Fermilab

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

LArSoft vectorization tests: status report

Guilherme Lima

LArSoft Coordination Meeting

June 19, 2018

Recalling the big picture

- On my last report here (March 13th), I presented plans to vectorize a simple function (GetDist2, in larreco's class pma::Segment3D)
- I then vectorized it using SIMD-vector types from VecCore package, and validated its results using two different vectorization libs (Vc and UME::SIMD)
  - interface change needed, arguments to function are vectorized types.
- I measured a ~3.2x speedup using Vc lib (AVX: theo.max of 4)
- Next steps: demonstrate its use from inside a real LArSoft binaries
  - add VecCore and Vc library to building system
  - modify calls to vectorized function (multi-point calculations)
  - check for measurable speedup (small CPU time of 0.5%)
  - vectorize other functions (candidates from DUNE code)

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.
Created a vectorized version of this function (see next slide)
and a benchmark for comparisons

Generic (vectorized) GetDist2(...) function

```
//=====  explicit SIMD code

template <typename Real_v>
VECGEOM_FORCE_INLINE
void GetDist2SIMD(const Vector3D<Real_v>& ps, const Vector3D<Real_v> &p0,
                  const Vector3D<Real_v> &p1, Real_v *__restrict__ result)
{
  using Bool_v  = vecCore::Mask_v<Real_v>;

  const Vector3D<Real_v> v0(ps - p0);
  const Vector3D<Real_v> v1(p1 - p0);
  const Vector3D<Real_v> v2(ps - p1);

  // most common case: if (v1Norm2 >= 1.0E-6) {  // >= 0.01mm
  const Real_v zero = Real_v(0.0);
  const Real_v v0v1 = v0.Dot(v1);
  const Real_v v2v1 = v2.Dot(v1);
  const Real_v v0Norm2 = v0.Mag2();
  const Real_v v1Norm2 = v1.Mag2();
  const Real_v v2Norm2 = v2.Mag2();

  const Bool_v opposite  = (v0v1 > zero) && (v2v1 < zero);
  const Real_v mag01_square = v0Norm2 * v1Norm2;
  const Real_v cosine01_square = v0v1 * v0v1 / NonZero(mag01_square);
  *result = (Real_v(1.0) - cosine01_square) * v0Norm2;

  const Bool_v nonnegv2v1 = (v2v1 >= zero) && (!opposite);
  const Bool_v nonposv0v1 = (v0v1 <= zero) && (!opposite);
  vecCore::MaskedAssign( *result, nonnegv2v1, Real_v(1.0001) * v2Norm2);
  vecCore::MaskedAssign( *result, nonposv0v1, Real_v(1.0001) * v0Norm2);
  vecCore::MaskedAssign( *result, *result <= zero, zero);

  Bool_v close = (v1Norm2 < Real_v(1.0e-6));
  if (!vecCore::MaskEmpty(close)) { // for a short p1-p0 segment or its projection
    Vector3D<Real_v> temp = 0.5 * (p0 + p1) - ps;
    vecCore::MaskedAssign( *result, close, temp.Mag2());
  }
  return;
}
```

templated on a FP type → scalar type (float, double), or vector type (Float_v, Double_v)

consts help compiler optimizations

avoid divisions by zero without adding if(cond)

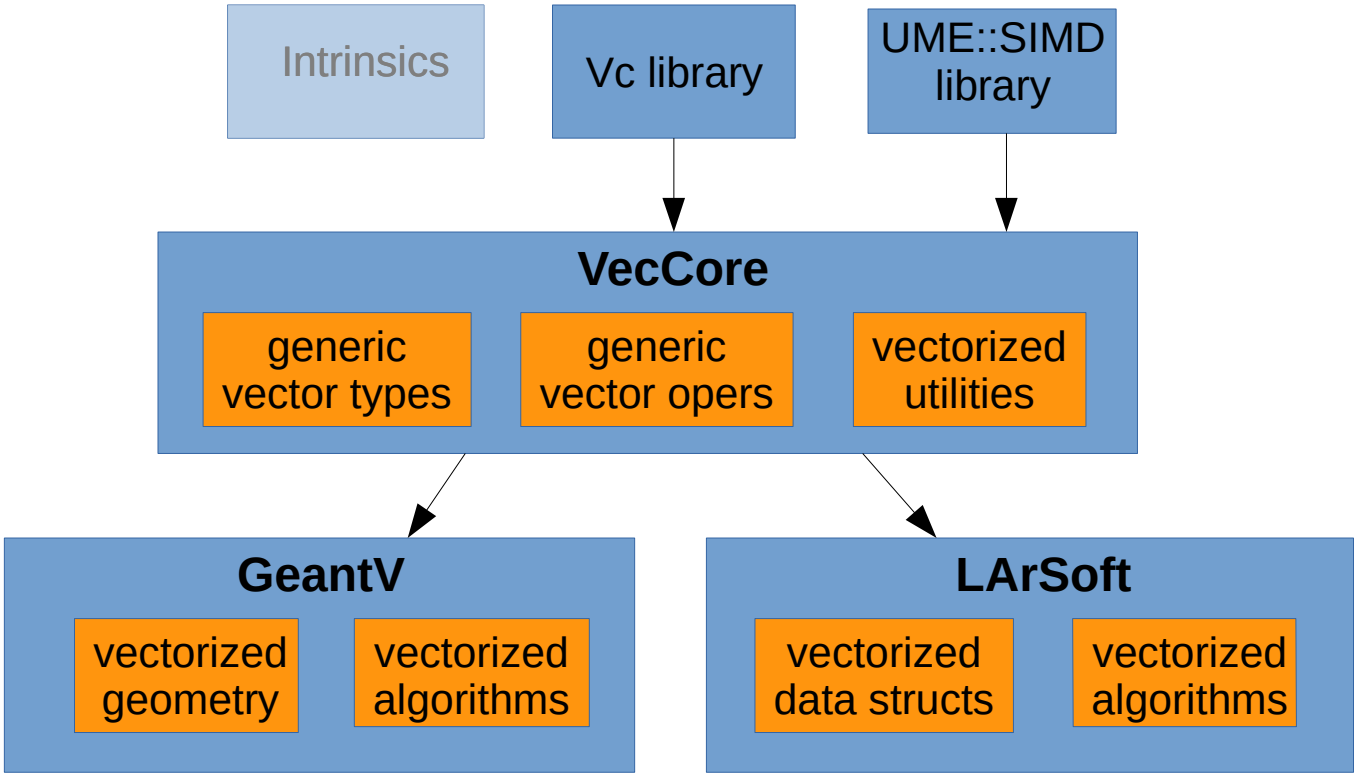
masks used as conditions...

...in MaskedAssigns to replace if(cond)

This version with vector types processes large numbers of points 3x faster!

UPS packaging

- Now working on the UPS packaging of VecCore, Vc and Ume::SIMD packages
  - using Tom Junk's script, modified as needed, to create UPS package structure for each package
  - UPS testing: list, setup, unsetup, environment needed
  - UPS vs CMake standard environment variables
  - Currently adapting and testing VecCore's CMake-based builds



UPS packaging

- Some relevant issues and questions:
  - UME::SIMD library is header-only (NULL flavor)
  - static (Vc lib) vs. shared libs (UPS) – any road blocks?
  - setup/unsetup/list tests ok. Any other requirements?
  - c++ standard: e15 (c++14) and e17 (c++17)
  - library compilation tags for vectorization (-msse / -msse4.2 / -mavx) → propagated to client packages
    - use multiple UPS tags for sse vs. avx?
  - vector capabilities available on the hardware
    - Vc build checks for vector capabilities of machine used during build
    - many grid nodes are not avx-capable
    - may be able to build Vc with all capabilities built-in, and then test target machine on-the-fly (via compilation flags or run-time) to avoid using incompatible operations (to be verified)

Summary

- Preliminary results suggest that good speedups are possible using SIMD vectorization
- Some work is needed in the LArSoft build system to make vectorized types available within LArSoft
- I am learning to do that work myself, but expert help can make a big difference
  - the UPS packaging seems to be under control (some tweaks may still be needed thoughs)
  - how to make UPS packages available in fnkits.fnal.gov
  - I need help with the LArSoft and/or DUNE build system, to include VecCore and Vc library headers and libs, and to adjust compilation switches
- In parallel, next to be vectorized:

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::ve
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

Backup slides

Vectorization libraries

- Vectorization libraries provide high level types to explicitly leverage SIMD vectorization without sacrificing portability, readability or maintainability
- User code is written in terms of vectorized types and preprocessor macros provided by vectorization library
- Undesired issue: strong dependence on a third-party vectorization library
  - mitigated using VecCore (see next slides)
- Examples of libraries:
  - M.Kretzman's Vc library
  - P.Karpinski's Ume::SIMD library
  - Agner Fog's Vector Class library
  - several others