Corvette: Program Correctness, Verification and Testing for Exascale

Koushik Sen and James Demmel (UC Berkeley) Costin Iancu (LBNL)

Project website: https://crd.lbl.gov/organization/computer-and-data-sciences/future-technologies/projects/corvette

1. Motivation

- High performance scientific computing
  - Exascale: O(10^6) nodes, O(10^3) cores per node
  - Productivity requires asynchrony and "relaxed" memory consistency
  - Non-deterministic execution is likely to cause hard to diagnose correctness and performance bugs
- Limited usage of testing and correctness tools
  - Tools are hard to find and use for HPC
  - Tools for multi-threaded programs ported to distributed systems do not scale
- Scientific applications are also difficult to debug
  - Floating-point programs in particular
  - Numerical exceptions (anomalies) can cause rare but critical bugs that are hard for non-experts to detect and fix

2. Goals

- Develop correctness tools for a large spectrum of programming and execution models: PGAS, MPI, dynamic parallelism
  - Explore state-of-the-art techniques that use dynamic analysis
  - **Develop precise tools with no more than 2x run-time overhead at large scale**
- Novel approaches to assist with debugging
  - Use minimal amount of concurrency to reproduce bug
  - Support two-level debugging of high-level abstractions
  - Detect causes of floating-point anomalies and determine the minimum precision needed to fix them
- Reproducible concurrency bugs and floating-point behavior
  - Identify sources of non-determinism in executions
  - Concurrency bugs include data races, atomicity violations, and deadlocks

3. Distributed Memory Support

Example: Dynamic Analysis of UPC Programs

(See our SC'11 paper)

- Efficient Data Race Detection for UPC
  - Thread Interposition Library and Lightweight Extensions
  - Framework for active testing UPC programs
- Instrument UPC source code at compile time
  - Using macro expansions, add hooks for analyses
- Phase 1: Race detector
  - Observe execution and predict which accesses may potentially have a data race
  - Filtering and sampling to reduce communication costs
- Phase 2: Race tester
  - Re-execute program while controlling the scheduler to create actual data race scenarios predicted in phase 1

Message Passing Concurrency

- MPI is ubiquitous
  - MPI processes communicate with messages only
  - Usually no data races between processes
- Data races may occur
  - Between local memory accesses and communication operations (ISend/IRecv)
  - Between communication operations (Send/Recv)
- MPI-3 introduces one-sided communication and non-blocking collectives

Extensions to Hybrid Programming Models

- PGAS (or MPI) for inter-node, shared-memory for intra-node
- Challenge: **tracing all memory accesses**
  - Research heuristics for reducing impact of instrumentation

4. Floating-Point Support

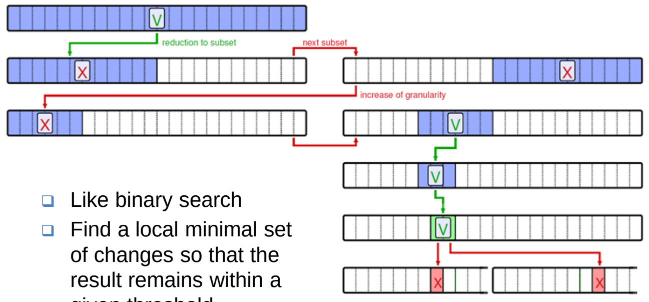
Common Anomalies

- Rounding error accumulations
- Conditional branches involving floating-point comparisons
  - May go astray due to the subtleties of floating-point arithmetic (e.g., NaN values)
  - Convergence misbehavior
- Under/overflows, resolution of ill-conditioned problems
  - Result may be wrong
- Benign vs. catastrophic cancellations

Current Techniques for Finding Anomalies

- Altering rounding mode of floating-point arithmetic hardware
  - May not normally be usable to remedy the problems
- Extending precision of floating-point computation
  - May increase run-time significantly
- Using interval arithmetic
  - Produces a certificate, but run-time cost is the greatest

Approach: Automated Delta Debugging



- Like binary search
- Find a local minimal set of changes so that the result remains within a given threshold