Performance of libdispatch Based Framework Demo

Christopher Jones

Outline

Building Port

Measurements

Building Linux Port

Started from repository

Port from Snow Leopard (OS X 10.6)
https://www.heily.com/trac/libdispatch

Used new compiler

gcc 4.6.2

Compiler optimizations created race conditions in libdispatch

Lock free implementation broken by reordering of memory ‘store’ to be after an atomic barrier

order reversed

```
tail->do_next = NULL;
prev = dispatch_atomic_xchg(&dq->dq_items_tail, tail);
if (prev) {
     prev->do_next = head;
} else {
```

Thread 1	Thread 2
T0) put tail1 on end of list	
	T2) put tail2 on end of list and prev == tail1
	T3) tail2->do_next = NULL
	T4) tail1->do_next = head2
T5) tail1->do_next = NULL	

Building Linux Port(2)

- Start from RPM
  - Port from Lion (OS X 10.7)
  - http://mark.heily.com/sites/mark.heily.com/files/libdispatch-f16-SRPMS.tgz

- Had to use clang
  - Lion version of libdispatch makes use of Apple's extension to C 'blocks'
    - Use of blocks is not fundamental and could be removed

- No threading problems seen with this port

Test System

Physical Machine

- Intel(R) Xeon(R) CPU E5620
- 16 physical cores @ 2.40GHz
  - 4Cores/CPU with 4 CPUs
- 47 GB RAM

Virtual Machine

- 16 virtual cores
- 15 GB RAM
- SL6
  - libdispatch port needs a more modern kernel than SL5 provides

Measurement Strategy

Dependencies

Got module dependencies (what data each module uses) from CMS framework

Timing

Get per event module timing and read TBranch from file timing for Minimum Bias reconstruction

Feed dependencies and timing to demo framework

Approximate module timing by

Busy wait: calculate an integral calibrated for # iterations/sec

causes a demo module to take full core

Sleep: call usleep

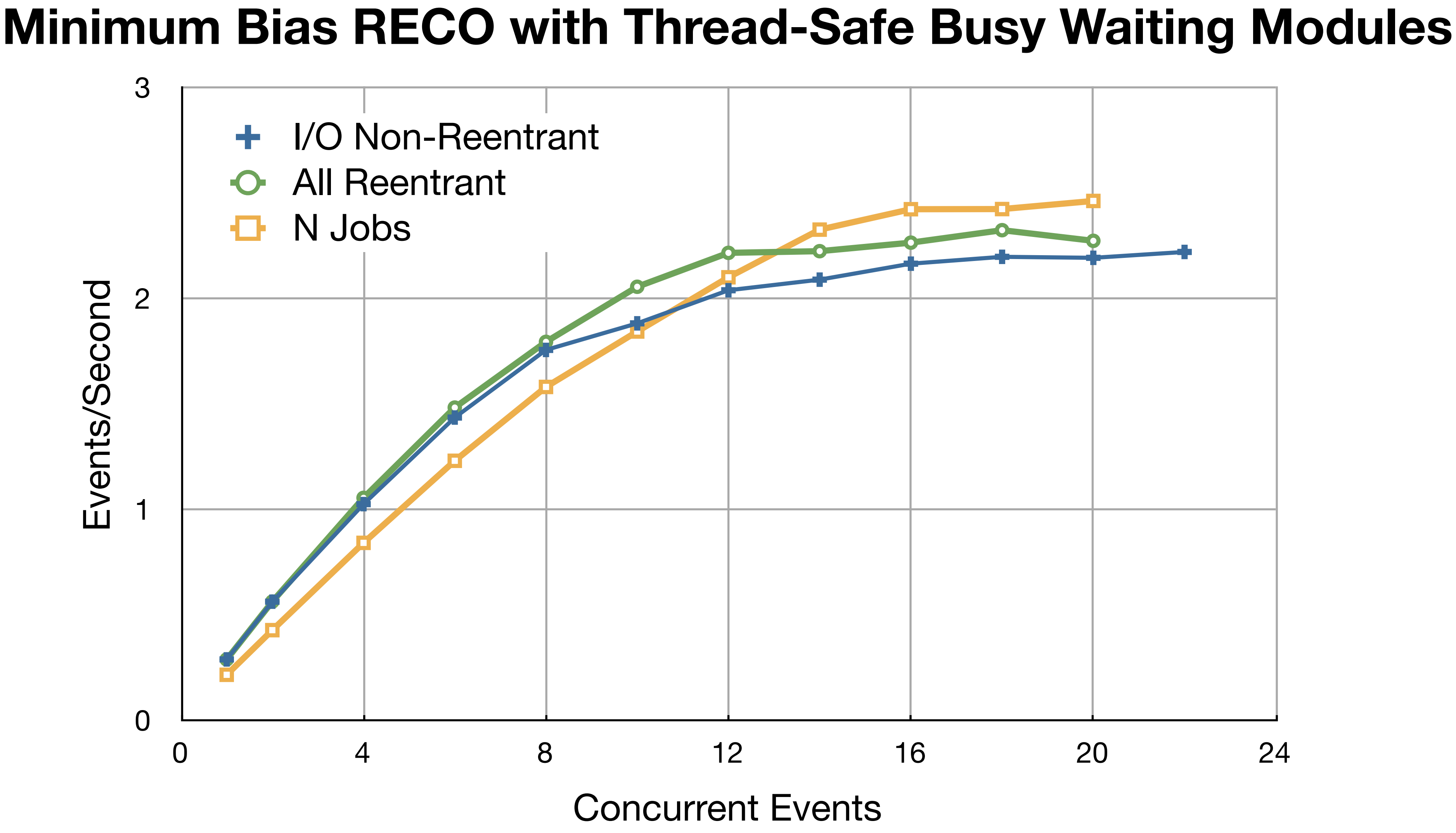
sleeping releases the core and allows another task to run
simulates having more cores available to the job

Threading tests

Producers and I/O are re-entrant

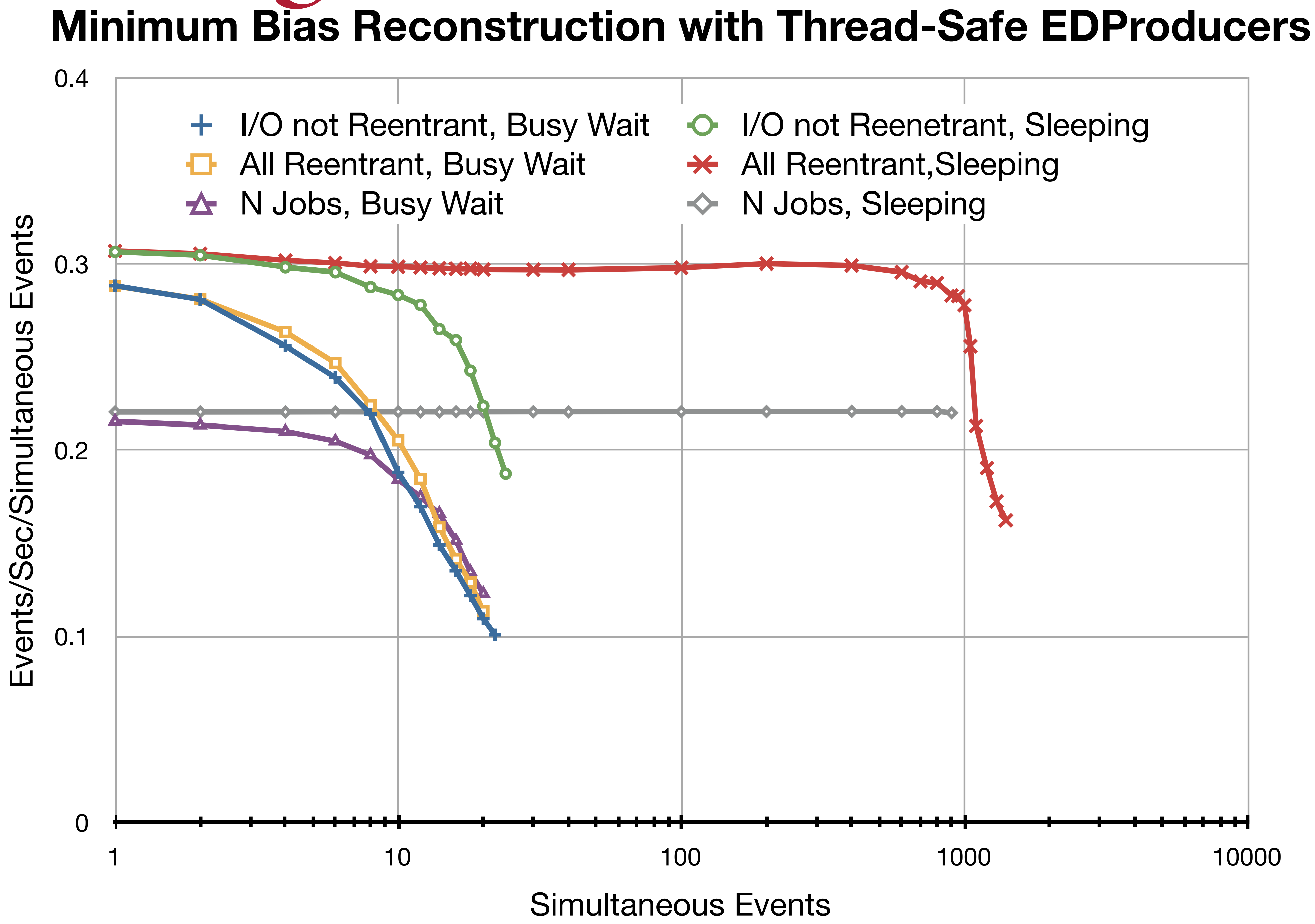
Producers are re-entrant but I/O can only processes one event at a time

Scaling: Busy Wait



Threaded versions flatten out sooner than N single threaded jobs since threaded jobs use up all 16 cores before reaching 16 concurrent events

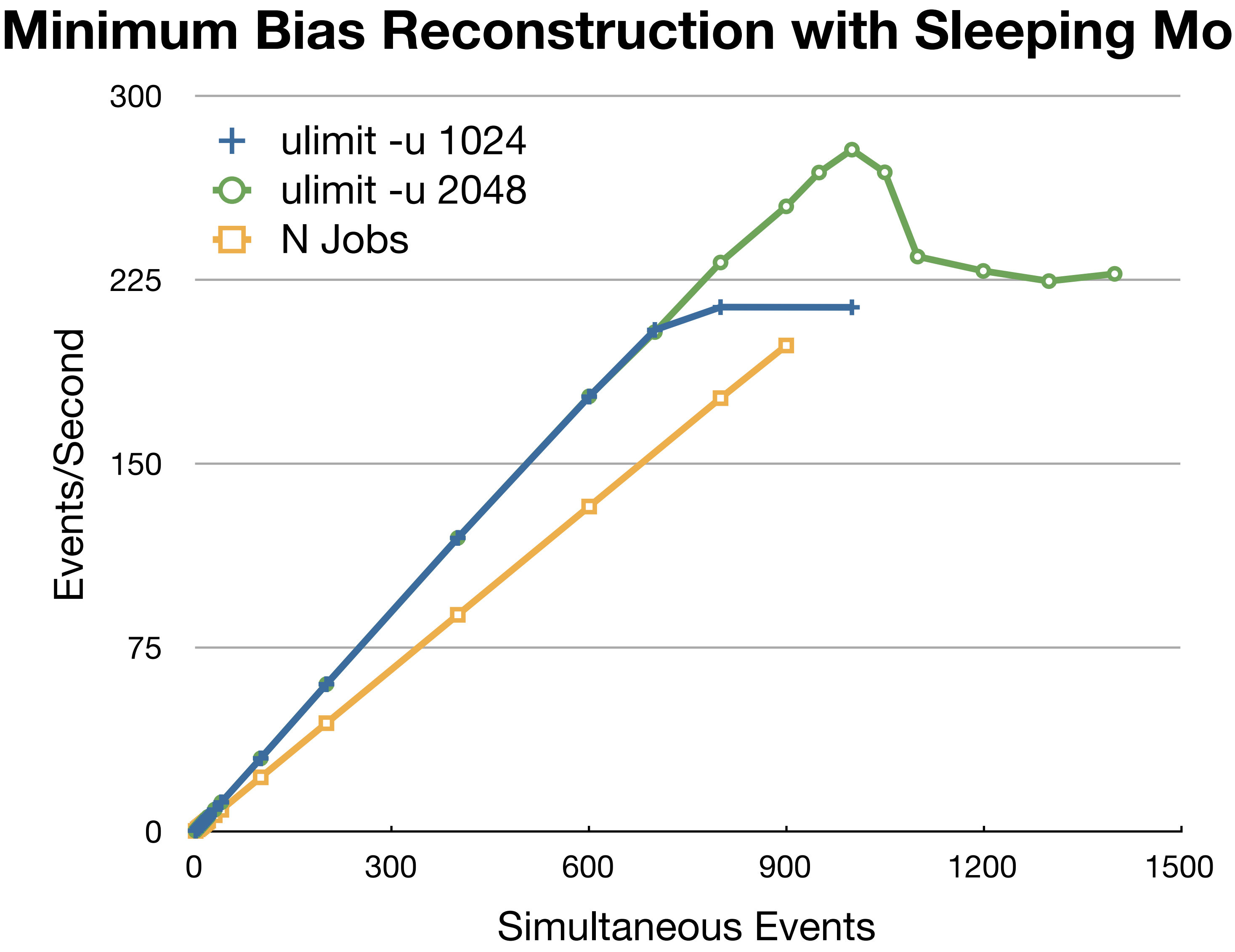
Scaling



Perfect Scaling is Flat in this Graph

N Job sleeping scaling failed since ran out of memory

Scaling Large Scale



First Hit Limit of 1024 Threads in System

Raised Limit and Have Hit an Unknown Limit

Not a memory limit since only using 680MB RSS (23GB VSize)

Number of running threads falls from 1600 to 1200 after peak

Conclusion

Promising Results for libdispatch

Scales Linearly up to 1000s of Concurrent Events

Accommodates thread-safe and non-thread-safe code

Easy to use internally to a module

Puzzling Failure of Scaling at Very Large Scale

Will try to find out the cause

Need to be aware that gcc 4.6.2 can cause problems with lock free implementations

C++11 standard's memory model will alleviate the problem

Additional Tools Would be Helpful

How many threads are active over time

Load on each CPU over time