Performance of OpenMP Based Framework Demo

Christopher Jones *FNAL*

Outline

OpenMP Overview

Measurements

Conclusion

OpenMP

Portable multiprocessing programming framework

Available for linux, Mac OS X, Windows, etc.

Built into compilers

gcc 4.6 has OpenMP 3.0

gcc 4.7 has OpenMP 3.1

Uses pragmas, libraries and environment variables

```
int ncount = 0;
#pragma omp parallel
#pragma omp shared(ncount)
{
   #pragma omp for
   for(i=0; i < 100; ++i) {
      int value = calculate(i);
      #pragma omp critical
      {
         ncount += value;
      }
   }
}
```

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

Exact same system as used for libdispatch tests

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

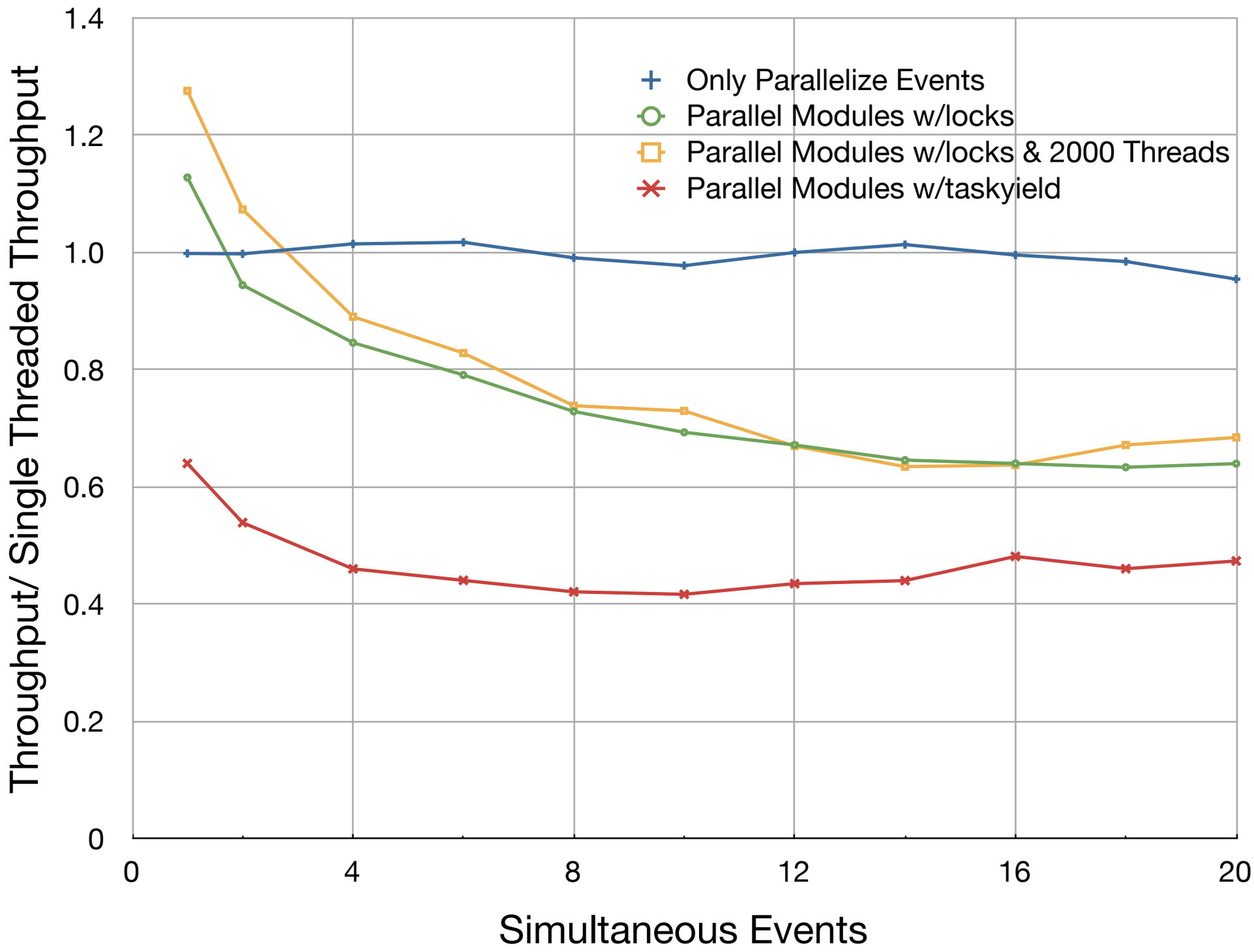
Threading tests

Producers and I/O are re-entrant

Subset of tests done for libdispatch

Throughput Scaling

OpenMP Throughput Relative to Single Threaded



Why Bad Scaling?

Want to only run a module once per event

Multiple simultaneous data requests means some tasks must wait

Only three ways to make a task wait

Implicitly at the end of a code block

- This is the traditional way to work with OpenMP

A lock

- This freezes the thread
- Having more threads than CPU helps some

taskyield construct

- Allows a task to give up its thread temporarily so other work can be done
- Spin wait on taskyield was too CPU intensive

libdispatch & TBB allow task notification

Can associate a new task to start when a group of tasks finish

Allows efficient waiting without using any thread resources

OpenMP does not (directly) support this behavior

Conclusion

OpenMP is not a good fit for parallel module system

Bad scaling since no 'task notification' system

Not an easy to use API

Compilers do not issue warnings/errors for pragmas

`#pragma omp taskyield` was not available in gcc 4.6 but it compiled fine

I had a hard time reasoning about when variables would be replicated across threads and what value they would have

```
void Module::prefetch(const Event& iEvent)
{
  for (auto& const g: m_getters) {
    Getter* temp = &g;
#pragma omp task untied default(shared), firstprivate(temp)
    iEvent.prefetch(temp);
  }
#pragma omp taskwait
}
```

CMS will not be using OpenMP for the framework