ARCHER: Effectively Spotting Data Races in Large OpenMP Applications

Ignacio Laguna, D. H. Ahn, G. L. Lee, M. Schulz (LLNL)
S. Atzeni, G. Gopalakrishnan, Z. Rakamarić (Univ. of Utah)
J. Protze (RWTH Aachen)

March 17-18, 2016
OpenMP is Widely Used in HPC Applications

- Standard to expresses parallelism in multi-threaded code
- Parallel code can be executed in CPUs or accelerators (e.g., GPUs)
- As in all multi-threaded programming models, data races can occur

```c
void simple(int n, float *a, float *b)
{
    int i;
    #pragma omp parallel for
    for (i=1; i<n; i++)
        b[i] = (a[i] + a[i-1]) / 2.0;
}
```
Identifying data races in large OpenMP applications is challenging

- **Scalability** is key
- **Accurate** and **precise** detection is very important
- **Low overhead** and **portability** allow adoption in practice

<table>
<thead>
<tr>
<th>Tool</th>
<th>Technology</th>
<th>Accuracy/Precision</th>
<th>Overhead</th>
<th>Portability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel SSA</td>
<td>Static</td>
<td><img src="red" alt="Pthread" /> <img src="yellow" alt="OpenMP" /></td>
<td><img src="green" alt="Portability" /></td>
<td><img src="green" alt="Portability" /></td>
</tr>
<tr>
<td>Intel Inspector</td>
<td>Dynamic</td>
<td><img src="green" alt="Pthread" /> <img src="yellow" alt="OpenMP" /></td>
<td><img src="red" alt="Overhead" /> <img src="red" alt="Portability" /></td>
<td><img src="red" alt="Portability" /></td>
</tr>
<tr>
<td>Helgrind</td>
<td>Dynamic</td>
<td><img src="green" alt="Pthread" /> <img src="red" alt="OpenMP" /></td>
<td><img src="red" alt="Porthead" /> <img src="red" alt="Portability" /></td>
<td><img src="red" alt="Portability" /></td>
</tr>
<tr>
<td>ThreadSanitizer</td>
<td>Dynamic</td>
<td><img src="green" alt="Pthread" /> <img src="red" alt="OpenMP" /></td>
<td><img src="yellow" alt="Porthead" /> <img src="green" alt="Portability" /></td>
<td><img src="green" alt="Portability" /></td>
</tr>
</tbody>
</table>
Combines static and dynamic techniques in a single tool

Build on top of:
- Polly for static dependence analysis
- ThreadSanitizer – dynamic happens-before analysis
- LLVM/Clang OpenMP runtime
Approach – Static Analysis Phase
Target instrumentation on a simple OpenMP program

```c
1 main() {
2     // Serial code
3     setup();
4     sort();
5
6     #pragma omp parallel for
7     for(int i = 0; i < N; ++i) {
8         a[i] = a[i] + 1;
9     }
10
11    #pragma omp parallel for
12    for(int i = 0; i < N - 1; ++i) {
13        a[i] = a[i + 1];
14    }
15
16    #pragma omp parallel
17    {
18        sort();
19    }
20
21     // Serial code
22     printResults();
23 }
```

Serial code blacklisted
Used in serial and parallel code
No data dependency code blacklisted
Potentially racy code instrumented
Potentially racy code instrumented
Serial code blacklisted
# Evaluation: OmpSCR Benchmarks

<table>
<thead>
<tr>
<th>Application</th>
<th>Inspector – Default</th>
<th>Inspector – ExtremeScope</th>
<th>Archer (no SA)</th>
<th>Archer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slowdown Mean</td>
<td>29.5</td>
<td>30.3</td>
<td>122.4</td>
<td>19.6</td>
</tr>
<tr>
<td>Geometric Mean</td>
<td>18.3</td>
<td>20.2</td>
<td>71.5</td>
<td>10</td>
</tr>
<tr>
<td>Detected Races</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>False Alarms</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

The total number of races in the benchmarks is 12.