Fermilab

Managed by Fermi Research Alliance, LLC for the U.S. Department of Energy Office of Science

LArSoft vectorization tests

Guilherme Lima

LArSoft Coordination Meeting

March 13, 2018

Vectorization and LArSoft

- At the Dec.05 LArSoft coord.meeting, I showed plans to find good candidates for vectorization, which could be used for simple tests
- Suggested to look at signal processing and hit finding
  - profiling results from Soon Y. Jun
    - look at *detsim* numbers on the proton_6GeV job, listed by functions (FUNS):
      - 9.9% IdealAdcSimulator::count(...)
      - 9.8% Legacy35tZeroSuppressService::filter(...)
      - 6.7% ExponentialChannelNoiseService::addNoise(...)
      (...)

dupetpc_06_57_00 LArSoft/protoDune (proton 6GeV detsim)

CPUTIME(Inclusive)	CPUTIME(Exclusive)	Function
1.21e+09 100 %	1.21e+09 100 %	Experiment Aggregate Metrics
1.61e+08 13.4%	1.61e+08 13.4%	__GI_memcpy
1.20e+08 9.9%	1.20e+08 9.9%	IdealAdcSimulator::count(double, unsigned int, unsigned int) const
1.38e+08 11.5%	1.18e+08 9.8%	Legacy35tZeroSuppressService::filter(std::vector > const&, unsigned int, float, std::vector
8.30e+07 6.9%	8.12e+07 6.7%	ExponentialChannelNoiseService::addNoise(unsigned int, std::vector >&) const
7.24e+08 60.0%	6.94e+07 5.8%	SimWireDUNE::produce(art::Event&)
4.91e+07 4.1%	4.91e+07 4.1%	AdcCodeHelper::subtract(short, float) const

    - All good candidates, but they are DUNE code, not LArSoft

Similar reports for other jobs

dupetpc_06_57_00 LArSoft/protoDune (beam cosmics 1GeV detsim)

CPUTIME(Inclusive)	CPUTIME(Exclusive)	Function
1.78e+09 100 %	1.78e+09 100 %	Experiment Aggregate Metrics
1.28e+08 7.2%	1.28e+08 7.2%	__gnu_cxx::__normal_iterator > > const*, std::vector > >, std::allocator > > > > > std::__lower_bound<__ > > > > >, __gnu_cxx::__normal_iterator > > const*, std::vector > >, std::allocator > > > > >, unsigned sh
1.13e+08 6.4%	1.13e+08 6.4%	inflate_fast
7.67e+07 4.3%	6.68e+07 3.8%	Legacy35tZeroSuppressService::filter(std::vector > const&, unsigned int, float, std::vector >&) const
2.13e+08 12.0%	6.29e+07 3.5%	int TStreamerInfo::ReadBuffer(TBuffer&, TVirtualCollectionProxy const&, TStreamerInfo::TCompInfo* co
6.25e+07 3.5%	6.25e+07 3.5%	__read_nocancel
5.79e+07 3.3%	5.79e+07 3.3%	IdealAdcSimulator::count(double, unsigned int, unsigned int) const
5.14e+07 2.9%	5.14e+07 2.9%	hc2cf_32
5.07e+07 2.8%	5.07e+07 2.8%	__GI_memcpy
1.95e+08 10.9%	4.38e+07 2.5%	void util::LArFFT::DoInvFFT(std::vector >&, std::vector >&)

dupetpc_06_57_00 LArSoft/Dune-FD (prodgenie_nue-dune10kt_1x2x6 detsim)

CPUTIME(Inclusive)	CPUTIME(Exclusive)	Function
7.07e+08 100 %	7.07e+08 100 %	Experiment Aggregate Metrics
1.09e+08 15.4%	1.09e+08 15.4%	IdealAdcSimulator::count(double, unsigned int, unsigned int) const
8.53e+07 12.1%	8.34e+07 11.8%	ExponentialChannelNoiseService::addNoise(unsigned int, std::vector >&) const
7.91e+07 11.2%	7.91e+07 11.2%	__GI_memcpy
7.92e+07 11.2%	6.78e+07 9.6%	Legacy35tZeroSuppressService::filter(std::vector > const&, unsigned int, float, std::vector >&) co
4.11e+08 58.2%	6.69e+07 9.5%	SimWireDUNE::produce(art::Event&)
2.84e+07 4.0%	2.84e+07 4.0%	AdcCodeHelper::subtract(short, float) const
2.00e+07 2.8%	2.00e+07 2.8%	std::vector >::size() const
1.75e+07 2.5%	1.63e+07 2.3%	ProvidedPedestalAdditionService::addPedestal(unsigned int, std::vector >&, float&, float&) const

Vectorization and LArSoft

- Keep looking… what's next?
  - few more lines down:
    - 0.9% – void util::LArFFT::DoInvFFT(std::vector >&, std::vector >&)
    - 0.8% – void util::LArFFT::Convolute(std::vector >&, std::vector >&)

    which led me to larsim, lardata, larwirecell

  - larwirecell and WCT (WireCell toolkit) – Brett Viren
    - FFT and inverse FFT transforms (fftw library)
    - Unit test? Look at source test/test_fft.cxx in http://github.com/WireCell/wire-cell-util
    - docs imply that fftw library already uses SIMD vectorization!
    - no significant progress to build and run examples
      - ups / mrb / spack / scons …
      - decided not worth to learn it all by myself, will ask for help if needed to build

Still looking...

- Still looking for a good LArSoft candidate for vectorization
  - Would like to get a ~simple unit test related to some algorithm relevant for LArSoft. Found RemoveIsolatedSpacePoints in larexamples/Algorithms, with some docs (README.md).
  - Asked Gianluca (last week) about how to run a simple test with it – need space points.
  - Gianluca suggested me to look at Segment3D::GestDist2(...) in module larreco/RecoAlg/PMAlg, file PmaSegment3D.*
  - Good one at first look
    - it is LArSoft code
    - a “leaf” function (does not call others)
    - profiling gives confusing info
      - CPU times: 0.5% (drastically dominated by TensorFlow)
      - CPI (cycles per instruction): ~10%
        (not sure how to interpret the relevance of 10% here)
    - should be able to get results ~quickly (a simple benchmark)

How is GetDist2(...) CPU time?

dupetpc_06_57_00 LArSoft/Dune-FD (prodgenie_nue-dune10kt_1x2x6 reco)

5.60e+10 66.5%	5.06e+08 0.6%	operator()
4.99e+08 0.6%	4.99e+08 0.6%	tbb::internal::generic_scheduler::lock_task_pool(tbb::internal::arena_slot*) const
4.67e+08 0.6%	4.67e+08 0.6%	tbb::internal::cpu_ctl_env::operator!=(tbb::internal::cpu_ctl_env const&) const
4.46e+08 0.5%	4.46e+08 0.5%	pma::Segment3D::GetDist2(TVector2 const&, TVector2 const&, TVector2 const&)
3.81e+08 0.5%	3.81e+08 0.5%	tbb::internal::generic_scheduler::steal_task(tbb::internal::arena_slot&)
8.94e+08 1.1%	2.90e+08 0.3%	pma::Segment3D::SumDist2Hits() const
2.79e+08 0.3%	2.79e+08 0.3%	tbb::internal::prolonged_pause()

dupetpc_06_57_00 LArSoft/protoDune (proton 6GeV reco)

CPI metric (not CPU time!)

CPI	PAPI_TOT_CYC:(I)	PAPI_TOT_CYC:(E)	PAPI_TOT_INS:(I)	PAPI_TOT_INS:(E)	Function
5.50e-01	1.66e+13 100 %	1.66e+13 100 %	3.02e+13 100 %	3.02e+13 100 %	Experiment Aggregate Metrics
4.01e-01	3.85e+12 23.2%	3.85e+12 23.2%	9.60e+12 31.8%	9.60e+12 31.8%	Eigen_tf::internal::gebp_kernel, 8, 4, fals float, long, long, long, long) [clone .const
4.59e-01	1.32e+12 7.9%	1.32e+12 7.9%	2.87e+12 9.5%	2.87e+12 9.5%	pma::Segment3D::GetDist2(TVector2 co
7.68e-01	8.60e+11 5.2%	8.56e+11 5.2%	1.12e+12 3.7%	1.12e+12 3.7%	__ieee754_exp
3.61e-01	7.67e+11 4.6%	7.67e+11 4.6%	2.12e+12 7.0%	2.12e+12 7.0%	Eigen_tf::internal::gebp_kernel, 8, 4, fals float, long, long, long, long) [clone .const
4.16e-01	6.52e+11 3.9%	6.52e+11 3.9%	1.57e+12 5.2%	1.57e+12 5.2%	Eigen_tf::internal::TensorContractionInpu const> const> const, Eigen_tf::ThreadPo const
6.95e-01	1.93e+12 11.6%	6.26e+11 3.8%	3.45e+12 11.4%	9.01e+11 3.0%	pma::Segment3D::SumDist2Hits() const
					Eigen_tf::internal::gemm_pack_rhs cons const, Eigen_tf::ThreadPoolDevice>, std

PMAlg::Segment3D::GetDist2(...)

```
double pma::Segment3D::GetDist2(const TVector2& psrc, const TVector2& p0, const TVector2& p1)
{
        pma::Vector2D v0(psrc.X() - p0.X(), psrc.Y() - p0.Y());
        pma::Vector2D v1(p1.X() - p0.X(), p1.Y() - p0.Y());
        pma::Vector2D v2(psrc.X() - p1.X(), psrc.Y() - p1.Y());

        double v1Norm2 = v1.Mag2();
        if (v1Norm2 >= 1.0E-6) // >= 0.01mm
        {
            double v0v1 = v0.Dot(v1);
            double v2v1 = v2.Dot(v1);
            double v0Norm2 = v0.Mag2();
            double v2Norm2 = v2.Mag2();

            double result = 0.0;
            if ((v0v1 > 0.0) && (v2v1 < 0.0))
            {
                    double cosine01_square = 0.0;
                    double mag01_square = v0Norm2 * v1Norm2;
                    if (mag01_square != 0.0) cosine01_square = v0v1 * v0v1 / mag01_square;

                    result = (1.0 - cosine01_square) * v0Norm2;
            }
            else // increase distance to prefer hit assigned to the vertex, not segment
            {
                    if (v0v1 <= 0.0) result = 1.0001 * v0Norm2;
                    else result = 1.0001 * v2Norm2;
            }
            if (result >= 0.0) return result;
            else return 0.0;
        }
        else // short segment or its projection
        {
                double dx = 0.5 * (p0.X() + p1.X()) - psrc.X();
                double dy = 0.5 * (p0.Y() + p1.Y()) - psrc.Y();
                return dx * dx + dy * dy;
        }
}
```

Vector arithmetics are usually easy to SIMD-vectorize.
I will create a vectorized version of this function and a benchmark for comparisons.

Next steps

- Based on benchmarking code on VecCore package (quadratic equation, see my talk from Dec.05)
- Adapting that benchmarking code to run with GetDist2(...) function
- Currently vectorizing this simple example for performance comparisons
- Plan is to bring vectorized code back into LArSoft for comparison (expected improvement to be small, unless TensorFlow timing is reduced by use of -mavx or -msse4.2)

Current status

- A few Dune and LArSoft functions identified as candidates
  - Profiling shows that our current DUNE candidates are promising (~10% time)
  - Started with a LArSoft Segment3D::GetDist2(…) function for a first evaluation
  - Looking for other LArSoft functions to vectorize
- I am currently working on the vectorized version of the GetDist2(…) function, and on a benchmark to compare performance
  - Using VecCore structure for my tests, for convenience
  - Porting vectorized code back to LArSoft may require dependence on VecCore types
  - The first one is the hardest… hopefully other ones will be able to reuse benchmarking infrastructure
  - Probably other functions will also need to be vectorized before a performance effect can be observed in client code
- In parallel, see if there is a version of LArSoft built with -mavx2 and/or -msse4.2 for profiling (big effect expected on TensorFlow functions, as Soon suggested)
- Planning to use vectorize DUNE functions (larger relative contributions)
- First preliminary results expected soon (in a couple of weeks?)

Perspectives

- Our goal is to evaluate whether SIMD vectorization can bring a significant performance gain for LArSoft utilities

- How much improvement to expect?
  - reminder: performance gains are harder and harder to reach (using profiling as a guide)
  - example doc suggests that some important performance tips are already part of LArSoft code
  - it is possible that an experiment's code (e.g. DUNE) will show more potential for performance improvements than the LArSoft core
  - the idea is that if a few methods vectorized can bring a 1%-2% global improvement, a deeper vectorization work can bring 5x-10x **more** improvements.
  - important question is: how much work is needed to produce a significant result in experiment's jobs?

- Our preliminary results will hopefully point the way to go