Archer: A Low Overhead Data Race Detector for OpenMP

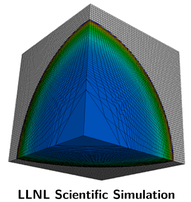
Simone Atzeni, Ganesh Gopalakrishnan (Advisor), Zvonimir Rakamarić (Advisor)

School of Computing, University of Utah

Text

Motivation

LLNL Sequoia IBM Blue Gene/Q



LLNL Scientific Simulation

Large-scale multiphysics applications are hard to verify

- Numerous sources of program-specific non-determinism
- OpenMP data races are particularly challenging to identify
- Errors remain masked unless triggered by inputs/schedule/scale
- Run-time analyses incur 100x slowdown and 8x memory overheads
- Static analyses suffer from low accuracy (e.g., false positives)

Need efficient verification/diagnostic tools for large OpenMP applications!

Contributions and Work in Progress

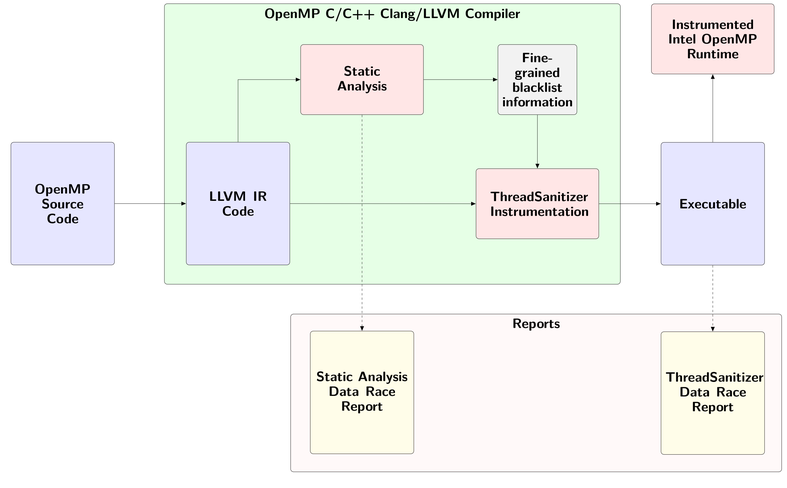
- Static analysis techniques that enhance data race detection
- Comparative study of race checkers for PThreads and OpenMP
- Combination of static and dynamic analysis techniques
- Static analysis helps triage races into likely categories
- Dynamic analysis does not generate false positives
- Decrease number of false positives and omissions
- Low overhead verification run-time

State of the Art

- **Intel Static Security Analysis**
  - \+ Good performance
  - \+ Supports OpenMP
  - \- False positives
  - \- False negatives on some race types
- **Intel Inspector**
  - \+ High accuracy and precision
  - \+ Supports OpenMP
  - \- Very high time overhead
- **ThreadSanitizer**
  - \+ High accuracy and precision
  - \+ Faster than other existing tools
  - \+ Targeted verification of code regions
  - \- Does not meaningfully support OpenMP
  - \- Checks every memory access
  - \- High memory overhead

How to develop an Accurate OpenMP Data Race Checker that works on practical HPC Codes?

Approach



Fine-grained targetting of suspicious code segments

- Static analysis
  - Data dependency
  - Loop-carried data dependency
  - Thread escape
- Warning about possible data races: need to be checked at run-time

Archer

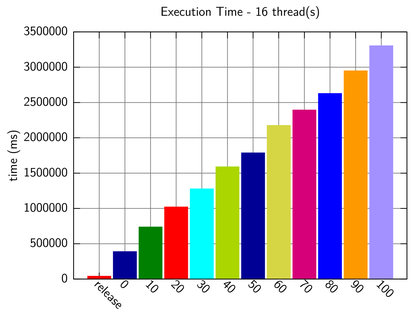
- Based on ThreadSanitizer (mature technology)
- Uses information generated by static analysis (avoids redundant checks)
- Fine-grained blacklist instrumentation (line of code resolution)
- Targets code likely affected by races (enhanced efficiency and accuracy)
- Reports data race found at runtime (less false positives)

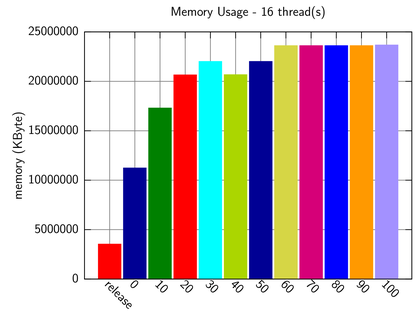
Intel OpenMP Runtime

- Instrumented to enable ThreadSanitizer to be applied to OpenMP programs

Preliminary Results

Feasibility Evaluation on AMG2013 CORAL Benchmark





Archer on Lulesh 2.0 Racing version

Static Analysis

```
#pragma omp parallel for
  for(int i = 0; i < numElem; ++i) {
    domain.fx(i) = fx_tmp;
    domain.fy(i) = fy_tmp;
    domain.fz(i) = fz_tmp;
  }
```

→ **No need to be checked at run-time**

Figure 1: Guarantee race freedom

Verified at run-time ←

```
  int index = elemToNode[0];
#pragma omp parallel for
  for(int i = 0; i < numElem; ++i) {
    domain.fx(index) += hgfx[0];
    domain.fy(index) += hgfy[0];
    domain.fz(index) += hgfz[0];
  }
```

Figure 2: Discover potential data races

Dynamic Analysis

```
WARNING: ThreadSanitizer: data race (pid=41317)
  Write of size 8 at 0x7d6c0001a478 by main thread:
    #0 .omp_microtask.39 lulesh.cc:1017 (lulesh_tsan_bl+0x000000094d60)
    #1 __kmp_invoke_microtask <null>:0 (libiomp5_tsan.so+0x0000000b9cd2)
    #2 __libc_start_main libc-start.c:226 (libc.so.6+0x00000001ed1c)

  Previous write of size 8 at 0x7d6c0001a478 by thread T2:
    #0 .omp_microtask.39 lulesh.cc:1021 (lulesh_tsan_bl+0x000000094e20)
    #1 __kmp_invoke_microtask <null>:0 (libiomp5_tsan.so+0x0000000b9cd2)
   ...
SUMMARY: ThreadSanitizer: data race lulesh.cc:1017 .omp_microtask.39
==================
ThreadSanitizer: reported 67 warnings
```

Figure 3: Archer Data Race Report of code in Figure 2

Future Work

Archer Tool Development

- Integrate and/or combine static analysis techniques in LLVM
  - Reduce omission of races, and highlight race-free code regions
  - Provide useful source-level debugging information pertaining to races
- Develop Customized OpenMP Data Race Checking Algorithms

Performance Tests

- Attain high performance on real OpenMP program

Further Information

Simone Atzeni
Ph.D. Student
address:
School of Computing
University of Utah
50 S. Central Campus Drive
Salt Lake City, UT 84112
e-mail:
simone@cs.utah.edu
atzeni1@llnl.gov

Acknowledgements

I would like to thank Dong H. Ahn (Advisor at LLNL), Ignacio Laguna (Advisor at LLNL), Greg Lee and, Martin Schulz from Lawrence Livermore National Laboratory for the advice and the support during my internship at LLNL. Furthermore, I want to express my appreciation to Joachim Protze from RWTH Aachen University for his work on the Intel OpenMP Runtime.

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.

LLNL-POST-657845