Lightweight Fault Detection in Parallelized Programs

Li Tan
UC Riverside

Min Feng
NEC Labs

Rajiv Gupta
UC Riverside

CGO'13, Shenzhen, China
Feb. 25, 2013
Program Parallelization

- Parallelism can be achieved via parallelization of sequential programs via easy-to-use parallel constructs $\rightarrow$ OpenMP, SpiceC [PPoPP’11], and TBB.

- Data Dependence Related Concurrency Bugs
  - Data races
  - Atomicity violations
Comparison Checking

- Conventional [PACT’98] [FSE’99]
  - Locate program faults by leveraging the availability of two versions of a program – one supposed correct version and one derived version

- In Our Scenario
  - A sequential version $S$ and a parallelized version $P$
  - Faulty parallelization $\rightarrow$ data dependence violation
    - Data dependences enforced by $S$ are not preserved by $P$
Debugging *Parallelized* Programs

**Basic Idea:**

- Comparison check the data dependences exercised by the executions of $S$ and $P$
  - dynamic *Data Dependence Graphs*: $sDDG + pDDG$
    - Nodes: execution instances of statements
    - Edges: data dependences between nodes
  - Faulty parallelization: *different* $sDDG$ and $pDDG$ constructed using the *same* input
#pragma omp parallel for
for (;;) {
    #pragma omp critical
    if (fgets(s, NUM, stdin) == NULL) break;
    if (s[0] == '!') {
        /* Other relevant code here */
    }
    if (!separate_sentence(s)) {
        /* Other relevant code here */
    }
}
Limitations

- DDG Construction Overhead

Execution Time

Memory Space
Limitations (Cont.)

- Graph Size and Checking Time

![Graph showing the sum of sDDG and pDDG Graph Size for various benchmarks](image)
Limitations (Cont.)

- Dependence Violation May Not Occur
  - Not every interleaving causes violation
  - As low as 10% chance to expose a data race; up to 22 hours to expose an atomicity violation [ASPLOS’09]

- Validity of Comparing Two Runs
  - Random numbers alter control flow [ISSTA’07]
  - Inconsistency ≠ a concurrency bug
Significant limitations…
How can we get rid of them?
**OPT-1: Region Graphs**

- Eliminate *irrelevant* dependences
  - Data dependences in sequentially executed code
  - **Savings**: time + space for tracking and checking

- *fine*-grained graphs $\rightarrow$ *coarse*-grained graphs
  - Statements (DDG) $\rightarrow$ Code regions (DRG)
  - **Savings**: graph size
OPT-1: Region Graphs (Cont.)

```c
#pragma omp parallel for
for (;;) {
    if (fgets(s, NUM, stdin) == NULL) break;
    if (s[0] == '!') {
        /* Other relevant code here */
    }
    if (!separate_sentence(s)) {
        /* Other relevant code here */
    }
}
```

/* the same data race example... */
**OPT-2: Summarize Region Instances**

- A single node in a DRG represents all execution instances of a region
  - Different dependences need to be distinguished
  - **Savings**: time for tracking and checking + graph size

- Annotate each edge by *dependence distances*
  - 0 indicates an *intra-iteration* dependence
  - A non-zero value indicates a *cross-iteration* dependence
OPT-2: Summarize Region Instances

```
#pragma omp parallel for
for (;;) {
    if (fgets(s, NUM, stdin) == NULL) break;
    if (s[0] == '!') {
        /* Other relevant code here */
    }
    if (!separate_sentence(s)) {
        /* Other relevant code here */
    }
}
```

/* the same data race example... */
OPT-3: Static Region Graph

- Only the sequential version needs to be run
  - Staticaly analyzing the parallel constructs in the parallelized version
  - **Savings**: time + space for tracking and checking

- Simplified concurrency bug detection
  - Check if data dependences allowed by OpenMP, SpiceC, and TBB violate sequential semantics
  - **Eliminate the limitations**:
    - Reproducibility rate + validity of comparing two runs
**OPT-3: Static Region Graph (Cont.)**

<table>
<thead>
<tr>
<th>Construct</th>
<th>Allowed Dependences</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpenMP</strong></td>
<td></td>
</tr>
<tr>
<td>parallel [for</td>
<td>do]</td>
</tr>
<tr>
<td>section</td>
<td>Intra-iteration dependences</td>
</tr>
<tr>
<td>critical</td>
<td>Intra-iteration/cross-iteration dependences</td>
</tr>
<tr>
<td>ordered</td>
<td>Intra-iteration/cross-iteration dependences</td>
</tr>
<tr>
<td><strong>SpiceC</strong></td>
<td></td>
</tr>
<tr>
<td>doall</td>
<td>Intra-iteration dependences</td>
</tr>
<tr>
<td>doacross</td>
<td>Intra-iteration dependences</td>
</tr>
<tr>
<td>pipelining</td>
<td>Intra-iteration/cross-iteration dependences</td>
</tr>
<tr>
<td>after(ITER-x, R_y)</td>
<td>Intra-iteration/cross-iteration dependences from region R_y to current region with a distance x</td>
</tr>
<tr>
<td>atomicity_check</td>
<td>Intra-iteration/cross-iteration dependences</td>
</tr>
<tr>
<td><strong>TBB</strong></td>
<td></td>
</tr>
<tr>
<td>parallel_for</td>
<td>Intra-iteration dependences</td>
</tr>
<tr>
<td>parallel_reduce</td>
<td>Intra-iteration dependences and cross-iteration dependences of <code>join</code></td>
</tr>
<tr>
<td>parallel_scan</td>
<td>Intra-iteration dependences and cross-iteration dependences of <code>reverse_join</code></td>
</tr>
<tr>
<td>parallel_pipeline</td>
<td>Intra-iteration dependences and cross-iteration dependences of <code>filter</code></td>
</tr>
</tbody>
</table>
OPT-3: Static Region Graph (Cont.)

```c
#pragma omp parallel for
for (;;) {
    if (fgets(s, NUM, stdin) == NULL) break;
    if (s[0] == '!') {
        /* Other relevant code here */
    }
    if (!separate_sentence(s)) {
        /* Other relevant code here */
    }
}
```

/* the same data race example... */
## Optimizing Comparison Checking

<table>
<thead>
<tr>
<th>Optimization</th>
<th>Execution Time</th>
<th>Memory Space</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tracking</td>
<td>Checking</td>
<td>Tracking</td>
<td>Graph Size</td>
</tr>
<tr>
<td><strong>OPT-1</strong></td>
<td>√</td>
<td>√</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td><strong>OPT-2</strong></td>
<td>√</td>
<td>√</td>
<td></td>
<td>√</td>
</tr>
<tr>
<td><strong>OPT-3</strong></td>
<td>√</td>
<td></td>
<td></td>
<td>√</td>
</tr>
</tbody>
</table>
Observation: A dependence present in sDRG, but not allowed by SRG, represents violation against sequential program semantics by parallelization expressed by parallel constructs.
Evaluation

- **Benchmarks**
  - Applied our technique to ten benchmarks parallelized using OpenMP, SpiceC, and TBB
  - Selected from MiBench, SPEC CPU2000, Lonestar, and PARSEC benchmark suites

- **Hardware Configuration**
  - A 2.66 GHz Intel Core Duo DELL Dimension 9200 machine with 4 GB RAM
  - Linux kernel 2.6.32
sDRG Construction Overhead

Execution Time

Memory Space
Graph Size Comparison

Previously

Now
# Comparison of DRG and DDG

## Cost of Using DRG as a Percentage of DDG

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Execution Time</th>
<th></th>
<th>Memory Space</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Tracking</td>
<td>Checking</td>
<td>Tracking</td>
<td>Graph Size</td>
</tr>
<tr>
<td>bodytrack</td>
<td>4.487%</td>
<td>0.371%</td>
<td>45.765%</td>
<td>0.049%</td>
</tr>
<tr>
<td>freqmine</td>
<td>14.416%</td>
<td>0.586%</td>
<td>51.985%</td>
<td>0.082%</td>
</tr>
<tr>
<td>256.bzip2</td>
<td>17.594%</td>
<td>0.623%</td>
<td>46.155%</td>
<td>0.584%</td>
</tr>
<tr>
<td>CRC32</td>
<td>22.854%</td>
<td>0.694%</td>
<td>47.240%</td>
<td>1.923%</td>
</tr>
<tr>
<td>Barnes-Hut</td>
<td>5.819%</td>
<td>1.519%</td>
<td>32.463%</td>
<td>0.139%</td>
</tr>
<tr>
<td>197.parser</td>
<td>9.474%</td>
<td>2.632%</td>
<td>34.419%</td>
<td>0.138%</td>
</tr>
<tr>
<td>ferret</td>
<td>12.713%</td>
<td>1.067%</td>
<td>33.196%</td>
<td>0.044%</td>
</tr>
<tr>
<td>DelaunayRefinement</td>
<td>33.483%</td>
<td>0.371%</td>
<td>71.006%</td>
<td>0.051%</td>
</tr>
<tr>
<td>swaptions</td>
<td>32.437%</td>
<td>1.976%</td>
<td>36.379%</td>
<td>0.494%</td>
</tr>
<tr>
<td>streamcluster</td>
<td>9.884%</td>
<td>1.282%</td>
<td>39.001%</td>
<td>0.064%</td>
</tr>
<tr>
<td><strong>GeoMean</strong></td>
<td><strong>13.463%</strong></td>
<td><strong>0.907%</strong></td>
<td><strong>42.534%</strong></td>
<td><strong>0.151%</strong></td>
</tr>
</tbody>
</table>
Breakdown of Overhead Reduction
Conclusions

• Debugging Parallelized Programs (OpenMP, SpiceC, and TBB)

Versatility

• Support for multiple types of concurrency bugs
• Support for multiple parallel programming models

Novelty

• No requirement for execution of parallel programs
• Elimination of reproducibility and validity problems
• Region level data dependence graphs
• Only 3x slowdown on average