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
- Errors remain masked unless triggered by inputs/schedule/scale
- OpenMP data races are particularly challenging to identify
- Numerous sources of program-specific non-determinism

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
  - Does not meaningfully support OpenMP
  - Faster than other existing tools
  - Targeted verification of code regions
  - Checks every memory access
  - High memory overhead

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

Motivation

OpenMP C/C++ Clang/LLVM Compiler

Static Analysis

Finite-grained targeting 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)
  • Finite-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

Approach

Preliminary Results

Archer on Lulesh 2.0 Racing version

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

Intel OpenMP Runtime
  \begin{itemize}
    \item Reduce omission of races, and highlight race-free code regions
    \item Provide useful source-level debugging information pertaining to races
    \item Develop Customized OpenMP Data Race Checking Algorithms
  \end{itemize}

Further Information

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.

LLNL Scientific Simulation

Archer: A Low Overhead Data Race Detector for OpenMP

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