Parallelizing LArSoft modules II: MPI and OMP together

ERIC CHURCH, JUAN BRANDI-LOZANO, MALACHI SCHRAMM

RDNS and SDI groups, PNNL

20-Sep-2016

Outline

- Motivation
  - Want to speed up the code with minimal memory hit.
- Still focused on GausHitFinder_module.cc
  - Remind of OMP work from August talk
  - Now we tack on MPI
- Results
  - … such as they are

One might hope to gain performance improvements by threading up various LArSoft modules

- At PNNL I have a colleague (Juan) fluent with OpenMP and with MPI
  - We have lots of scientific computing resources at PNNL. In particular, one 24 core machine with 128+ GBytes memory we can play with. It’s largely all ours.
- OMP only requires small CMakeLists.txt changes and adding a couple pragmas in front of desired for loop
  - One big shared memory chunk that all the threads see
- We implemented OMP and have since moved to MPI
  - MPI distributes the memory across cores, and we throw some iterations of loops at those cores.
- In both cases, goal is to assemble the object at end of module after all threads are done, and, in one place, put_into() the event. Meaning, one serial task is still required to gather all output up.

Time spent in a typical reco chain 5_08

```
=================================================================================================================
TimeTracker printout (sec)                 Min        Avg         Max        Median        RMS        nEvts
=================================================================================================================
Full event                       26.4698      29.1435      31.1214      29.4146      1.47419      10
---------------------------------------------------------------------------------------------------------------
reco:rns:RandomNumberSaver                3.4461e-05    8.39357e-05   0.000455685   4.3729e-05    0.000124034     10
reco:digitfilter:NoiseFilter          13.428       13.5213      13.7195      13.4706      0.0937018      10
reco:caldata:CalWireROI                 3.92545      4.2721       4.55916       4.319      0.176564       10
reco:gaushit:GausHitFinder                1.44308       2.67415       3.65894       2.72471      0.738649       10
reco:TriggerResults:TriggerResultInserter    2.2549e-05    3.02089e-05   8.0534e-05    2.49175e-05   1.68072e-05     10
end_path:hitana:GausHitFinderAna          0.384017      0.472613      0.569486      0.489097      0.0613182      10
end_path:out1:RootOutput                7.25915      8.20184       8.92039      8.37981      0.524692       10
```

ROOT 6_02 out of the box

```
reco:gaushit:GausHitFinder                2.75883       7.17243       12.1659       7.28882       2.66237       10
```

These are times for 10 MicroBooNE real data events.

GSL 1 Thread vs 8 Threads – uB data

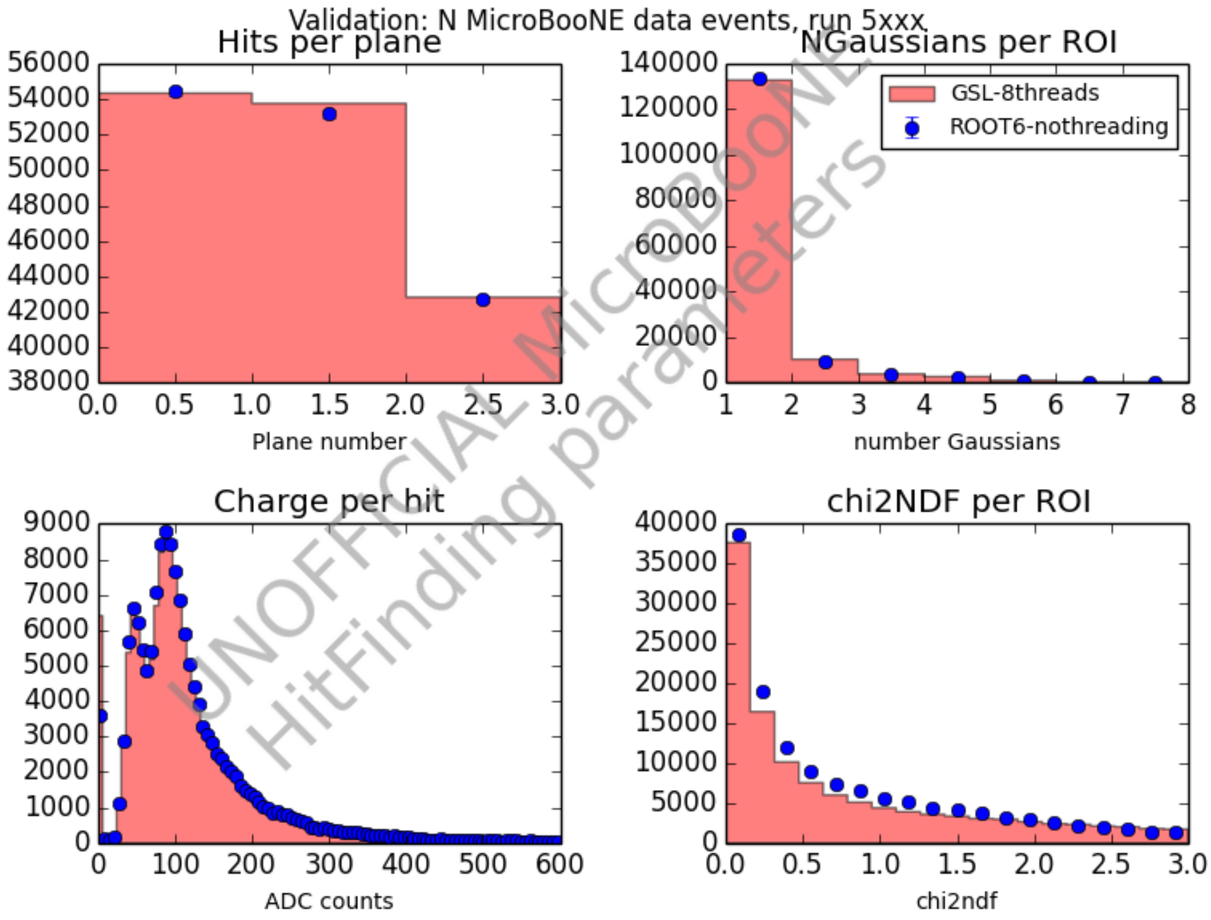
```
========================================================================================================
==========================================
TimeTracker printout (sec)                    Min          Avg          Max          Median          RMS          nEvts
========================================================================================================
==========================================
Full event                              27.0138      28.7438      30.038      29.088      1.01262      10
--------------------------------------------------------------------------------------------------------
reco:rns:RandomNumberSaver                    5.9985e-05    0.000112205   0.000446773   6.93335e-05   0.000112637     10
reco:digitfilter:NoiseFilter               14.6211      14.7859      14.9135      14.8057      0.105385       10
reco:caldata:CalWireROI                      3.80954      4.13443      4.43179      4.17129      0.17415       10
reco:gaushit:GausHitFinder                   0.621876      1.14477      1.55812      1.16916      0.335909       10
reco:TriggerResults:TriggerResultInserter    3.7077e-05    6.42121e-05   9.7182e-05    5.7808e-05    2.4508e-05      10
end_path:hitana:GausHitFinderAna              0.367328      0.473554      0.604935      0.499931      0.0754328      10
end_path:out1:RootOutput                      7.28578      8.2021       8.90999      8.3884      0.523765       10
========================================================================================================
==========================================
```

- GSL 8 Threads

```
reco:gaushit:GausHitFinder                   0.205231      0.463043      0.780243      0.510354      0.166003       10
```

x5 faster just getting rid of ROOT6 in favor of GSL fitting. another x2.5 faster going to 8 threads.

Validation plots



Nothing has been broken going to GSL! ….

What modules should be next?

- All above pushed to larreco feature/echurch_
- RawDigitFilter in MicroBooNE’s case, anyway, is another big offender.
  - Removes various noise sources
  - This is changing bigly for MCC8, so let’s not assume it remains an offender
- But, it is another module that runs over (groups of) wires, and should probably be easy to ||’ize.
  - I should never say easy.
  - There could be memory problems holding onto groups of wires.
  - But the whole ROOT fitting rathole is absent in this module.
- In general, a lot of low hanging fruit for OMP threading for little cost.

How does this all fit into HPC at FNAL?

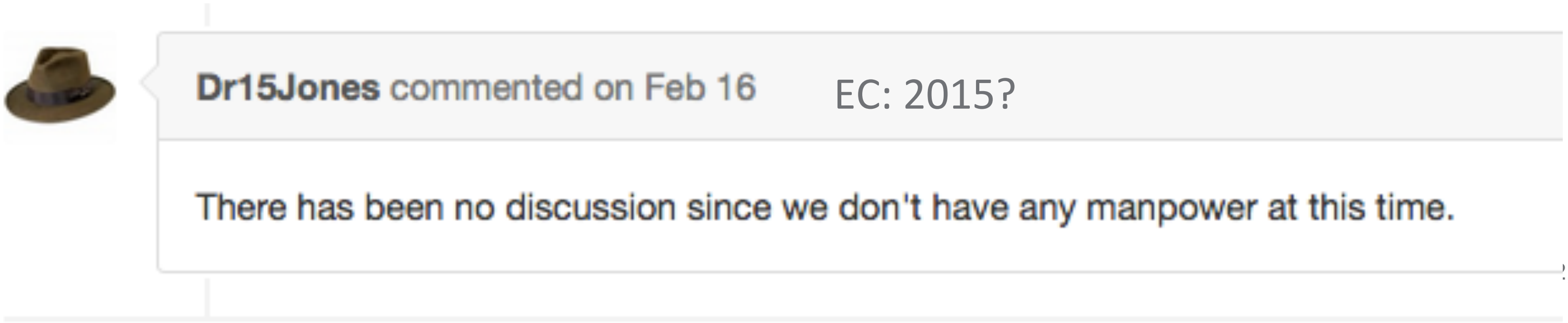
- I don't know.

- condor knows how to allocate jobs for multi-threaded code, I think.
  - Does jobsub know how to wrap that up? Certainly, not all modules' needs can be balanced and jobs put on appropriate worker nodes.

- => if all cores on a node are already spinning, there's nothing to be gained from any of this threading.

MPI – message passing interface

- I put out a message on artists listserv to ask if any consideration has been given by *art* to launch events within a job with MPI.
  - I received the answer that artdaq does in fact use MPI.
  - Which is true: basically to set up the running BoardReader and EventBuilder processes and pass fragments from one to the other.
  - Kind of a non-answer wrt *art* and LArSoft
- If I search on cmssw I find that there's an open issue suggesting that N events be spawned with MPI and finish and more started …
  - This seems like a great idea, but alas it ends with a whimper …
  - https://github.com/cms-sw/cmssw/issues/12922



Dr15Jones commented on Feb 16

EC: 2015?

There has been no discussion since we don't have any manpower at this time.

MPI 2

- This would mean jobs do not get “stuck” on a slow event, as other cores are still at work on other events on nodes across an infiniband-connected network.
  - This is probably the right way to use MPI
- Absent using MPI like that, can we launch jobs that, within a given module, are allowed to fork processes out to other nodes?
  - Umm, no, it turns out.
  - The central issue: in the deeply buried *art* state machine we can’t just grab onto the main() and launch MPI processes around the one module we care about.
  - We have to launch MPI once and have *all* the modules in the job run on N processors.
    - This is stupid: modules which you don’t care about just run N redundant copies. produce() modules stomp on each other’s output.
    - TimeService and MemoryService, e.g., choke and die cuz they’re trying to write to the same sql .db file simultaneously.

MPI 3

- Nevertheless, we did precisely the aforementioned thing.
  - Shut off TimeService and MemoryService

- In GausHitFinder only do we pass messages.
  - Meaning, only here do we deliberately code to MPI.

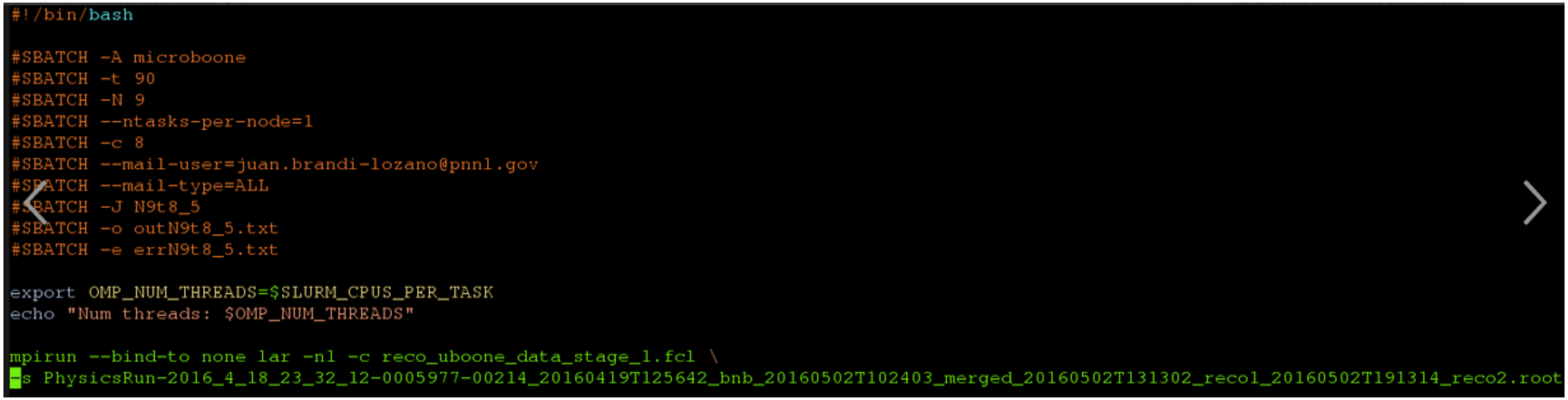
- We do the following
  - We ask particular ranges of the Wire iterations to go run on other nodes.
  - Proc 0 is the master; it sits and waits for the others to pass back their data
  - There are lots of gymnastics required to pack up the data on each end and ship it and receive it as raw bytes.
    - art::Ptrs, etc, may not be passed as messages
    - std::maps may not be passed (nor std::anything)
    - We have to loop over wires to receive the data from each proc
    - The hitCol is assembled, and finally put_onto() the event as in the OMP case

MPI 4

- Next stupid thing that's necessary is in order to not have our N instances of GausHitFinders all try to put_onto() their data and thus stomp all over each other at the output stage, we kill procs 1,2,3, …,Nproc after everyone reports to proc0, and we only allow proc 0 to proceed.
  - Maybe instead there's a way to suppress the put_onto() in the non-0 procs but we didn't pursue that.
- We dump the expected hits out to the art-root file.
- Subsequent modules finish out fine,
- **We can run precisely one event in this manner.**

- It's a non-optimal, proof-of-principle.
  - Not sure entirely which principle.
  - MPI could be used, I guess, is the statement.

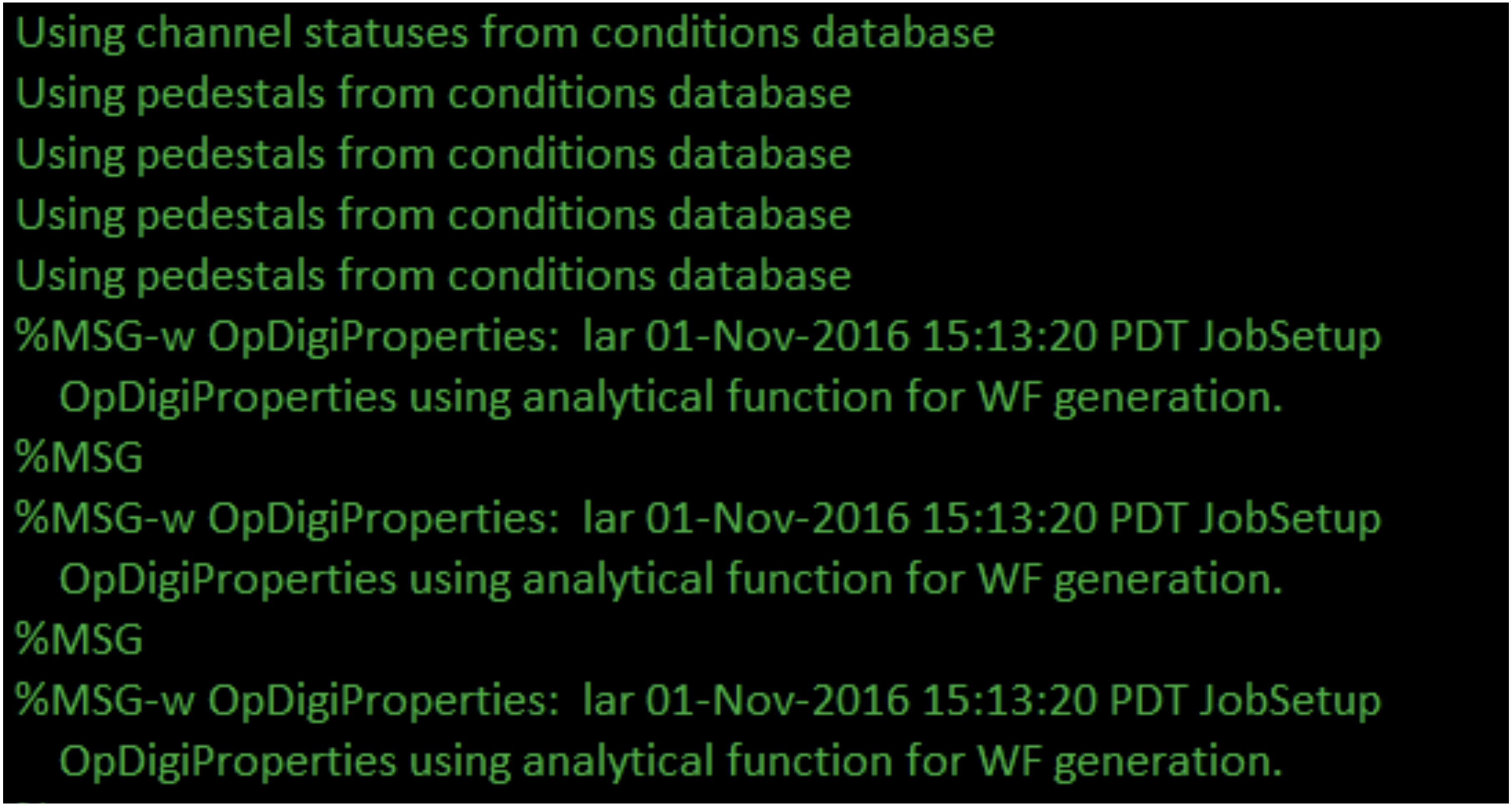
Launch the job



```
#!/bin/bash

#SBATCH -A microboone
#SBATCH -t 90
#SBATCH -N 9
#SBATCH --ntasks-per-node=1
#SBATCH -c 8
#SBATCH --mail-user=juan.brandi-lozano@pnnl.gov
#SBATCH --mail-type=ALL
#SBATCH -J N9t8_5
#SBATCH -o outN9t8_5.txt
#SBATCH -e errN9t8_5.txt

export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
echo "Num threads: $OMP_NUM_THREADS"

mpirun --bind-to none lar -n1 -c reco_uboone_data_stage_1.fcl \
-s PhysicsRun-2016_4_18_23_32_12-0005977-00214_20160419T125642_bnb_20160502T102403_merged_20160502T131302_reco1_20160502T191314_reco2.root
```

Output from redundant module running

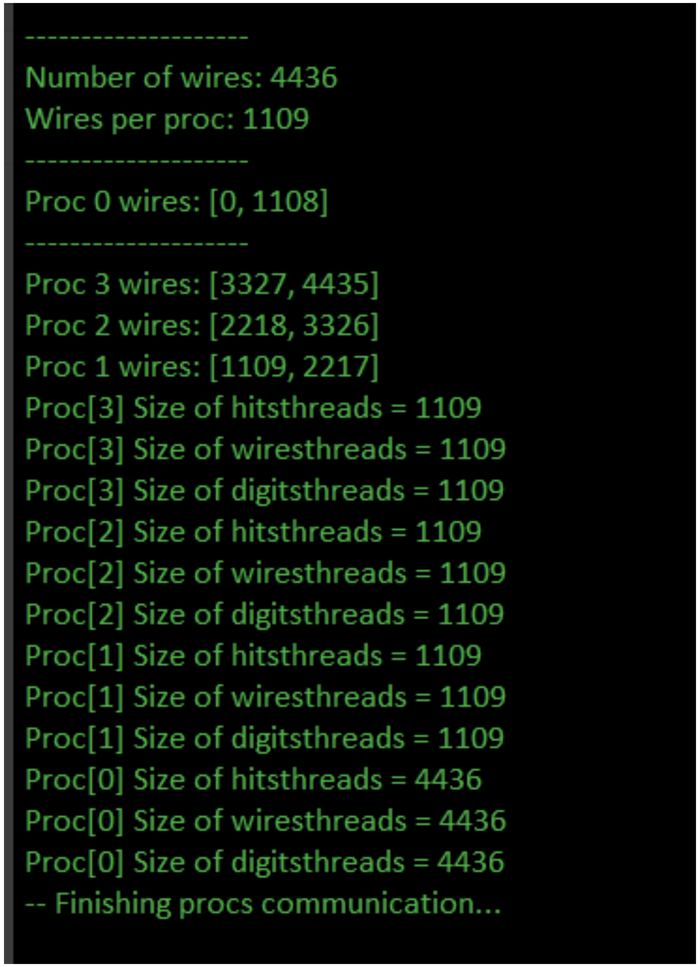
Set N=4. Services and everything all redundantly reporting 4 times



```
Using channel statuses from conditions database
Using pedestals from conditions database
Using pedestals from conditions database
Using pedestals from conditions database
Using pedestals from conditions database
%MSG-w OpDigiProperties:  lar 01-Nov-2016 15:13:20 PDT JobSetup
   OpDigiProperties using analytical function for WF generation.
%MSG
%MSG-w OpDigiProperties:  lar 01-Nov-2016 15:13:20 PDT JobSetup
   OpDigiProperties using analytical function for WF generation.
%MSG
%MSG-w OpDigiProperties:  lar 01-Nov-2016 15:13:20 PDT JobSetup
   OpDigiProperties using analytical function for WF generation.
```

In GausHitFinder only each proc does unique work

Each Proc 1-3 works on 1109 wires



```
--------------------
Number of wires: 4436
Wires per proc: 1109
--------------------
Proc 0 wires: [0, 1108]
--------------------
Proc 3 wires: [3327, 4435]
Proc 2 wires: [2218, 3326]
Proc 1 wires: [1109, 2217]
Proc[3] Size of hitsthreads = 1109
Proc[3] Size of wiresthreads = 1109
Proc[3] Size of digitsthreads = 1109
Proc[2] Size of hitsthreads = 1109
Proc[2] Size of wiresthreads = 1109
Proc[2] Size of digitsthreads = 1109
Proc[1] Size of hitsthreads = 1109
Proc[1] Size of wiresthreads = 1109
Proc[1] Size of digitsthreads = 1109
Proc[0] Size of hitsthreads = 4436
Proc[0] Size of wiresthreads = 4436
Proc[0] Size of digitsthreads = 4436
-- Finishing procs communication...
```

MPI*OMP results

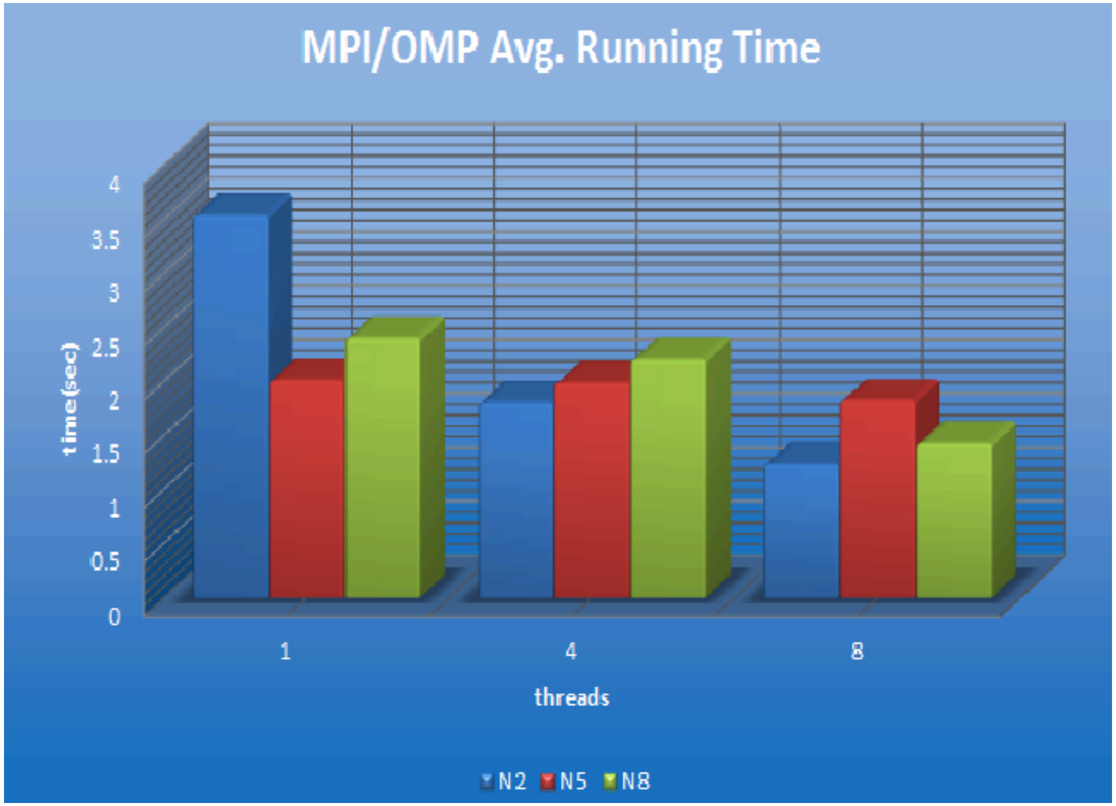
- I remind that within each of the N MPI process we still spawn our M OMP threads

- There's some overhead to the N-1 message passing, and it is expected that performance gains will only be observed if the work performed in the Wire iterations is substantial compared to that time.

MPI*OMP wall time in GausHitFinder

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by **Battelle** *Since 1965*

Unhappily, the compute time required is fast enough that we don't clearly see the desired scaling

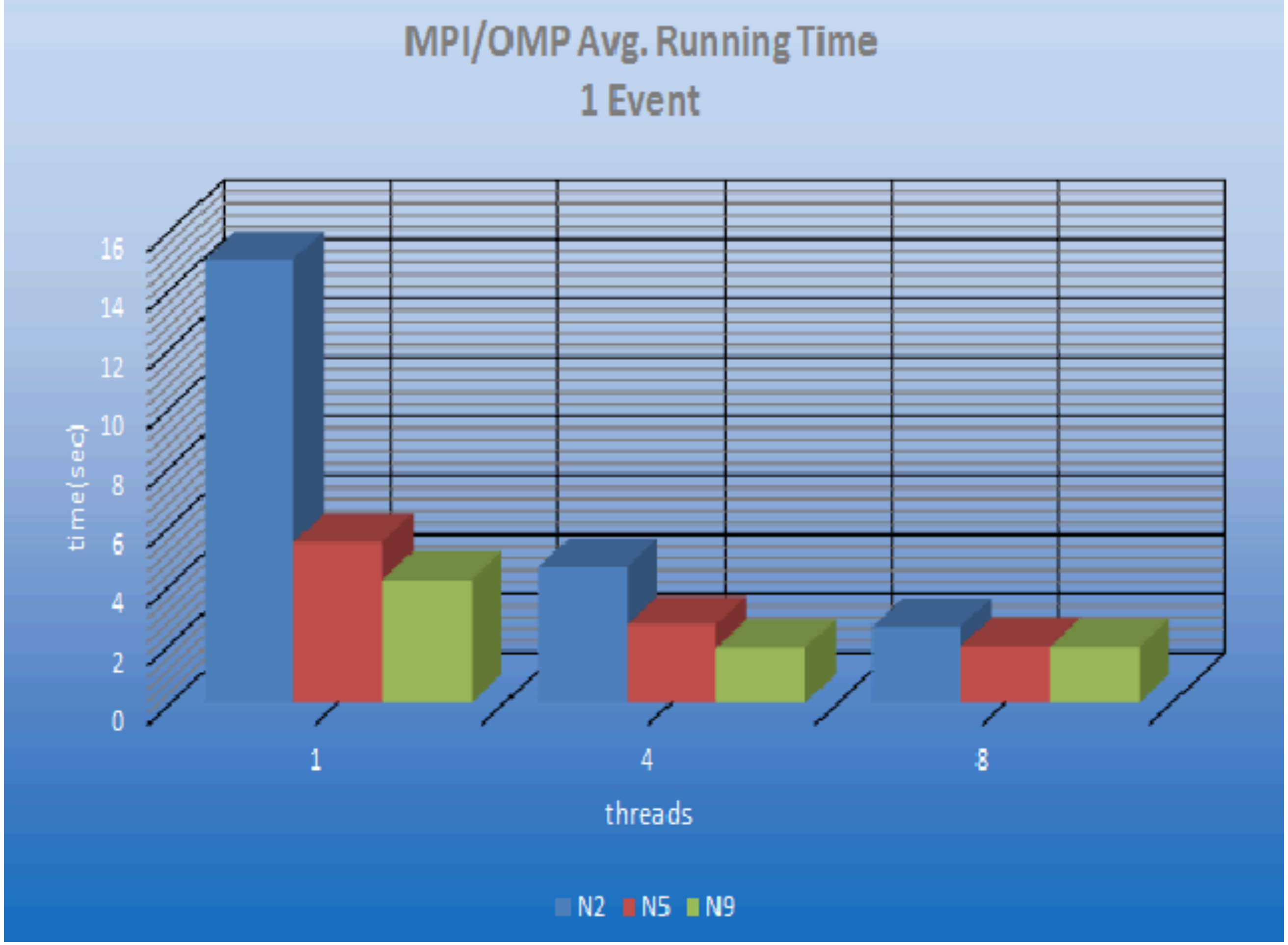


MPI*OMP wall time in GausHitFinder with usleep (1000) inside the Wire loop

Pacific Northwest NATIONAL LABORATORY

Proudly Operated by **Battelle** *Since 1965*

Bloating up the compute time per thread, we see scaling (perhaps not linear).



Summary, next work

- ~~Would like to do some MPI implementation~~
  - ~~Would perhaps be very useful to show how this works across multiple nodes.~~
- ***art* should consider work to allow MPI spawned events**
  - **or some means by which to fork mpi jobs module-by-module**
- We'd like to partner with FNAL to do some of this work if it's deemed valuable.