checks on Task-local Objects

• Loop Invariant Check Optimization
Experimental Setup

• 16-way (4x4) Intel Xeon 2.4GHz system
  – 30 GB memory
  – Red Hat Linux (RHEL 5)

• Sun Hotspot JDK 1.6

• All benchmarks written in HJ using only Finish/Async/Isolated constructs
  – JGF benchmarks used with their highest input size
    • Except MolDyn for which size A was used
  – http://habanero.rice.edu/hj

• ESP-bags algorithm – implemented in a tool called TaskChecker, along with the optimizations
# Slowdown of ESP-bags Algorithm

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Number of Tasks</th>
<th>Original Time (s)</th>
<th>ESP-bags Slowdown Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>w/o Opts</td>
</tr>
<tr>
<td>Crypt</td>
<td>1.3e7</td>
<td>15.24</td>
<td>7.63</td>
</tr>
<tr>
<td>LUFact</td>
<td>1.6e6</td>
<td>15.19</td>
<td>12.45</td>
</tr>
<tr>
<td>MolDyn</td>
<td>5.1e5</td>
<td>45.88</td>
<td>10.57</td>
</tr>
<tr>
<td>MonteCarlo</td>
<td>3.0e5</td>
<td>19.55</td>
<td>1.99</td>
</tr>
<tr>
<td>RayTracer</td>
<td>5.0e2</td>
<td>38.85</td>
<td>11.89</td>
</tr>
<tr>
<td>Series</td>
<td>1.0e6</td>
<td>1395.81</td>
<td>1.01</td>
</tr>
<tr>
<td>SOR</td>
<td>2.0e5</td>
<td>3.03</td>
<td>14.99</td>
</tr>
<tr>
<td>SparseMatMult</td>
<td>6.4e1</td>
<td>13.59</td>
<td>12.79</td>
</tr>
<tr>
<td>Fannkuch</td>
<td>1.0e6</td>
<td>7.71</td>
<td>1.49</td>
</tr>
<tr>
<td>Fasta</td>
<td>4.0e0</td>
<td>1.39</td>
<td>3.88</td>
</tr>
<tr>
<td>Mandelbrot</td>
<td>1.6e1</td>
<td>11.89</td>
<td>1.02</td>
</tr>
<tr>
<td>Matmul</td>
<td>1.0e3</td>
<td>19.59</td>
<td>6.43</td>
</tr>
<tr>
<td><strong>Geometric Mean</strong></td>
<td></td>
<td></td>
<td><strong>4.86</strong></td>
</tr>
</tbody>
</table>
## Comparison with Other Race Detectors

<table>
<thead>
<tr>
<th>Properties</th>
<th>SP-bags</th>
<th>ESP-bags</th>
<th>FastTrack</th>
</tr>
</thead>
<tbody>
<tr>
<td>Language</td>
<td>Cilk</td>
<td>X10/HJ</td>
<td>Java</td>
</tr>
<tr>
<td>Number of Test Programs</td>
<td>8</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>Minimum Slowdown</td>
<td>2.41</td>
<td>1</td>
<td>0.9</td>
</tr>
<tr>
<td>Maximum Slowdown</td>
<td>11.09</td>
<td>10.08</td>
<td>14.8</td>
</tr>
<tr>
<td>Average Slowdown (Geometric Mean)</td>
<td>7.05</td>
<td>3.05</td>
<td>6.19</td>
</tr>
<tr>
<td>Space Overhead</td>
<td>O(1)</td>
<td>O(1)</td>
<td>O(n)</td>
</tr>
<tr>
<td>Serial/Parallel</td>
<td>Serial</td>
<td>Serial</td>
<td>Parallel</td>
</tr>
<tr>
<td>Guarantees</td>
<td>Per-Input</td>
<td>Per-Input</td>
<td>Per-Execution</td>
</tr>
<tr>
<td>Schedule Dependent?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Breakdown of Optimizations

[Bar chart showing slowdown factor for different optimizations and benchmarks, including No Opt, Read-only Opt, Escape Opt, LICM Opt, RW Opt, and Full Opt.]
Extensions for Isolated

• Need to check that isolated and non-isolated accesses that may execute in parallel do not conflict

• Two additional fields
  – Can be restricted to memory locations accessed in isolated blocks

• Refer to the paper for more details
Determinism vs. Data Races

• Important for structured parallel algorithms
  – Intended to be deterministic

• Data-race freedom implies determinism in some cases (only with async-finish)
  – Implies temporal and spatial disjointness

• Algorithm dynamically checks for determinism
  – Guarantees per-input, not just per-execution
Conclusions

• ESP-bags algorithm: Extending SP-bags to enable Data-race and Determinism Analysis for async-finish

• Implementation in tool - TaskChecker

• Static Compiler Optimizations reduce overhead

• Average slowdown of 3.05x on a suite of 12 benchmarks
Future Work

• Extend ESP-bags to support more constructs:
  – HJ: futures, phasers
  – X10: futures, clocks, conditional atomics
  – Java: thread fork/join, Java Concurrency Utilities

• Parallelize ESP-bags

• Combine with static determinism verification:
  
  *Automatic Verification of Determinism for Structured Parallel Programs (SAS’10)*
Rice Habanero Multicore Software Project: Enabling Technologies for Extreme Scale

Parallel Applications

**Portable execution model**
1) Lightweight asynchronous tasks and data transfers
   - `async`, `finish`, `asyncMemcpy`
2) Locality control for task and data distribution
   - `hierarchical place tree`
3) Mutual exclusion
   - `ownership-based isolation`
4) Collective, point-to-point, stream synchronization
   - `phasers`

**Two-level programming model**
- Declarative Coordination Language for Domain Experts, CnC (Intel Concurrent Collections)
- Task-Parallel Languages for Parallelism-aware Developers, Habanero-Java (from X10 v1.5) and Habanero-C

**Habanero Programming Languages**

**Habanero Static Compiler & Parallel Intermediate Representation**

**Habanero Runtime and Tools**

Includes TaskChecker tool!

Extreme Scale Platforms

http://habanero.rice.edu