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
August 28, 2017

# Vectorization and LArSoft

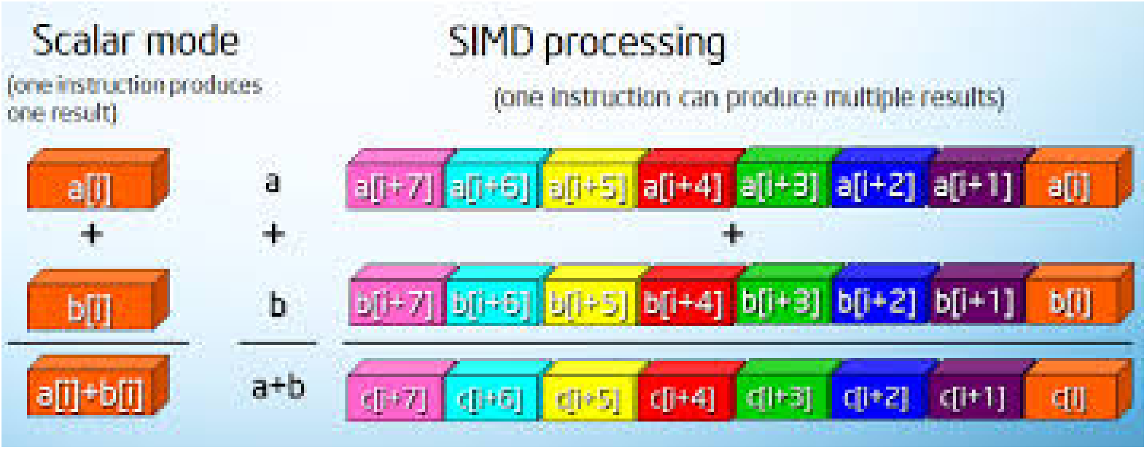
- Goals
  - Use vectorization to improve LArSoft performance
  - Outline of this talk

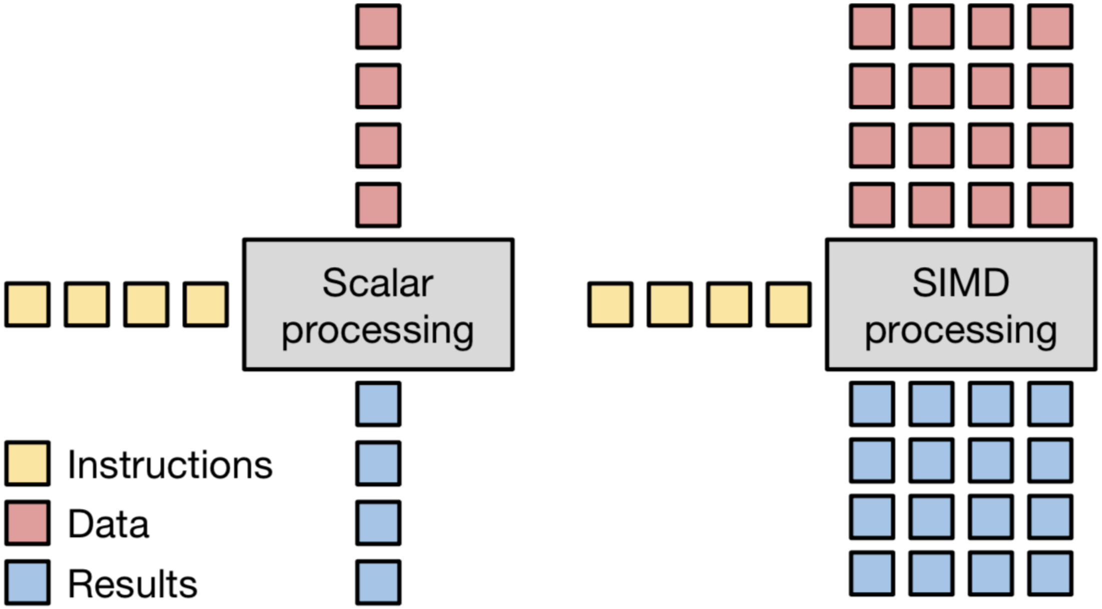
    * SIMD vectorization
    * VecCore library

    * Plans and status

# SIMD Vectorization

- Traditional programs operate in scalar mode
- Modern hardware can use SIMD vectorization for instruction-level parallelism
- Modern compilers can *auto-vectorize* binaries in very special cases
  - very simple loops with well-aligned arrays
- Developers can significantly improve the vectorization efficiency using explicit vectorization techniques



# SIMD Vectorization

- At the lowest level, SIMD vectorization consists of
  - Loading data onto the vector registers (gather?)
  - Perform SIMD-vector arithmetic and logic operations
  - Save data from registers back into memory (scatter?)

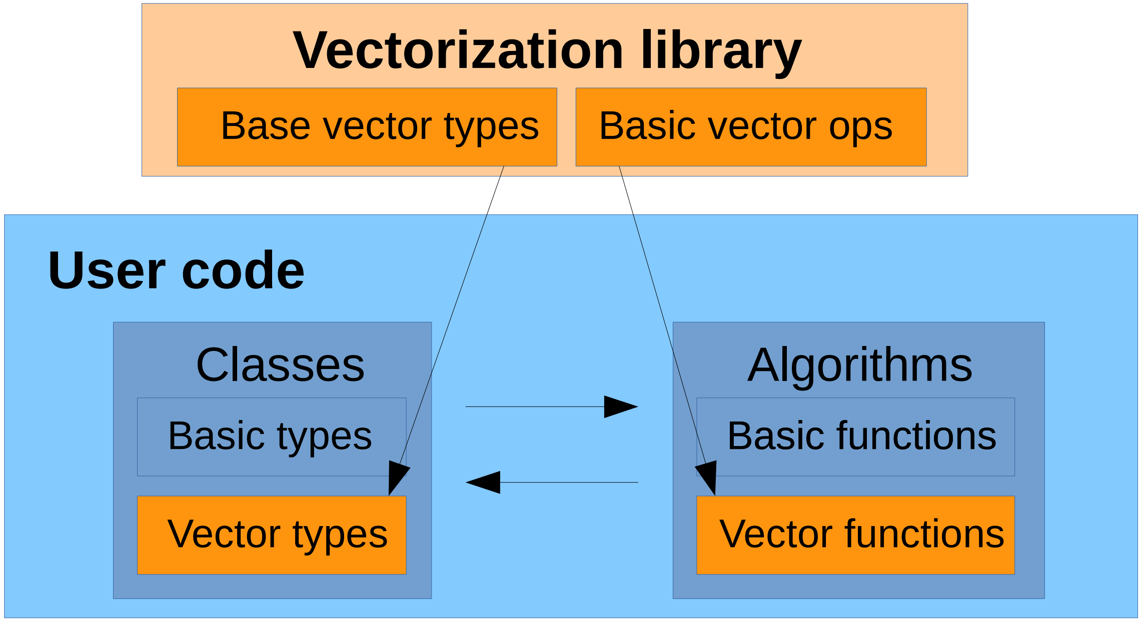
    → gathers/scatters overhead can be minimized by redesigning the data structures

- Minimize performance limitations (vectorization inefficiencies)
  - alignment issues
  - data locality
  - code locality (cache misses)
  - branching (if/then/else, switch/case, early returns)
  - etc.

- Vectorization procedure easier using *vectorization libraries*

# 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



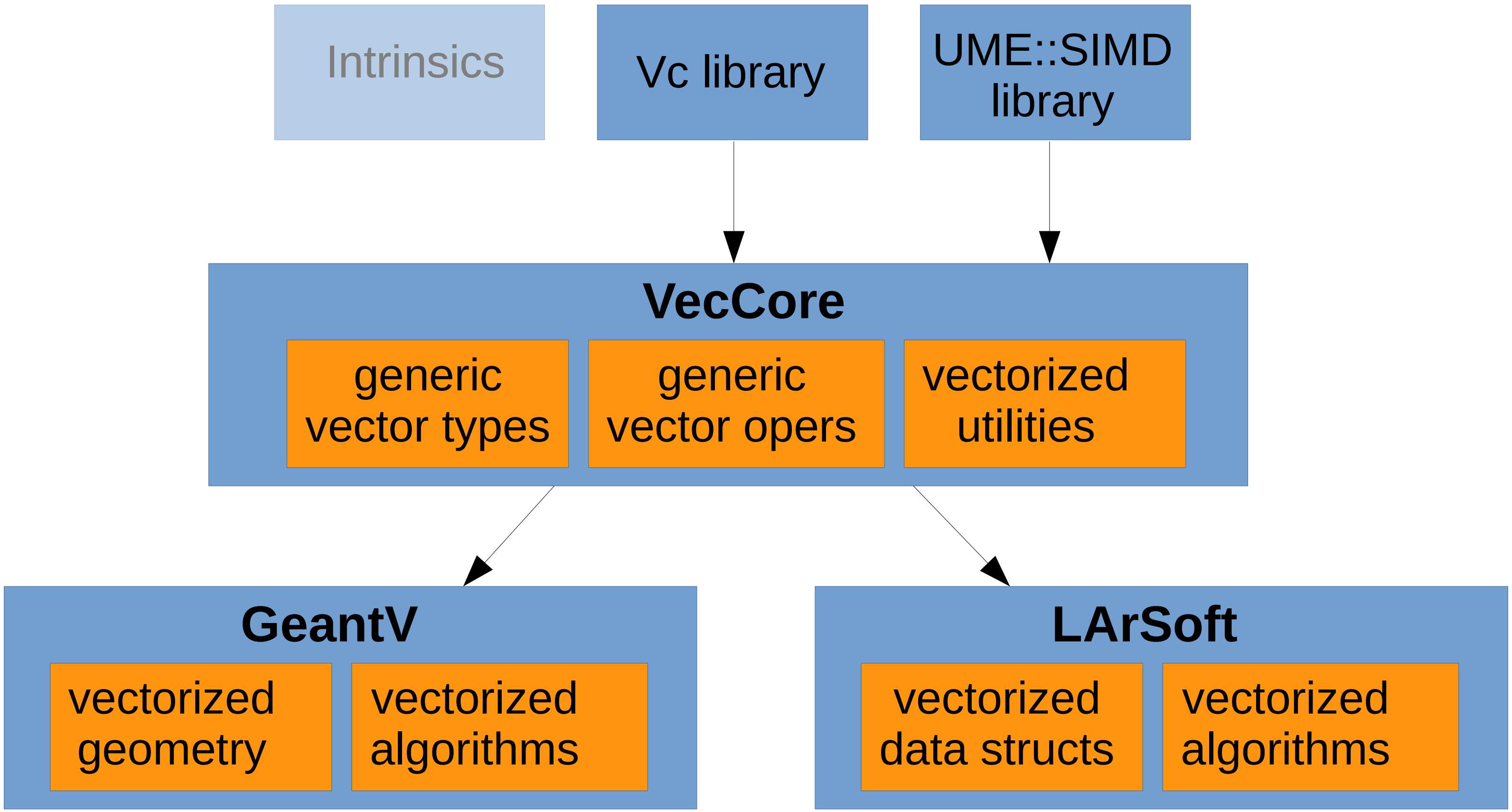
# Introducing VecCore

- Developed within GeantV project
- Currently being integrated into ROOT
- Provides a uniform interface for SIMD vectorization
  - Backends form a coherent set of types to be used together
  - Arithmetics, comparisons, logical operators
  - Vectorized math functions
  - Masking/blending operations
  - Gather/Scatter operations
  - Support for multiple architectures without code duplication

- Support multiple backend implementations
  - Scalar/CUDA
  - Vc Library — https://github.com/VcDevel/Vc
  - UME::SIMD — https://github.com/edanor/umesimd

- See these slides for more information about VecCore

# Introducing VecCore

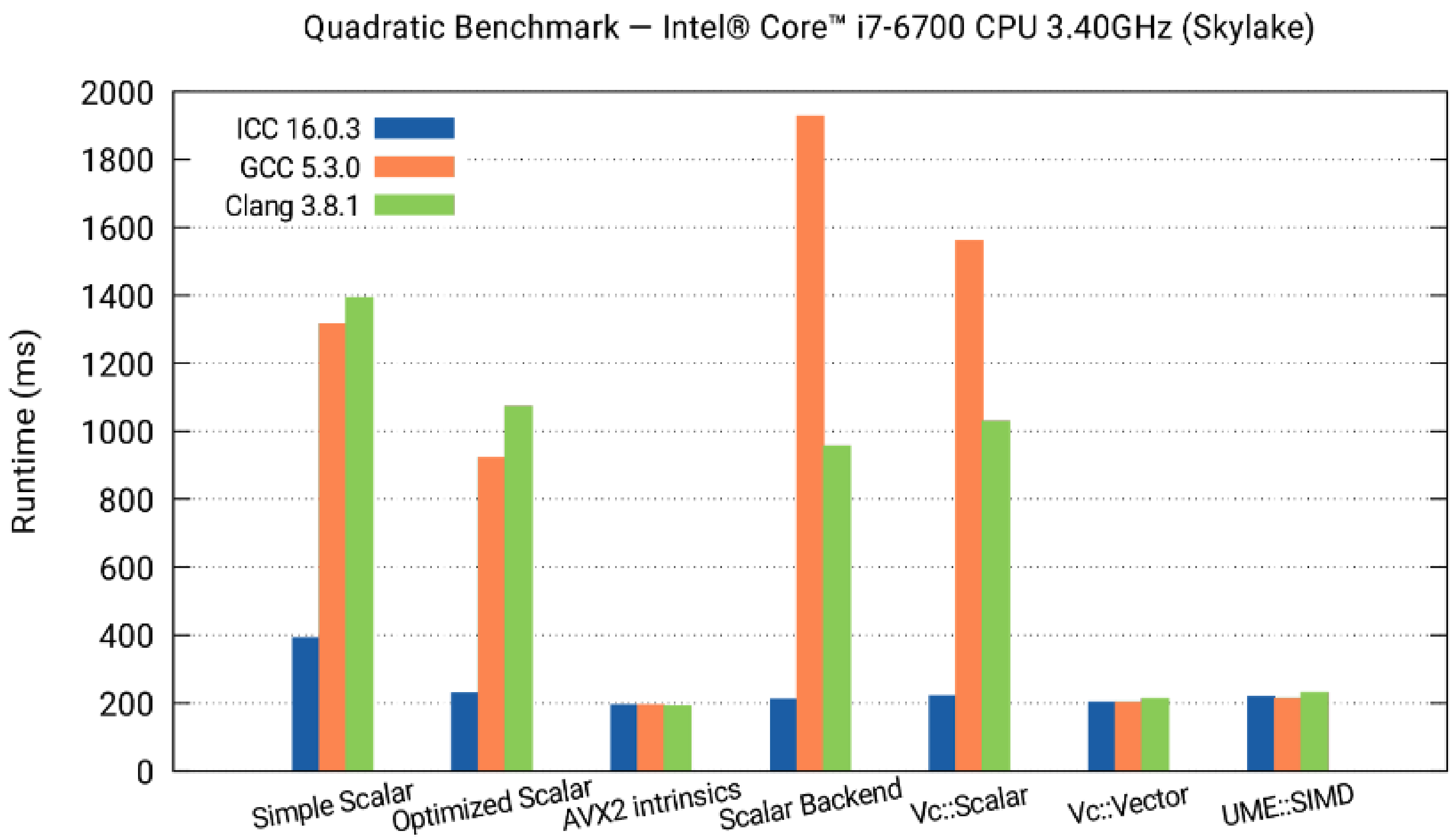


# VecCore details

- Source: VecGeom/VecCore/
- Generic vectorized types
  - Real_v, Float_v, Double_v, Int_v, Int16_v, Int32_v, Int64_v, UInt_v, ..., UInt64_v
    **→ relevant algorithms re-written in terms of these generic vectorized types**
- Vectorized operations
  - Arithmetics, MaskedAssign(), Blend(), IsFull(), IsAny(), isEmpty(), EarlyReturnsAllowed()
- Implementation backends
  - Scalar, ScalarWrapper
  - VcScalar, VcVector, VcSimdArray<N>
  - UMESimd, UMESimdArray<N>

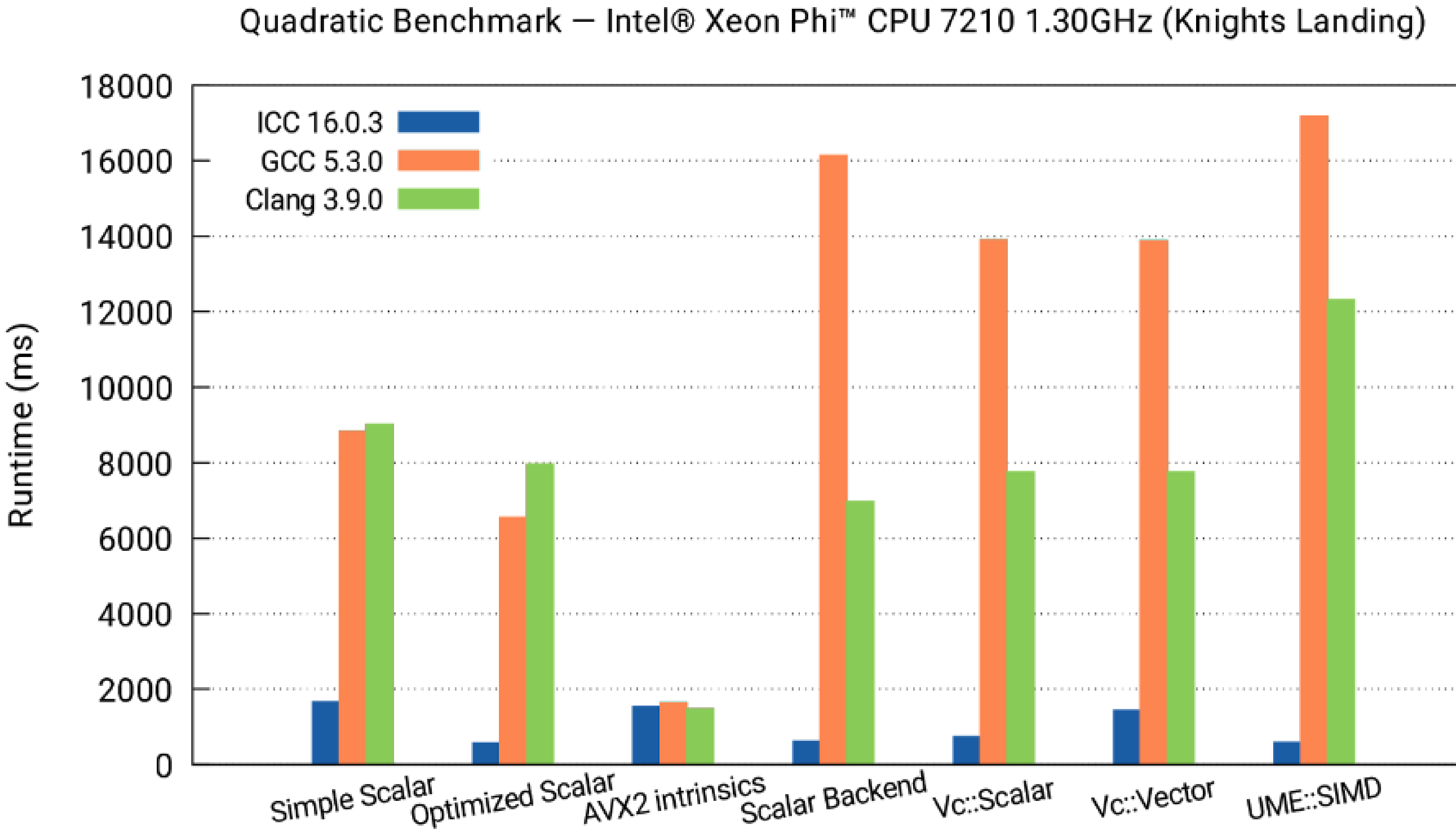
- Implementation is selected at compilation time via CMake switches (if supported by the system)
  - -DVC=[ON|off]  -DUMESIMD=[on|OFF]  -DCUDA=[on|OFF]
  - Note that carefully designed programs can use multiple backends at the same time (e.g. quadratic solver)
  - Also supports GPU (through CUDA)

# Quadratic solver: performance




Tests by Guilherme Amadio (CERN)

# Quadratic solver: performance




Tests by Guilherme Amadio (CERN)

# LArSoft vectorization plans

- Familiarity with LArSoft environment
- Introduce VecCore library into the build
  - need some help for fast progress (GP)
- Identify LArSoft candidates for initial vectorization tests
  - ES, GP: detector simulation and hit finding
  - SYJ: profiling results
- Preliminary tests with localized changes
  - Benchmarking tools?
- Consider redesigned data structures and adapted interfaces
  - reduce gather/scatter overhead needed for vectorization
  - our experience with GeantV shows that the gains from data and code locality can be quite significant