Profiling Work

in Preparation for Optimization with GPUs

Marc Paterno

January 23, 2024

Why this work?

- Goal is to accelerate DUNE physics processing using GPUs, for some algorithm in LArSoft. This aligns with the LArSoft "high priority" goal list.
- Multi-step process:
  1. Identify a likely candidate module from LArSoft.[1]
  2. Collect performance data to see where the code is taking the most time.
  3. Improve the serial algorithm performance.
  4. Parallelize the serial algorithm.
  5. If the result is still insufficient, adapt the parallel algorithm for GPU usage.
- `PDFastSimPAR` was the clear most time-consuming module used in the DUNE workflows that is found in LArSoft.

[1] Thank you to Tom Junk and Laura Paulucci for their guidance.

Workflow based on a standard DUNE FD simulation workflows

`lar` configurations using this module:

- `prodbackground_radiological_decay0_dunevd10kt_1x8x14.fcl`
- `prodmarley_nue_flat_radiological_decay0_dunevd10kt_1x8x14_3view_30deg.fcl`

Geant4 simulation used as input:

- `/pnfs/dune/persistent/users/lpaulucc/leprodtests/prodradiological_decay0_dunevd10kt_1x8x14_gen.root`

To make profiling data collection easier, I broke the workflow into two parts:

- Everything before the `PDFastSimPAR` module, which I write to an art/ROOT file, and
- the `PDFastSimPAR` module run alone, on the output from the previous step.

Profiling data collection

- I am using the Intel VTune performance analysis suite of tools.
- Running a `prof` build on a SLF7 Linux machine.
- Hardware is Skylake AVX512.
- Standard `prof` build does not activate the compiler options to make full use of the instruction set. Essentially no automatic vectorization is done.
- VTune collects a huge amount of data; I run on only 1 event to keep the data analysis feasible.
- Happily, previous analysis shows that the time taken to process events in the given file is very uniform.

First profiling results

- Biggest hotspot in LArSoft code is `phot::fast_acos`, for a total of 7.111 seconds (out of 48.22 seconds `PDFastSimPAR::produce`).
- Called from two places within the code.

phot::fast_acos

- Implementation from Approximations for Digital Computers, C. Hastings, Jr, published by Princeton University Press (1955), with flourishes that seem to be related to an implementation posted by NVIDIA.
- Invented before the IEEE floating point standard was devised.
- Not all computers at that time had instruction sets that included trig functions.
- Do we need such a thing today?

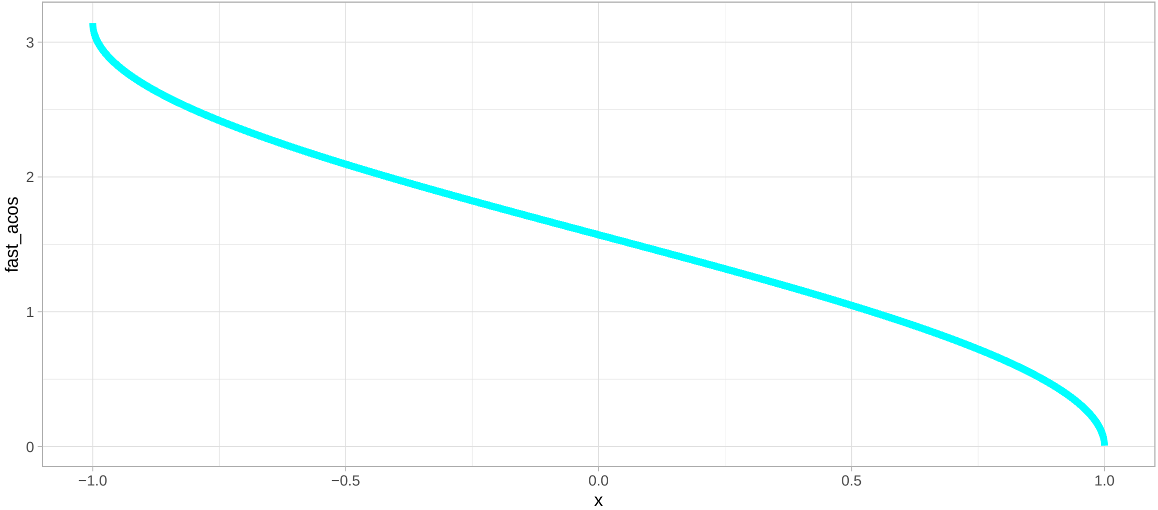
Microbenchmarking results

- Data collected on the same machine as used for VTune results.

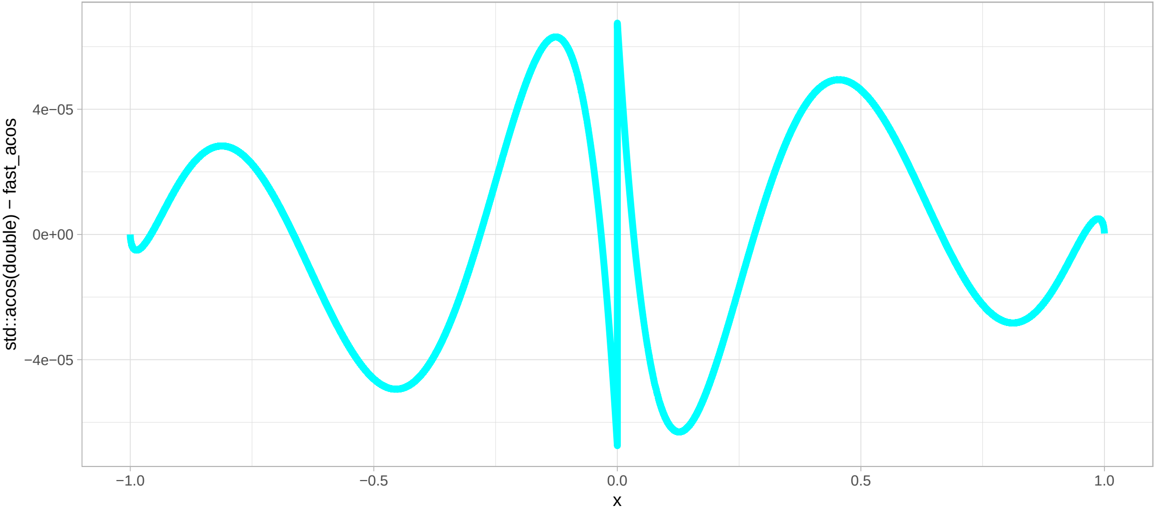
ns/op	ins/op	branches/op	name	relative
38.00	217	35	acosd	3.072
22.73	114	20	acosf	1.838
12.37	85	9	fast_acos	1.000

- `fast_acos` is clearly faster than even the single-precision math library function.
- Less time per operation, because of fewer instructions and fewer branches encountered.

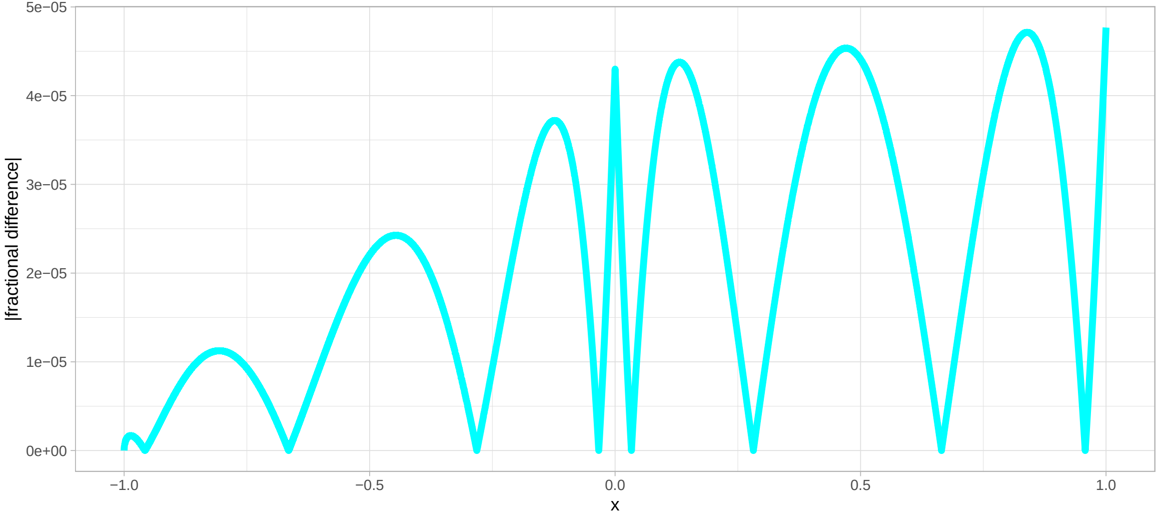
phot::fast_acos shape of the curve



phot::fast_acos difference from C library



phot::fast_acos relative difference from C library



Is phot::fast_acos sufficiently accurate?

- If not, then replacing it with `std::acos` is trivial; the cost is a factor of 1.8 in the time taken for this operation.
- If yes, then I have some modifications for you to consider...

New implementations

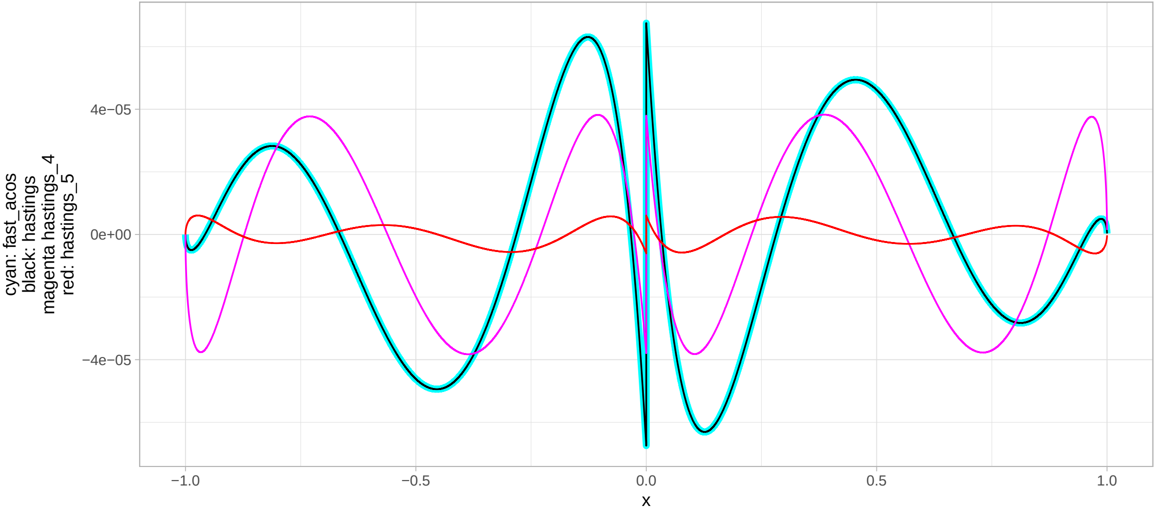
- `hastings_acos` is the same algorithm, stripped of flourishes that make sense for GPUs but are counterproductive on CPUs.
- `hastings_acos_4` is the same mathematical form with slightly improved constants. This results in an improved approximation with identical instruction counts, branches, and execution time.
- `hastings_acos_5` is a similar mathematical form, with one more term in the approximation. It yields a still better approximation, at some cost in instruction counts and thus execution time.

Benchmarking results

ns/op	ins/op	branches/op	name	relative
38.00	217	35	acosd	3.072
22.73	114	20	acosf	1.838
12.37	85	9	fast_acos	1.000
8.26	61	8	hastings_acos_5	0.668
7.06	57	8	hastings_acos	0.571
6.94	57	8	hastings_acos_4	0.561

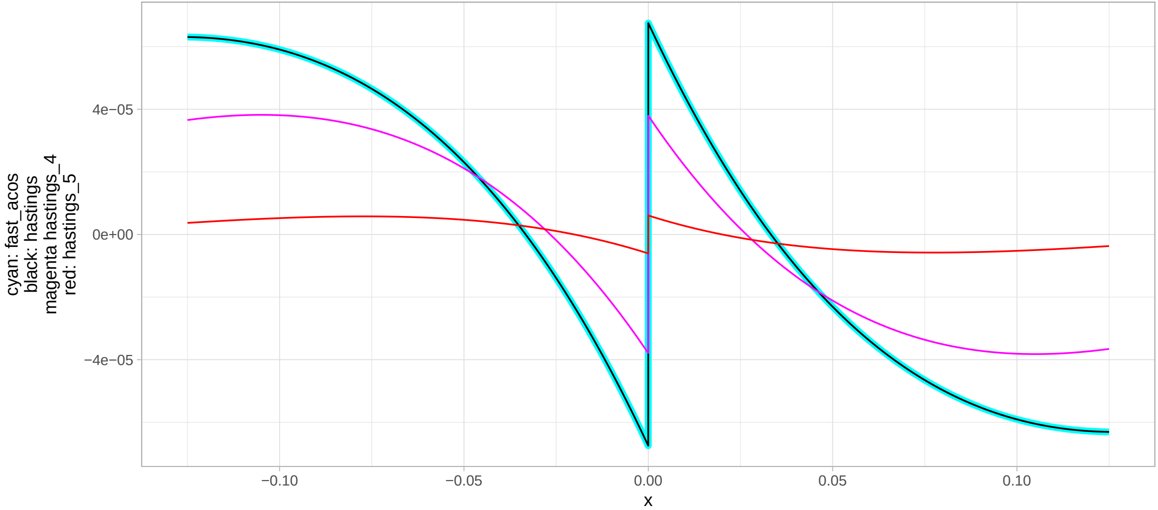
- The time difference between `hastings_acos` and `hastings_acos4` is not signficant. The instruction counts and branch counts are the same; the generated assembly differs only in the values of the constants loaded into memory. The difference in time reflects the precision with which nanobench can measure the code.

Comparison of absolute differences in calculated results



Fermilab

Comparison of absolute differences in calculated results



VTune results

- Numbers are (inclusive) times, in seconds, spent in the named function, for code using the given algorithm.

algorithm	acos	PDFastSimPAR::produce
fast_acos	7.111	48.22
hastings_acos_5	5.310	45.60
hastings_acos_4	4.130	44.71

Backup slides.

```
 double fast_acos(double x) {
  double negate = double(x < 0.);
  x = std::abs(x);
  // following line is  min(1.,x)
  x -= double(x > 1.) * (x - 1.);
  double ret = -0.0187293;
  ret = ret * x;
  ret = ret + 0.0742610;
  ret = ret * x;
  ret = ret - 0.2121144;
  ret = ret * x;
  ret = ret + 1.5707288;
  ret = ret * std::sqrt(1. - x);
  ret = ret - 2. * negate * ret;
  return negate * M_PI + ret;
}
```

```
double hastings_acos(double xin) {
  double const x = std::abs(xin);
  double const a0 =  1.5707288;
  double const a1 = -0.2121144;
  double const a2 =  0.0742610;
  double const a3 = -0.0187293;
  double ret = a3;
  ret *= x;
  ret += a2;
  ret *= x;
  ret += a1;
  ret *= x;
  ret += a0;
  ret *= std::sqrt(1.0-x);
  if (xin >= 0) return ret;
  return M_PI - ret;
}
```

How I generated hastings_acos_4 and hastings_acos_5

- The functional form of all the “fast” algorithms is:

$$\cos^{-1} x \approx \sqrt{1-x}(a_0 + x(a_1 + x(a_2 + \ldots)))$$

- The coefficients a_i are found by minimizing Δ:

$$\Delta = \max\left|f(x) - \cos^{-1}(x)\right|, \quad \text{for } -1 \le x \le 1$$

- The original algorithm has the fit parameters calculated in single precision.
- hastings_acos has identical parameters but fewer operations & branches.
- hastings_acos_4 has fit parameters calcluated to double precsion, and is otherwise identical to hastings_acos_4.
- hastings_acos_5 uses 5 fit parameters calculated to double precsion.
- Using 6 parameters yielded a slower algorithm but no better accuracy.