Verifying Qthreads: Is Model Checking viable for User-Level Tasking runtimes?

Noah Evans

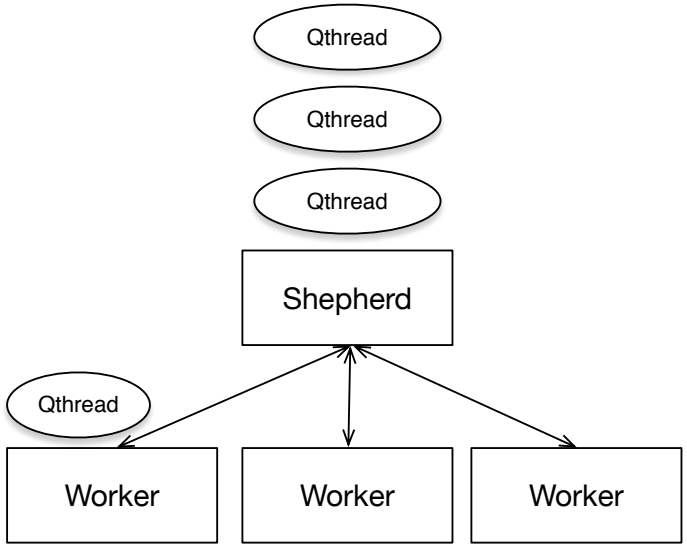
Center for Computing Research

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.
(*SAND2018-11134 C*)

Background

1. Extreme scales -> management of parallelism progressively harder and more important.
2. Being able to automatically parallelize applications == grand challenge.
3. Task parallel gets a little closer.
4. Break up the app into smallest pieces -> schedule with global runtime.
5. lots of context switches so implemented as user level threads with scheduling in user space.

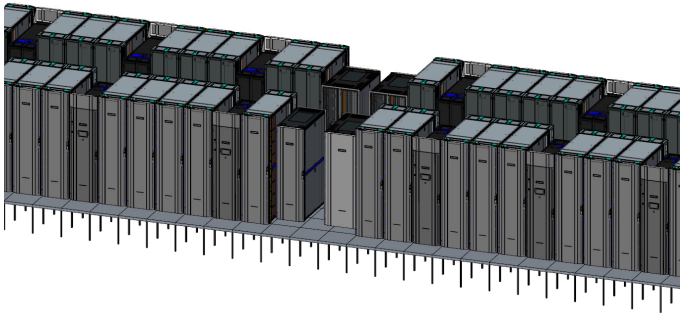
CCR
Center for Computing Research



Qthreads

1. models concurrency non uniform work loads and memory.
2. FEBs (full empty bits)
   1. CSP channels, but word sized
   2. Sleep until someone writes.
3. non uniform memory aware scheduling
   1. NUMA regions as well as cores

CCR
Center for Computing Research



Users

1. Default thread model for Cray's Chapel language.
2. An execution engine in Kokkos.
3. Part of the system software suite in Sandia's astra.
4. Being integrated into OpenMPI.

Qthreads Problem: New Architectures, Old code

1. Good news: new challenges with modern architectures.
   - Relaxed memory semantics in Power and Arm.
   - Qthreads' FEB and NUMA aware semantics potentially a good fit.
2. Bad news:
   - Half finished research features can't be conditionally compiled (i.e. unifdefed) out.
   - X86 centric lingering race conditions.

CCR
Center for Computing Research

Potential Solution: Verify Qthreads

1. Idea: model checking initially, and deep spec later.
2. Qthreads model
   A. New Qthreads behavior without cruft into the code base.
   B. Test across memory models (c.f. Abe et al.)
3. Use model as basis for a “deep specification”

CCR
Center for Computing Research

Initial Feasibility Study: Use the Spin Model Checker

1. SPIN's sequential consistency to start
2. No one to one correspondence with the code base (no Modex, CIVL etc...)
3. Can Spin verify a large tasking runtime like Qthreads?

CCR
Center for Computing Research

Three Problems: Construction, Scale, Practicality

1. Possible to take a C runtime like qthreads and model it?
   1. Construction: Semantic mismatches, pointers especially function pointers, model the loading of programs?
   2. Scale to a reasonable number of threads?
   3. Practical? I.e. does it tell us anything useful?

Construction

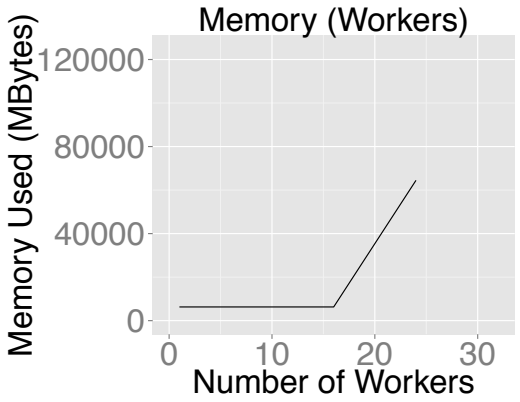
1. Pointers -> array accesses.
    1. Memory “pool” which is allocated and return numbers that indirect into the array.
2. Function -> Inlines
    1. Add output values to function calls
    2. Gotos to simulate returns
3. Function Pointers -> Defunctionalized table
    1. Manually turn function pointers into table lookups

CCR
Center for Computing Research

Scale

1. Experimental Setup: 64 core Haswell, 132 GB ram
2. Bit-state hashing
3. Research question: How many workers and shepherds can we scale to?

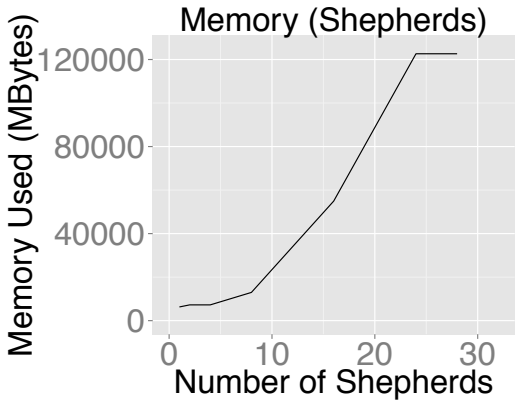
CCR
Center for Computing Research

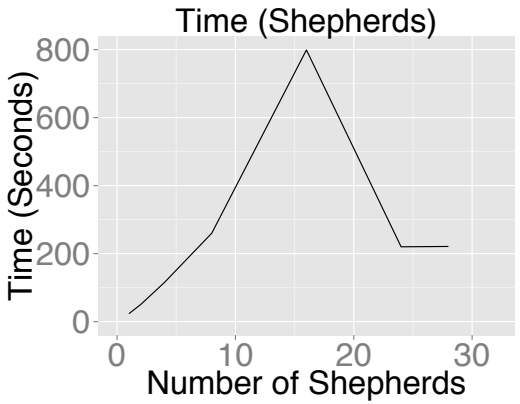


Workers are Compute Bound

1. Workers are compute bound, get to 24 before hitting unreasonable amounts (> 1 hour) of time to verify.

CCR
Center for Computing Research





Shepherds are Memory Bound

1. Blows up SPINs memory after 16 shepherds.
   - 800 seconds to success, fails quickly later
2. 16 Shepherds still sufficient to explore interesting problems

CCR
Center for Computing Research

Practicality: SPIN catches a graverobber!

1. Concurrency bug, appears after 2 tasks.
2. Scheduler steals a terminated task off the queue.
3. Assert fails -- can't steal dead task (no graverobbing allowed!)
4. May be a modeling error (more likely an untriggered race).
5. Still shows us that SPIN can reason about scheduling task runtimes.

CCR
Center for Computing Research

Related Work

1. Model checking for MPI (MPI-SPIN)
2. Automated Model Extraction (CIVL, Modex)
3. Tasking runtime verification (Legion's operational semantics)
4. Our approach: first tasking runtime with a checked model.

Future Work

1. Use SPIN variants to explore alternative memory models
   - especially ARM
2. Scale using distributed approaches
   - (e.g. Multiple hash functions for bitstates).
3. Finish implementing the rest of Qthreads model
   - Different schedulers.
   - Yield semantics.
4. Extend to other runtimes (Argobots, Massivethreads?)?

CCR
Center for Computing Research

Conclusions

1. Tasking models way to handle heterogeneity and parallelism at extreme scale.
2. Qthreads has contributions to make in this tasking space
3. Need new techniques to deal with parallelism and relaxed memory models
   - formal verification can help here.
4. It is feasible and practical to verify Qthreads semantics in SPIN.
   - Reasonable size models tell us about the runtime behavior.
5. Next, look at relaxed memory models and scaling up.

Questions?

CCR
Center for Computing Research