Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Parallelizing LArSoft modules

ERIC CHURCH, JUAN BRANDI-LOZANO, MALACHI SCHRAMM

RDNS and SDI groups, PNNL

20-Sep-2016

U.S. DEPARTMENT OF ENERGY

Outline

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

- Motivation
  - Want to speed up the code with minimal memory hit.
  - PNNL resources (computers, FTEs)
- Focused on GausHitFinder_module.cc
  - Problems we knew we had to overcome
  - Problems that caused a bit of trouble, but which we overcame
  - Stubborn shiz that was irritating that needed solving and was beyond the scope of what we thought we were taking on! … which we overcame
- Results
  - Resource changes (both better and worse)
    - Mainly better
- What should be next?

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

Pacific Northwest NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- At PNNL we have some people 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 thought we'd quickly implement OMP and then move to MPI
  - MPI first on one machine, then on N machines (HPC scalability)
    - We're only at MPI now! "Simple" OMP'ifying took longer than expected.
  - MPI distributes the memory across cores, and we would throw particular 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.

Candidate modules

- Any which have a big for loop on the 8,000+ channels in a big LArTPC, for example. 150k+ in DUNE?
  - Want a module, in which iterations don't depend on each other, and could in principle be performed concurrently.
  - Erica suggested GaussHitFinder_module in a MicroBooNE collaboration meeting talk a couple months ago, and Ruth mentioned this possibility at ICHEP too.
  - Want a module which takes as much time as many other single module, depending on what reco chain is being run, of course.
  - At PNNL, we had chosen GausHitFinder_module independently, for the same reasons others had identified it as one worth attacking.

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

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

One might rather tackle NoiseFilter or CalWireROI, but that's for this particular reco chain.

These are times for 10 MicroBooNE real data events.

GausHitFinder_module.cc

- Takes CalROI waveforms on all wires as input
  - These are deconvolved waveforms that have been identified as possibly containing a hit
  - Loops over these, makes a histogram of each, and then fits n Gaussians in a THF1 with a TF1 function – set as Gaussian with nx3 parameters. Deletes that histogram and fit at each iteration.
  - Right after that, in the middle of the big while(wires) loop, it calls HitCreator, keeping track of all associations, move()'s the pointer to a Hit() and pushes that back onto a HitCollection hcol, and at the end put_into()'s the Event.

Henceforth, till explicitly stated,

- Single muon MC sample of 20 events in MicroBooNE,
  - Sometimes I will show uB data.
- v05_08_00 (ROOT 5.34)
  - Sometimes I will show v06_02_00 (ROOT6) results
- Will use OpenMP
- Have not yet pushed our git-forked PNNL repo branches back to FNAL redmine.

Obvious thing to do

- Simply put an OMP pragma in front of the while loop over wires.

```
#pragma omp parallel
    {
    //iterating over all wires
    #pragma omp for schedule(dynamic)
    for(size_t wireIter = 0; wireIter < wireVecHandle->size();
           wireIter++)
    {
…
```

- Export OMP_NUM_THREADS=1,2,4,8,24 ….
  - This little change causes wireIter iterations to go to that many processor threads. As they finish the next one is assigned out ….

- Boom. Mic-drop.

First two problems.

- First problem. hcol is global. => thread contention.
  - Okay, instead of constructing each hcol iteration, hang onto the arguments – build them in std::maps – and sequentially build the hcol after all threads are joined, using each item in the map.
- One other problem: HitCreator(), which remains inside the thread function, takes as an argument the art::Services geom singleton (global!). It needs thesSignalType – Induction/Collection.
  - This causes thread contention again.
  - Easily solved. Get the signalType, and pass it instead of full art::geom object.
    - Requires new lardata HitCreator constructor.

Assembling the arguments – 3 slides we can skip if people don't care


```
std::map < uint16_t, std::vector<recob::Hit>> hitsthreads;
std::map < uint16_t, std:: vector <art::Ptr<recob::Wire>> > wiresthreads;
std::map < uint16_t, std:: vector <art::Ptr<raw::RawDigit>> > digitsthreads;
```

```
                  ....
                  numHits++;
                  onehitperthread.emplace_back(hitcreator.move()
                  onewireperthread.emplace_back(wire);
                  onedigitperthread.emplace_back(rawdigits);
              } // <---End loop over gaussians
        }//<---End loop over merged candidate hits
    } //<---End looping over ROI's
```

Using the arguments

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

```
            hitsthreads[tid]   = onehitperthread;
            wiresthreads[tid]  = onewireperthread;
            digitsthreads[tid] = onedigitperthread;

          }//<---End looping over all the wires
       } // end of omp parallel region
```

……….

```
for(auto const& jj : t_ID ){
      //    std::vector <int> t_IDtmp(0);
      //std::iota (std::begin(t_IDtmp), std::end(t_IDtmp), 0);
      //for(auto const& jj : t_IDtmp ){

      auto  hind = hitsthreads.find(jj);
      auto  wind = wiresthreads.find(jj);
      auto  dind = digitsthreads.find(jj);
      if (hind == hitsthreads.end()) break;
      if (wind == wiresthreads.end()) break;
      if (dind == digitsthreads.end()) break;
      for (uint16_t kk=0 ; kk < (hind->second).size(); kk++ ) {

        hcol.emplace_back(hind->second.at(kk),wind->second.at(kk),dind->second.at(kk)
        }
    }
```

std::map arguments, the real lesson

- The point of all this is that std::maps are not thread safe, as-is.
- However, one can access them across threads if all their keys have been initialized. Who knew? (This crowd, probably.) I did not.

```
for(int ii=0; ii<(int)wireVecHandle->size(); ii++){
  hitsthreads[ii] = std::vector<recob::Hit>();
  wiresthreads[ii] = std::vector<art::Ptr<recob::Wire>>();
  digitsthreads[ii] = std::vector<art::Ptr<raw::RawDigit>>()

}
```

Lesson

- Do not be so hasty to drop the mic…

ROOT 5.34's TF1 fit to TH1F approach – this and next 4 slides are worth presenting if people want the narrative (in which there is in fact educational value)...

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

- TF1 fit to TH1F segfaults if $OMP_NUM_THREADS>1
- Googling for it shows it to be a well-known problem.
  - Some big global ROOT something is instantiated that multiple threads can't contend for concurrently
- And here, the long, winding trail in the ROOT forest begins.

ROOT::Fit::Fitter

- The Wisdom (Extensive googling) suggests to use ROOT::Fit::Fitter instead of THF1/+THF1 and to wrap each instance in its own Tthread().
- We followed this path.
  - Example code from Andreas Zoglauer as in ROOT tutorial implemented.
  - One needs to define a struct{} to pass to the thread function in order to set, and get back things like start, end points, fit parameters, their errors, chi2ndf. Each ROI instance of the class needs to hold/give back the relevant values.
- This sort of worked. Still, however, some contention somewhere with n threads => gdb dumps of segfaults show crashes at such places
  - And, worse, the memory bloats to 15 GB/evt on the 1st event, and was not given back.
  - This, despite being sure to properly destruct any stray anything we could find.

Let’s try ROOT6

- We jump to branch LARSOFT_SUITE_v06_02_00_ec
  - Online there is some suggestion multi-threading is better in ROOT6
  - And before re-implementing TThread approach, we find some internet chatter that ROOT6 has perhaps even solved the TF1 fit to TH1F approach
    - We rewind to that would-be solution.
    - **Still segfaults** with OMP_NUM_THREADS>1. So, just no.

  - Thus, Re-implement ROOT::Fit::Fitter TThread() approach.
    - Now, with 8 threads the memory hovers at a mere ~2.5 Gbytes after a few events!
    - However, performance is sporadic.
      - sometimes after all threads are evidently complete and joined, the ROOT Ouput takes a long time and event sometimes crashes.
      - Sometimes there is a hang, which I can get past with *gdb –p procID; continue* in another window. Seems, erm, inelegant and unsustainable
      - Changing thread num up or down can make memory leak re-appear.
      - Sometimes –nskip 10 makes everything work fine.
      - => something remains F’d in ||’izing root even in ROOT6
      - Could be user mis-implementation, for sure, still

ROOT5 jobs’ calgrind output

- TCloning seems to be aggressively copying everything sometimes.

```
    GB
15.08^                                                                   :
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                              @@:#::@:
     |                                                             @@@:#::@:
     |                                                             @@@:#::@:
     |                                                             @@@:#::@:
     |                                                             @@@:#::@:
     |                                                             @@@:#::@:
     |                                                             @@@:#::@:
     |                                                             @@@:#::@:
   0 +----------------------------------------------------------------------->Gi
     0                                                                   519.5
```

calgrind stack trace from previous event

```
 71 544,704,049,969   16,060,353,864   16,036,905,004    23,448,860          0
99.85% (16,036,905,004B) (heap allocation functions) malloc/new/new[], --alloc-fns, etc.
->93.55% (15,023,996,928B) 0x7ED87C5: char (&TTHREAD_TLS_INIT_ARRAY<4, char [2048], char>()) [2048] (ThreadLocalStorage.h:242)
| ->93.55% (15,023,996,928B) 0x7ED81C5: TRegexp::MakeWildcard(char const*) (TRegexp.cxx:152)
|   ->93.55% (15,023,996,928B) 0x7ED7EFA: TRegexp::TRegexp(char const*, bool) (TRegexp.cxx:55)
|     ->93.55% (15,023,996,928B) 0x7EC0A25: TPluginHandler::CanHandle(char const*, char const*) (TPluginManager.cxx:174)
|       ->93.55% (15,023,996,928B) 0x7EC350F: TPluginManager::FindHandler(char const*, char const*) (TPluginManager.cxx:612)
|         ->93.52% (15,019,802,624B) 0x6F3E6E9: ROOT::Math::Factory::CreateMinimizer(std::string const&, std::string const&) (Factory.cxx:91)
|         | ->93.52% (15,019,802,624B) 0x6F3FE20: ROOT::Fit::FitConfig::CreateMinimizer() (FitConfig.cxx:211)
|         |   ->93.52% (15,019,802,624B) 0x6F52514: ROOT::Fit::Fitter::DoInitMinimizer() (Fitter.cxx:652)
|         |     ->93.52% (15,019,802,624B) 0x6F52B9B: ROOT::Fit::Fitter::DoMinimization(ROOT::Math::IBaseFunctionMultiDim const&,
ROOT::Math::IBaseFunctionMultiDim const*) (Fitter.cxx:722)
|         |       ->93.52% (15,019,802,624B) 0x6F501D9: ROOT::Fit::Fitter::DoLeastSquareFit(ROOT::Fit::BinData const&) (Fitter.cxx:336)
|         |         ->93.52% (15,019,802,624B) 0x5F69F9B: ROOT::Fit::Fitter::Fit(ROOT::Fit::BinData const&) (Fitter.h:139)
|         |           ->93.52% (15,019,802,624B) 0x25D9ED42: hit::GausHitFinder::FitGaussians(std::vector<float, std::allocator<float> > const&,
std::vector<std::pair<int, int>, std::allocator<std::pair<int, int> > > const&, int, int, double, std::vector<std::pair<double, double>,
std::allocator<std::pair<double, double> > >&, double&, int&) (GausHitFinder_module.cc:1086)
|         |             ->93.52% (15,019,802,624B) 0x25D9BD1E: hit::FitGaussian_Tt(void*) (GausHitFinder_module.cc:286)
|         |               ->93.52% (15,019,802,624B) 0x7A06154: TThread::Function(void*) (TThread.cxx:801)
|         |                 ->93.52% (15,019,802,624B) 0x3D2D207A9F: start_thread (in /lib64/libpthread-2.12.so)
|         |                   ->93.52% (15,019,802,624B) 0x3D2CAE893B: clone (in /lib64/libc-2.12.so)
|         |
|         ->00.03% (4,194,304B) in 1+ places, all below ms_print's threshold (01.00%)
```

At this point, let's investigate other Fitters

- First thought was to use R
  - LARSOFT_SUITE_v06_02_00_Rcpp
  - Minimal (6-FTE-hours spent figuring out the needed CMakeLists.txt change) for Rcpp to compile/build LArSoft module on our pnnl machine.
  - R code exists that can minimize a chi2 on a non-linear function and return fit parameters.
  - However, nobody uses R this way. People instead use C++ in R so that they can make their calls in interactive R faster.
  - What we want can be done, however: one seems to need to build huge long strings that are the interactive R calls and then pass them through a parser in our C++ code.
    - Would be inelegant to assemble these strings thread-by-thread
    - Not obviously thread safe either. Didn't get that far to know .
    - We abandoned this, because of …

GNU Scientific Library (GSL)

- ROOT's MINUIT fitter sits on top of GSL …
- All we'd be doing is eliminating all the ROOT fitting wrapping, at the expense of needing to code up two free functions, and make it all work:
  - One function is an N-gaussian residual-from-data calculator
  - Another function calculates Its first derivative
- GSL is manifestly thread safe. There are no unknown globals or singletons being created anywhere. No crazy TCloning going on.
- It comes with any of the LArSoft bundles from scisoft.fnal.gov! (I didn't realize that at first.)
  - CMakeLists.txt change took a mere 1 FTE hour.
- _LARSOFT_SUITE_v06_02_00_gsl

GSL does what we'd hoped all along

- 1 and 8 thread peak VMem goes from ~2 to 2.3 GB for single particle MC muons.
  - Unsurprising linear memory increase.
  - No memory leaks: memory recovers after each event
- CPU time is reduced by about x6 on this sample.
  - Unsurprising asymptotic performance

calgrind plot for GSL running

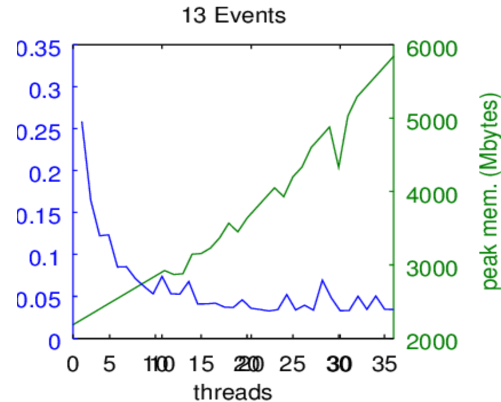
Single particle MC on 4 events, but data looks similarly x2

This is *resident* memory coming from calgrind, I think.

GB
1.241^
0
>Gi
0
355.7

CPU/VMem –single particle MC

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



8 threads gives x5 speed increase in single muon MC at ~15% increase in virtpeak memory

CPU/VMem – MicroBooNE data

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

- We try this now on MicroBooNE data where we have a lot of crossing muons, and therefore, many more hits than single particle MC
- => Repeatable, performant

ROOT 6_02 (unthreaded, out of box)

- ==
- TimeTracker printout (sec) Min Avg Max Median RMS nEvts
- ==
- Full event 28.9618 34.8124 40.2324 35.2362 3.35886 10
- ---
- reco:rns:RandomNumberSaver 5.5597e-05 0.000114742 0.000408752 8.8504e-05 9.94096e-05 10
- reco:digitfilter:NoiseFilter 14.5245 14.5967 14.7099 14.5796 0.0575846 10
- reco:caldata:CalWireROI 3.78673 4.13537 4.44043 4.17267 0.180542 10
- **reco:gaushit:GausHitFinder** 2.75883 **7.17243** 12.1659 7.28882 2.66237 10
- reco:TriggerResults:TriggerResultInserter 3.8007e-05 4.67128e-05 7.2272e-05 4.6923e-05 9.54044e-06 10
- end_path:hitana:GausHitFinderAna 0.436671 0.616667 0.768997 0.649385 0.114106 10
- end_path:out1:RootOutput 7.32778 8.28955 8.96363 8.4851 0.540422 10

ROOT6 x2 slower than ROOT5.34 (2.67 from slide 5, remember.)

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

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

x2 faster just getting rid of ROOT6. another x2.5 faster going to 8 threads.

Pacific Northwest
NATIONAL LABORATORY
Proudly Operated by Battelle Since 1965

Validation plots

Validation: N MicroBooNE data events, run 5xxx

Hits per plane

Plane number

NGaussians per ROI

number Gaussians

Charge per hit

ADC counts

chi2NDF per ROI

chi2ndf

Nothing has been broken going to GSL! …. Chi2ndf is even better?

What modules should be next?

- RawDigitFilter in MicroBooNE's case, anyway, is another big offender.
  - Removes various noise sources

- This 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.

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.

- We think a 10-20% memory increase at the expense of x3-5 cpu time decrease is good.
  - Probably not all modules will benefit so dramatically
  - If all cores on all nodes are already busy, none of this buys anything.
    - But, it does by something, if cores on nodes are at all idle.
  - Also, could this 10-20% increase trip up the max memory enforcing mechanisms?

Not obvious where PNNL goes with this

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

- Would like to do some MPI implementation
  - Would perhaps be very useful to show how this works across multiple nodes.
- We might investigate getting a paper out of this
  - What does LArSoft say about that?
- We'd like to partner with FNAL to do some of this work if it's deemed valuable.
  - Our money to explore much further will be out very soon