Efficient and scalable data race detector and tester for a SPMD + PGAS language

**Motivation**

- Unified Parallel C (UPC)
  - Parallel extension of C
  - SPMD + PGAS
  - Limited tool support for correctness
- Build tool for testing / debugging UPC programs
  - Scalable & efficient
  - Few false positives
  - Reproducible
- Unique challenges for UPC
  - Different programming model (SPMD)
  - Barrier synchronization (including split-phase)
  - Bulk communication and collective operations
  - Large scale on multiple nodes
  - Distributed memory ordering and consistency

**Active Testing**

- Leverage program analysis to make testing quickly find real concurrency bugs
  - Phase 1: Use imprecise static or dynamic program analysis to find bug patterns where a potential concurrency bug can happen
  - Phase 2: Directed testing to confirm potential bugs as real
- Our previous work
  - CalFuzzer for Java
  - Thrille for C/C++ with pthreads

**Race Detection**

- Store shared memory access information locally
  - e = (m, t, a, p, s) for each shared memory access
  - Using efficient data structures (see "Data Layout")
  - Keep only the weakest accesses
- At barrier boundary, send access info to “owner” thread
- For each access e from other threads,
  - Find overlapping intervals in IS-List
  - For each interval,
    - Check for existing weaker accesses (ignore e if found)
    - Check for racing accesses (report if found)
    - Add e to IS-List, and remove stronger accesses

**Data Layout**

- e1 ⊆ e2 (access e1 is weaker-than e2) iff
  - e1.m ⊆ e2.m
  - e1.t = * V e1.t = e2.t
  - e1.L ⊆ e2.L
  - e1.a = Write ∨ e1.a = e2.a

**Weaker-than Relation**

**May-happen-in-parallel**

**Race Reproduction**

- Consider each race pair from phase 1: (s₁, s₂)
- Control the scheduler as follows for each access:
  - Before T1 executes s₁, block until some other thread executes s₂
  - When a matching s₂ has overlapping address, report real race
  - Resolve race in both orders (s₁, s₂) (weaker access type)

**Results**

- Low overhead for data race detection
- Shows scalability on small clusters with room for improvement
- Approximate record and replay
- Non-blocking transfers of analysis data among threads

**Future Work**

- Improve scalability by reducing communication
- Non-blocking transfers of analysis data among threads
- Dynamic exponential back-off for code “hot spots”
- Additional analyses
  - Memory consistency analysis
  - Deadlock analysis
  - Approximate record and replay