Notes on DXExMachina performance.

November 8, 2015

1 Accompanying documents

Benchmarks (on master) in separate pdf. Further benchmarks and profiling data in repository.

2 Improvements since abstract

- Conservative handles cyclic simulations.
- Scheduler ported from Adevs + own heuristic.
- Memory pooling.

  Aligning messages on cache lines to avoid false sharing

  Thread local pools to avoid expensive cross-thread alloc/dealloc.

  Gvt required for conservative, based on existing null message exchange.
- Frequency of locking reduced, (sparse) usage of memory_order_relaxed. Where possible explicit locking switched to atomic.
- A (large) nr of bugfixes and code optimizations.

3 Summarized results benchmarks

For the detailed results we refer to accompanying pdf and profiling call graphs.

3.1 Devstone

Faster due to directconnect. Optimistic suffers from transitions at same time points (frequency reverts higher), can gain with random ta.

3.2 Phold

Slightly faster than adevs. A reasonable amount of messaging can be done without slowing down simulator. Most of the runtime is spent in rng.

3.3 Interconnect

Slower than adevs due to high volume of messages. This creates a very cache unfriendly pattern, combined with cost of allocation on the heap for dxex. The completely cyclic dependencies of this model makes it very hard for our parallel implementations. Usage of pools reduces calls to new/malloc/sbrk, but can't eliminate it. Cache pattern is far harder to correct. Alignment of a Message on cache lines, and quadratic growing pools also help, but can't solve the problem.

4 Open issues

- Conservative lookahead caching (devstone -r), cpdevs is forced to ask useless lookahead values.
- Optimistic use pools in states. (from shared_ptr to ptr), see profile opdevs in repo.
- Optimistic hangs/too slow in cyclic (interconnect, phold)
- Directconnect : Yentl's faster algorithm. Not (yet) implement since it does not show in profiling graphs as a significant (<1%) cost.
- Conservative : Exploit full dependency graph to calculate next event directly (unsure of benefits).

5 Causes of slowdown (hardware)

A short illustration of key differences in hardware events for both simulators.

- Cache : Adevs : < 1% cache misses, dxex : <10% cache misses
- Context switches : Adevs : 4e-8 switches/cycles, dxex 3e-8 (conservative)
- Branch prediction : Adevs : 4%, dxex 0.08
- Cpu stalls : Adevs : 0.2 stalled cycle/instruction, dxex 0.8 stalled cycle/instruction.

The switches are (in single core) caused by syscalls (malloc, sbrk, free, time), in parallel by kernel space locking (mutex, thread, (depending on implementation atomics)).