



Parallelization in LArSoft reconstruction - SciDAC4 developments

Giuseppe Cerati (FNAL)

LArSoft Multi-threading and Acceleration Workshop

Mar 2, 2023

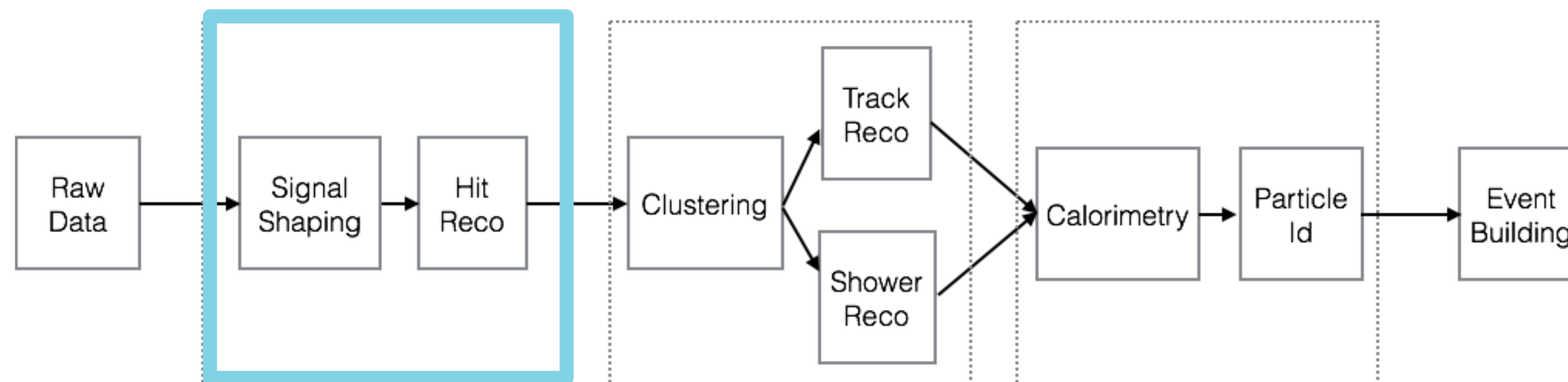
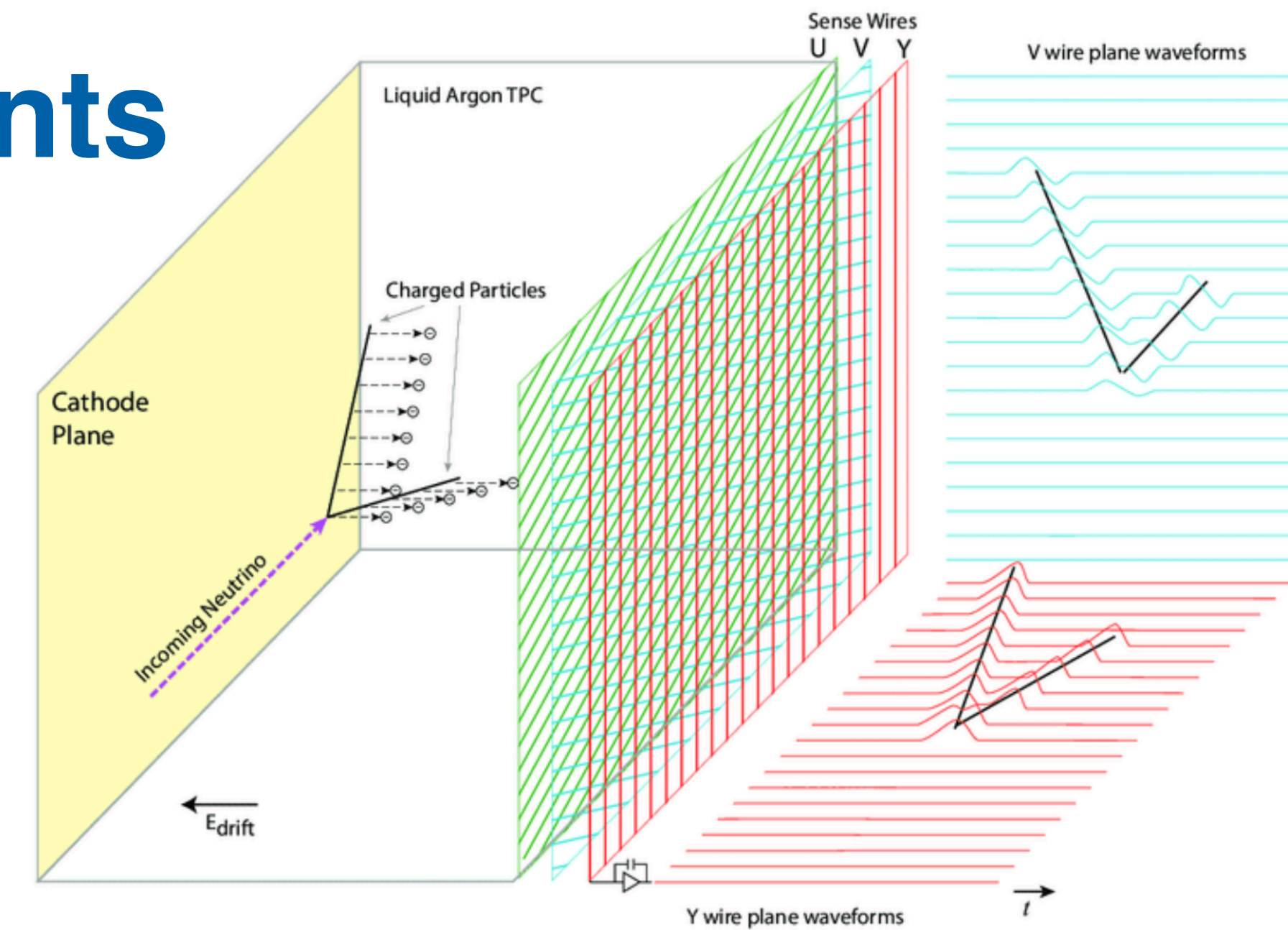
Project Goals



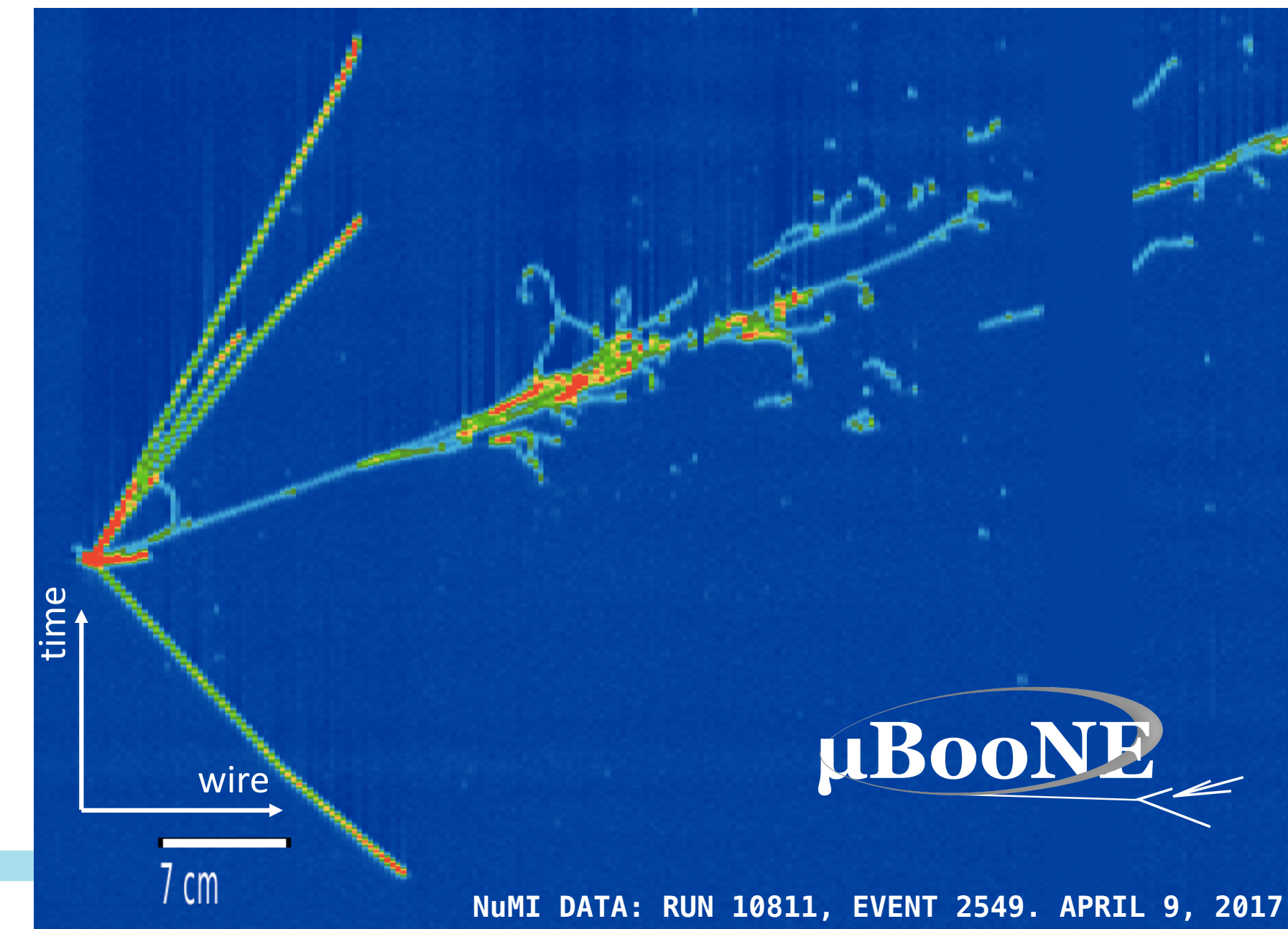
- “HEP event reconstruction with cutting edge computing architectures” project supported by the **DOE SciDAC-4 program**
 - <https://computing.fnal.gov/hepreco-scidac4/>; <https://www.scidac.gov/>
- Collaboration between physicists at Fermilab and computer scientists at UOregon
- Mission: **accelerate HEP event reconstruction using modern parallel architectures**
- Focus on two areas:
 - Novel parallel algorithm for charged particle **tracking in CMS**
 - Pioneer similar techniques for **reconstruction in LArTPC detectors**
- Goals of the project are the following:
 1. Identify key algorithms for the outcome of the experiments that dominate reconstruction time
 2. Re-design the algorithms to make efficient usage of data- and instruction-level parallelism
 3. Deploy the new code in the experiments’ framework
 4. Explore execution on different architectures and platforms

Reconstruction for LArTPC ν experiments

- Charged particles produced in neutrino interactions ionize the argon, ionization electrons drift in electric field towards anode planes
- Sense wires detect the incoming charge, producing beautiful detector data images
- Reconstruction in LArTPC experiments is **challenging** due to unknown interaction point, many possible topologies, noise, contamination of cosmic rays
 - Takes O(minutes)/event in MicroBooNE
 - ICARUS $\sim 5\times$ bigger, DUNE Far Detector O(100) \times bigger
- LArTPC detectors are modular in nature \rightarrow **parallelism!**



Typical reconstruction chain for LArTPC experiments



- Parallelization of Hit Finder Algorithm

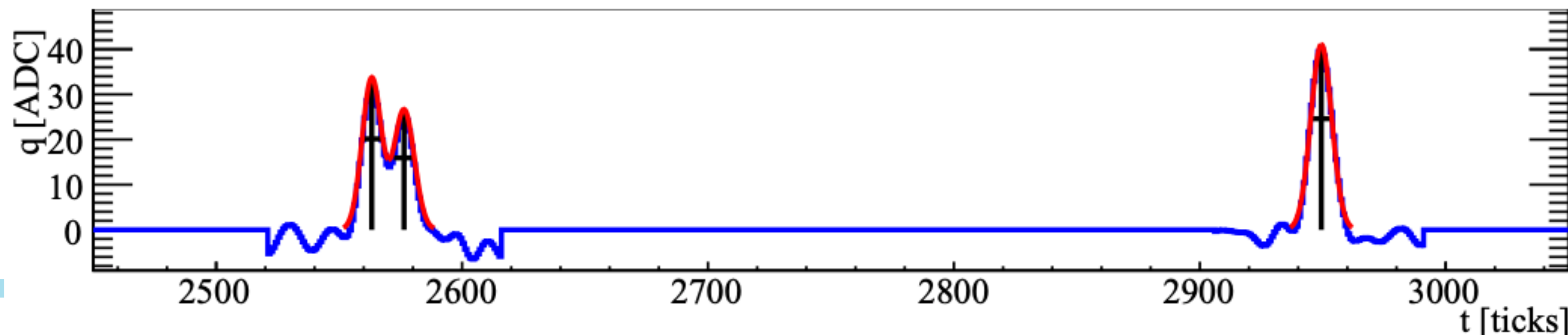
Initial Study: Hit Finding

Optimizing the hit finding algorithm for liquid argon TPC neutrino detectors using parallel architectures

Sophie Berkman (Fermilab), Giuseppe Cerati (Fermilab), Kyle Knoepfel (Fermilab), Marc Mengel (Fermilab), Allison Reinsvold Hall (Fermilab) et al. (Jul 1, 2021)

Published in: *JINST* 17 (2022) 01, P01026 • e-Print: [2107.00812](https://arxiv.org/abs/2107.00812) [physics.ins-det]

- MicroBooNE: $\sim 8\text{k}$ wires readout at 2 MHz, deconvolved wire signals are **Gaussian pulses**
- Hit finding: **identify** pulses and determine their **peak** position and **width**
- It can be used to take a **significant fraction** of the reconstruction workflow
 - few percent to few tens of percent depending on the experiment
- Wires (and ROIs) can be **independently processed**:
 - algorithm suitable to demonstrate speedup potential by parallelizing LArTPC reconstruction



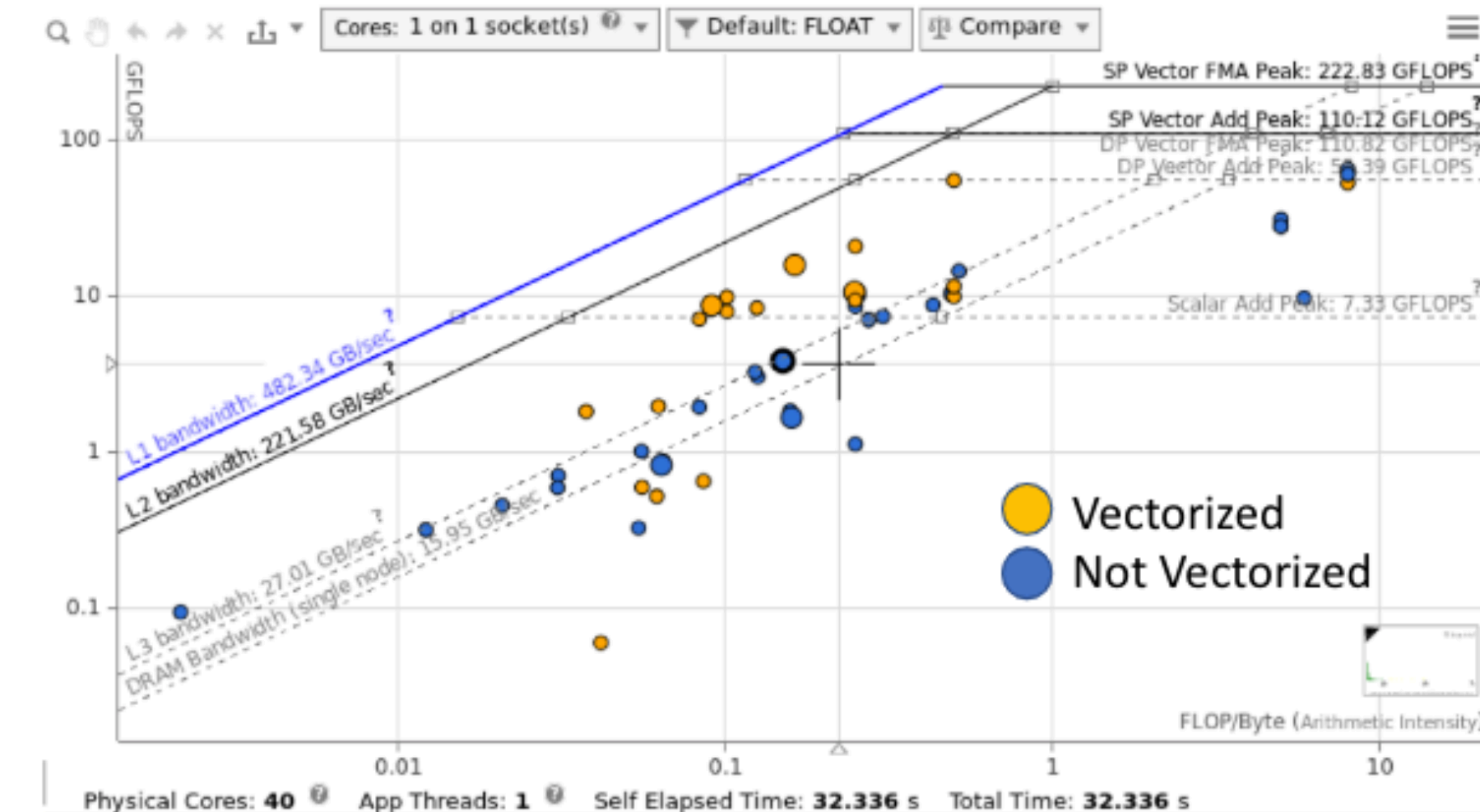
Standalone Implementation

- Replicated LArSoft hit finder as **standalone code** for testing and optimization
- Replaced Gaussian fit based on Minuit+ROOT with a local implementation of **Levenberg-Marquardt minimization**
 - gradient descent when far from minimum and Hessian minimization when close to it
 - implementation based on “Data Reduction and Error Analysis for the Physical Sciences”
 - include boundaries on fit parameters for better fit stability
- Early tests showed that standalone implementation is **~8x faster** than default
 - **before optimizations** and without any vectorization or multi-threading

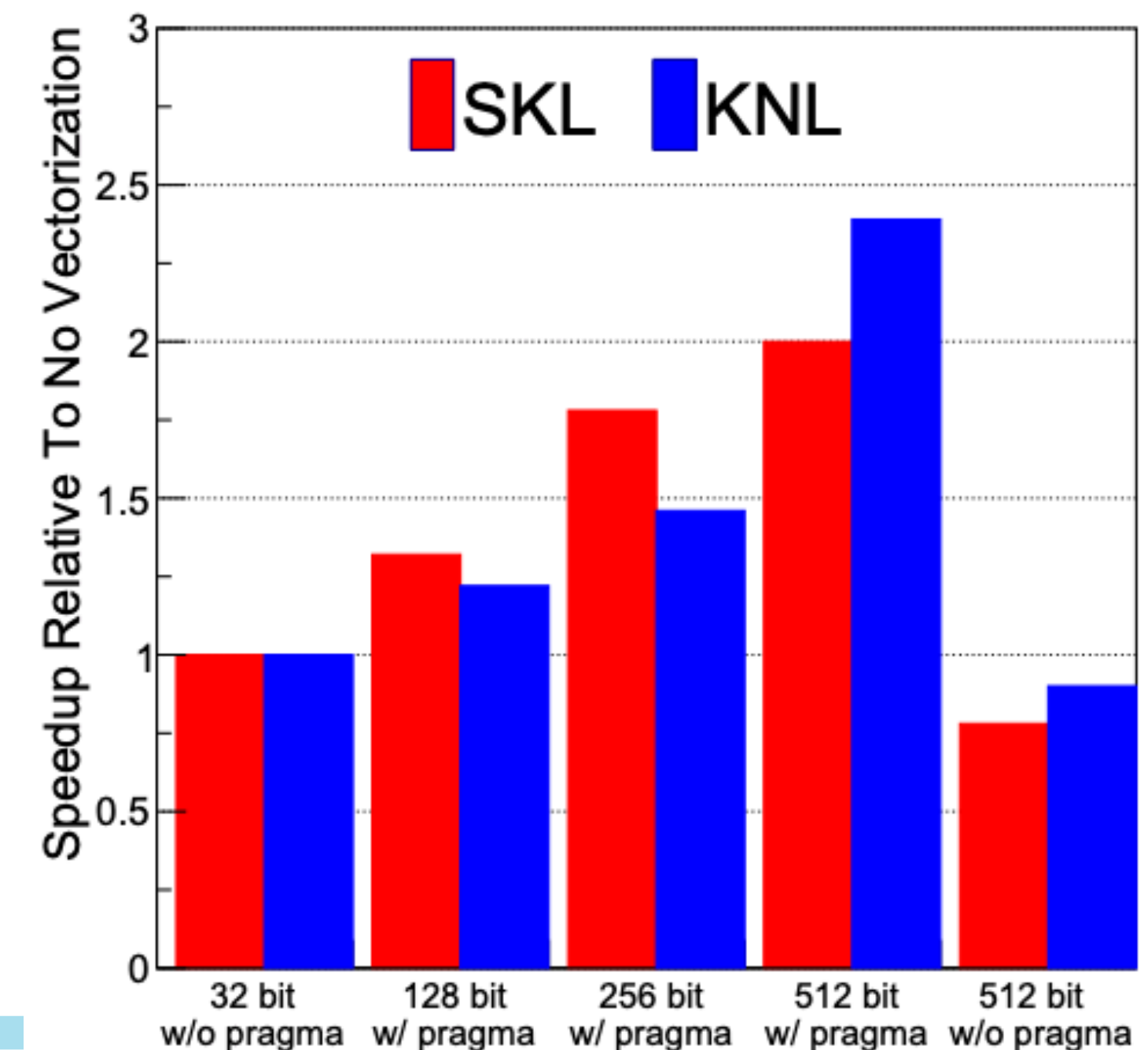
Vectorization Results

- Profiling the code (e.g. roofline) shows that most of the time is still spent in the minimization algorithm
 - number of iterations needed to converge is variable: difficult to vectorize across multiple hit candidates.
- We choose to **vectorize specific loops** within the algorithm, typically **across data bins**
 - main limitations: only a subset of the code is vectorized, number of bins is same order as vector unit size
- About **2x speedups**, both on Skylake Gold (SKL) and KNL when compiling with icc+AVX-512

Roofline analysis



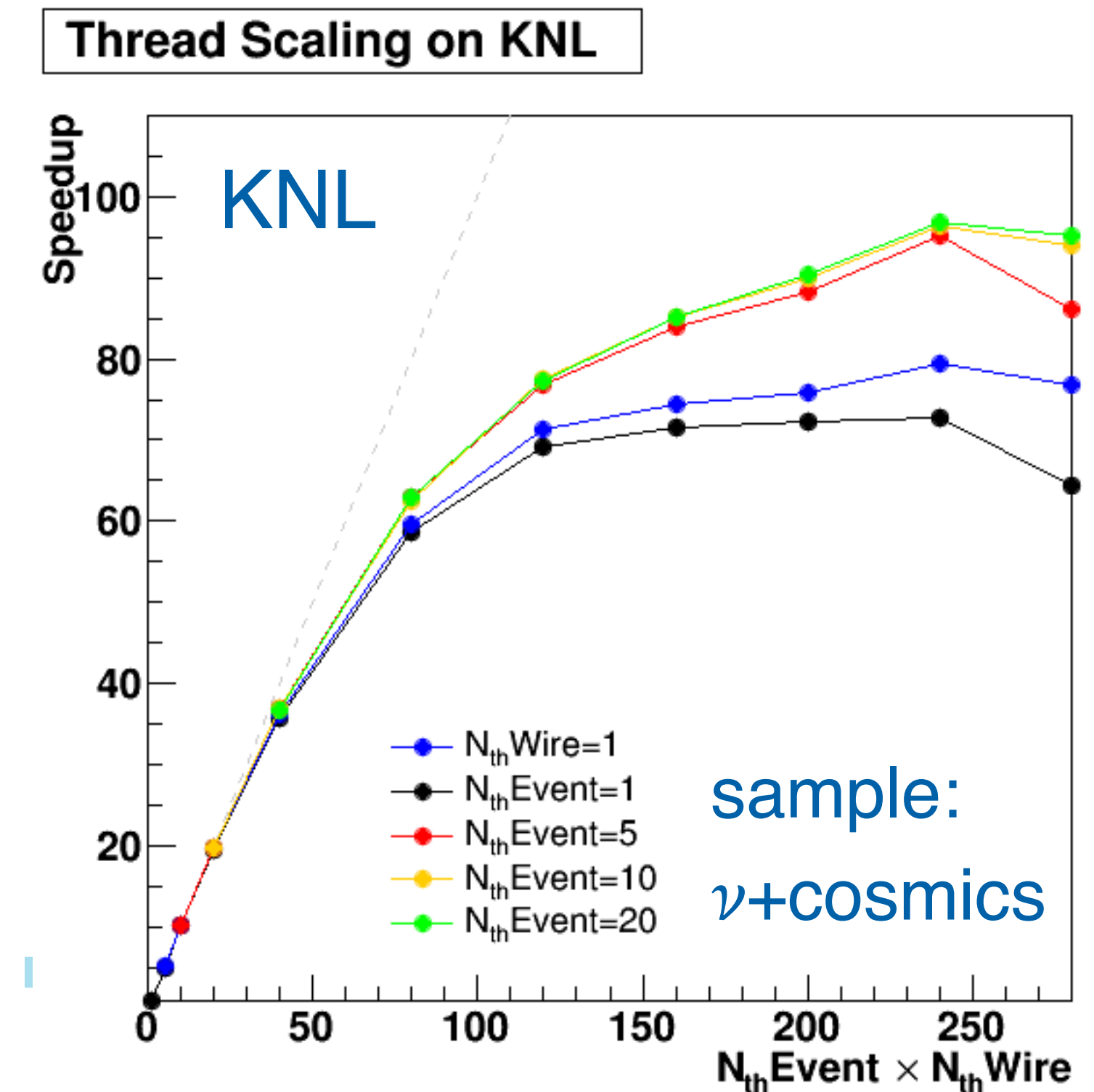
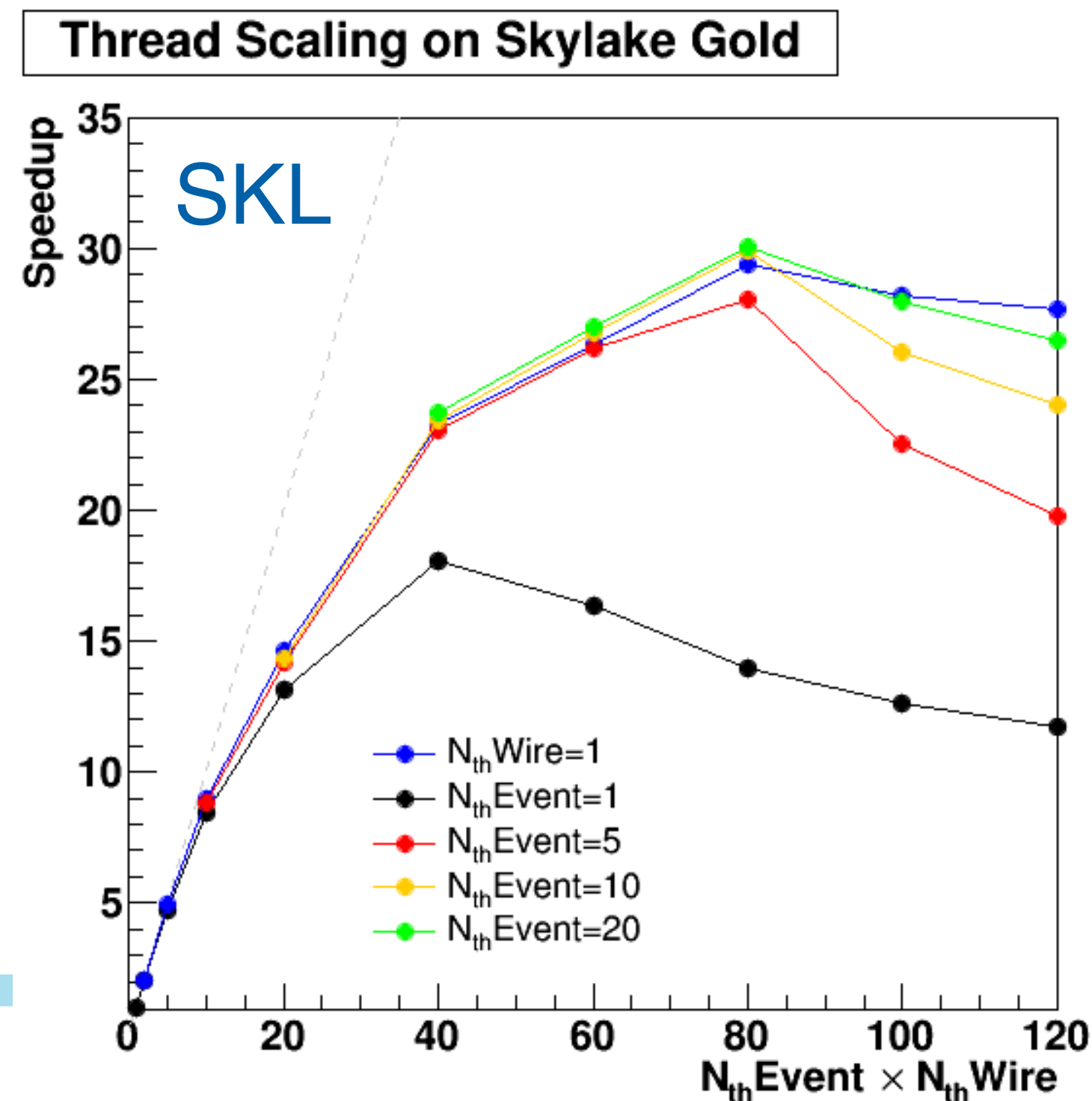
Standalone Hit Finder



Multi-threading Results

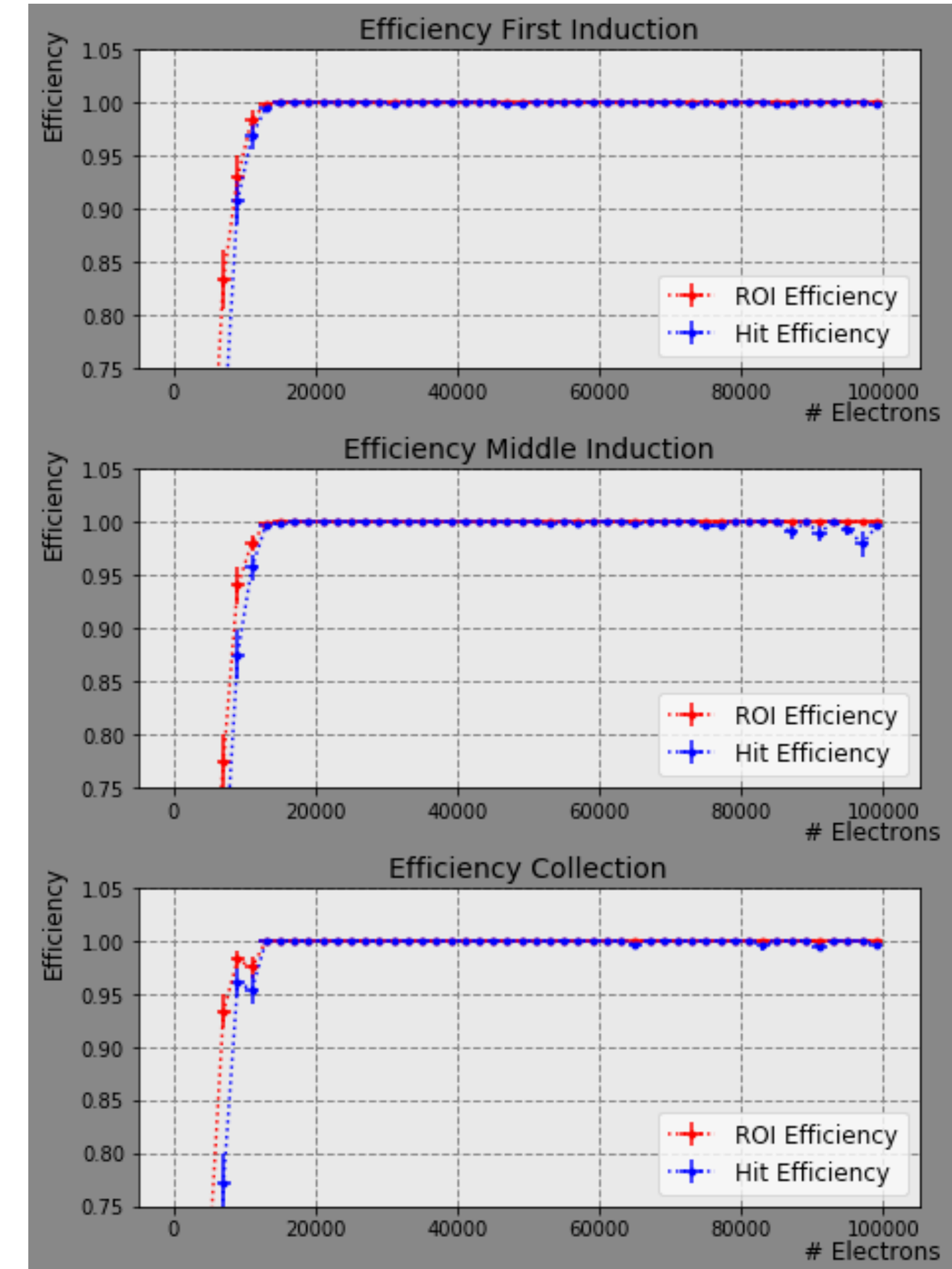
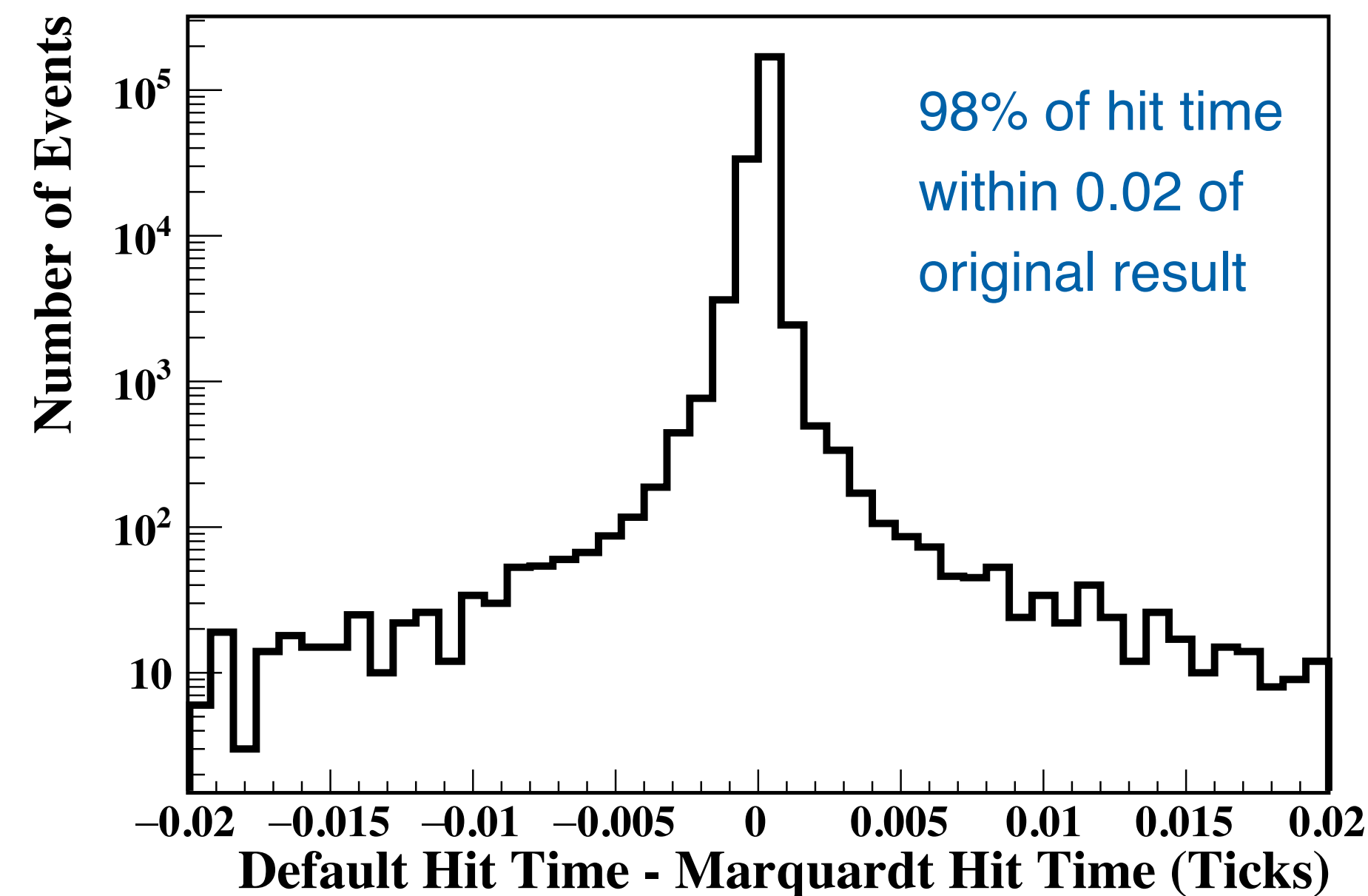
- In standalone version, implemented using **OpenMP** with dynamic scheduling
- Best performance achieved with **two-level nested parallelization**
 - *parallel for over events*
 - regions of interest on **wires**: *parallel region with omp for+critical* (output synchronization)

- Results show near ideal scaling at low thread counts
 - **speedup** increases up to **30x (95x)** for 80 (240) threads on Skylake Gold (KNL)



Validation of Algorithm Output

- Physics output **validated against original algorithm**
 - one to one comparison of hit parameters shows little difference
- Algorithm is **fully efficient** across all planes both in MicroBooNE and ICARUS
 - detectors with large differences in signal-to-noise ratio
 - waveforms with low S/N need fit parameters limits



LArSoft Integration

- Minimization algorithm **integrated** and used as a **plugin in LArSoft**
 - currently compiled with gcc by default
 - testing the Levenberg-Marquardt hit finder in MicroBooNE and ICARUS reconstruction shows speedups of 12x and 7x respectively (single thread)
- Multi-threading enabled in LArSoft within the hit finder module
 - implementation of wire+ROI level parallelization with TBB
 - rely on art for event-level multi-threading
- First vectorized and multi-threaded algorithm for LArTPC!

- Multi-threading of “1D” MC ICARUS signal processing sequence

Context: multithreading for production jobs

- art and larsoft provide multithreading capabilities through TBB library
 - art multithreading can process concurrently data across events or within the same event
- Grid allocations have total available memory split by CPU cores
- Grid jobs often need slots with large memory, thus getting multiple cores
- Production jobs are however running single-threaded, thus use only one core
- We can achieve significant processing speedups if we are able to exploit multithreading and increase our core utilization efficiency
 - multithreading within the event doesn't need to load more event data, can exploit unused cores given the same memory allocation
 - target for production jobs is to have efficient multithreading at moderate thread counts

Services and Multithreading

- Art does not allow to run multithreaded if services are not thread safe and consequently marked as “SHARED”
 - see [this talk](#) by Kyle for details
- Currently in ICARUS stage0_run2_icarus_mc.fcl the following services are loaded, and only the first two are LEGACY (not SHARED)
 - SIOVChannelStatusService, SIOVDetPedestalService, DetectorClocksServiceStandard, DetectorPropertiesServiceStandard, SignalShapingICARUSService, IcarusGeometryHelper, ICARUSChannelMap, LArPropertiesServiceStandard
- Scisoft team has been working on larsoft services with the goal of making them thread safe. Work is however taking significant time as changes are non-trivial and require to be propagated to downstream experiment code

However...

- Scisoft team is targeting thread safety both across and within events
- Since we only care about the latter, the situation is significantly simpler:
 - SIOVChannelStatusService and SIOVDetPedestalService access information from a DB
 - Thread safety within events only requires that the DB access is done at event boundaries or anyways only once per event
 - This can be enforced and we can make the service SHARED
 - using the “EnsureOnlyOneSchedule” functionality ([link](#) to class)
 - adding an std::mutex in the DBUpdate function
- Development merged in larevt in Nov. 2022

Multithreading implementation in modules

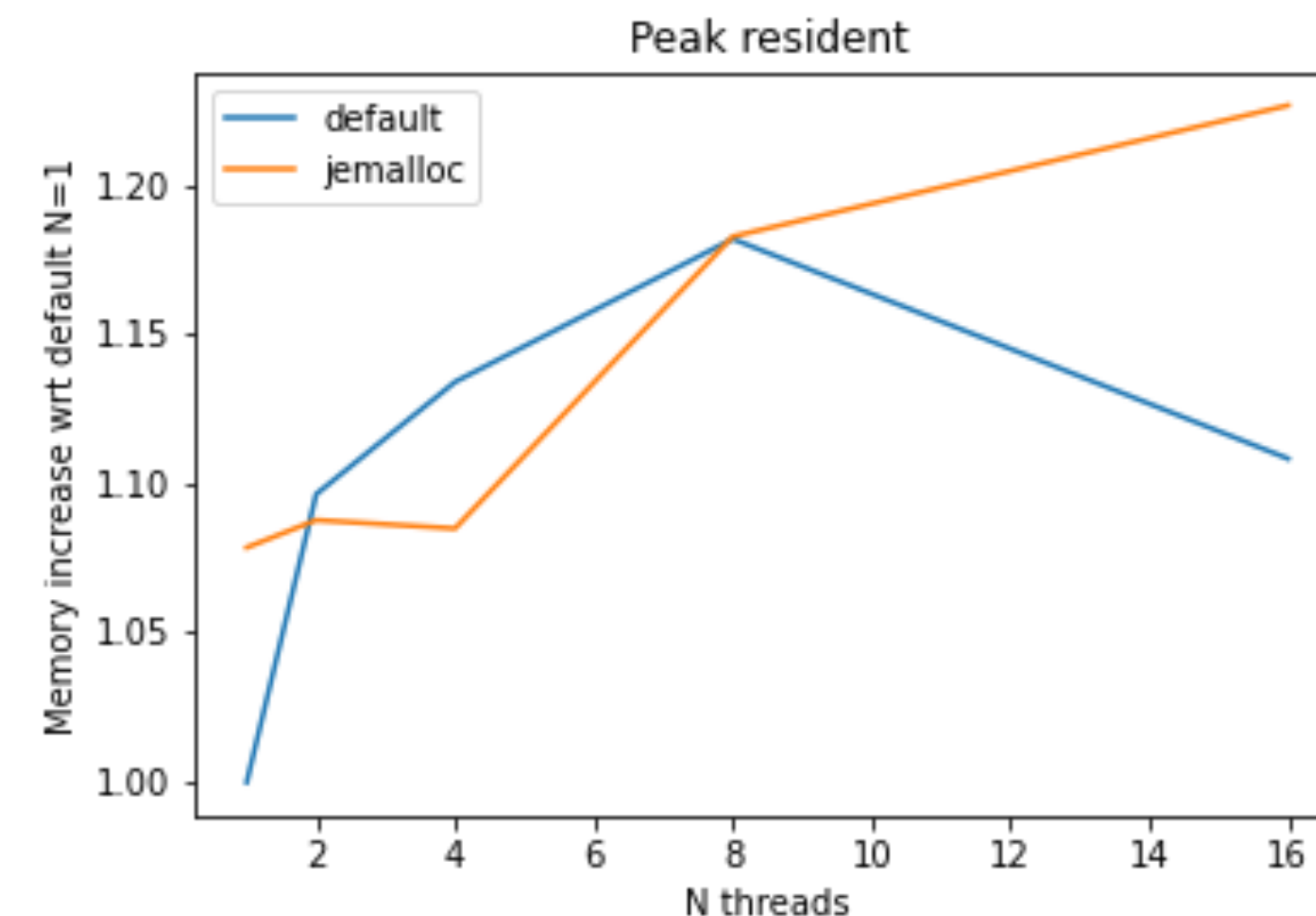
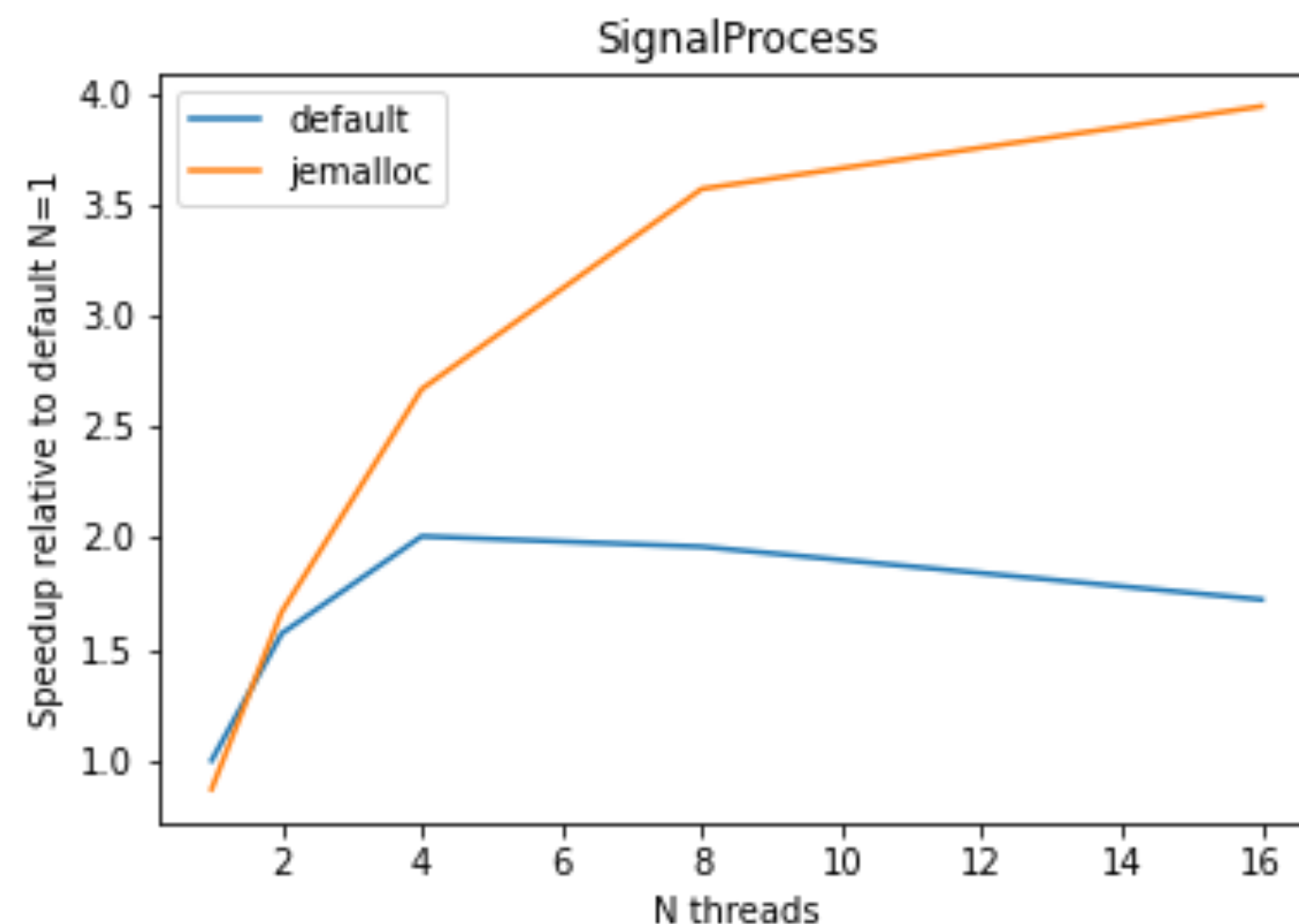
- Significant work on multiple fronts in the past years, time to harvest it
 - hit finder, icarus signal processing (thanks to Tracy Usher!), Wire Cell
- Icarus stage0_run2_icarus_mc.fcl basically has the following steps multithreaded with TBB already:
 - MCDecoderICARUSTPCwROI, Decon1DROI, ROIFinder, GausHitFinder
 - Although with some updates to Decoder and ROIFinder: PR to icaruscode is open
- Next I show results from this setup:
 - latest icaruscode release+PR, running stage0_run2_icarus_mc.fcl
 - tested without (default) and with jemalloc library for memory allocations
 - not meant to be a “final/optimized” version, goal is to demonstrate functionality
 - motivate usage of multi-threading for both 1D and 2D deconvolution processing

Scaling results

Out of the box, not necessarily optimized/tuned.

- Tested on icarusbuild02, without other ongoing jobs
 - not a production environment
- Can achieve up to 4x speedup for the 4 modules that are multithreaded
- Full stage0 processing speedup limited by other time consuming modules
 - but some of them may be low hanging fruits for speedups
- Memory increase is overall small, as expected

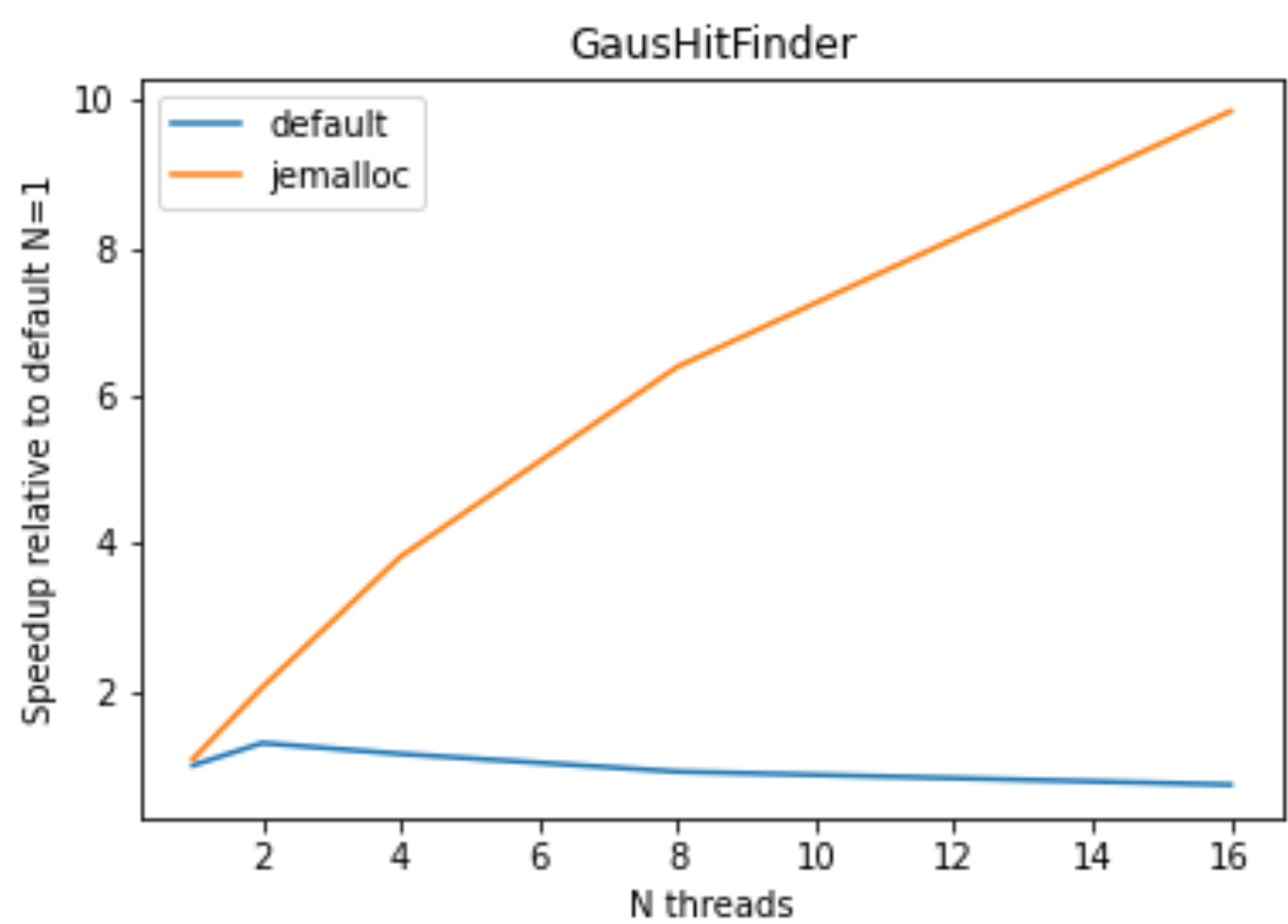
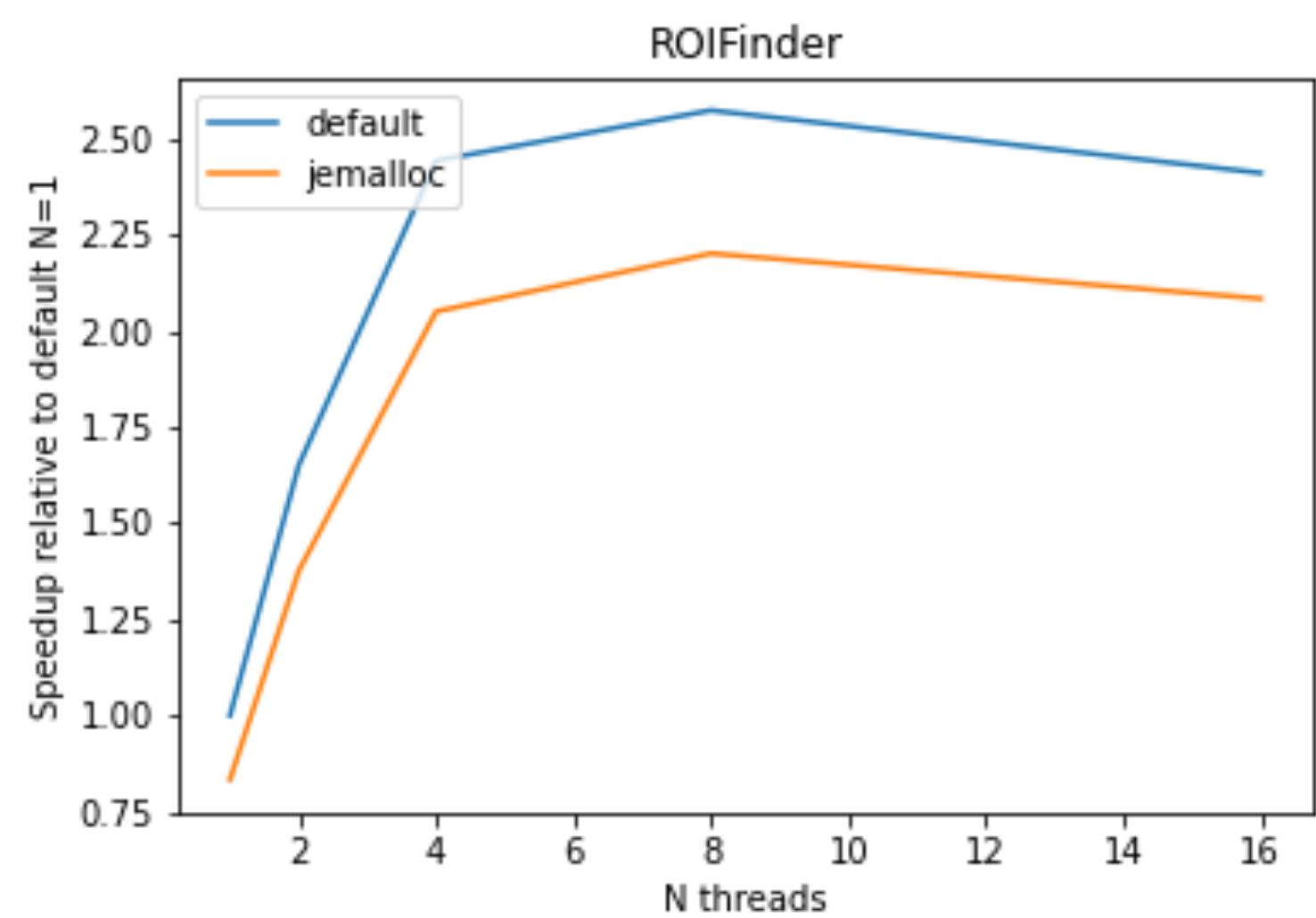
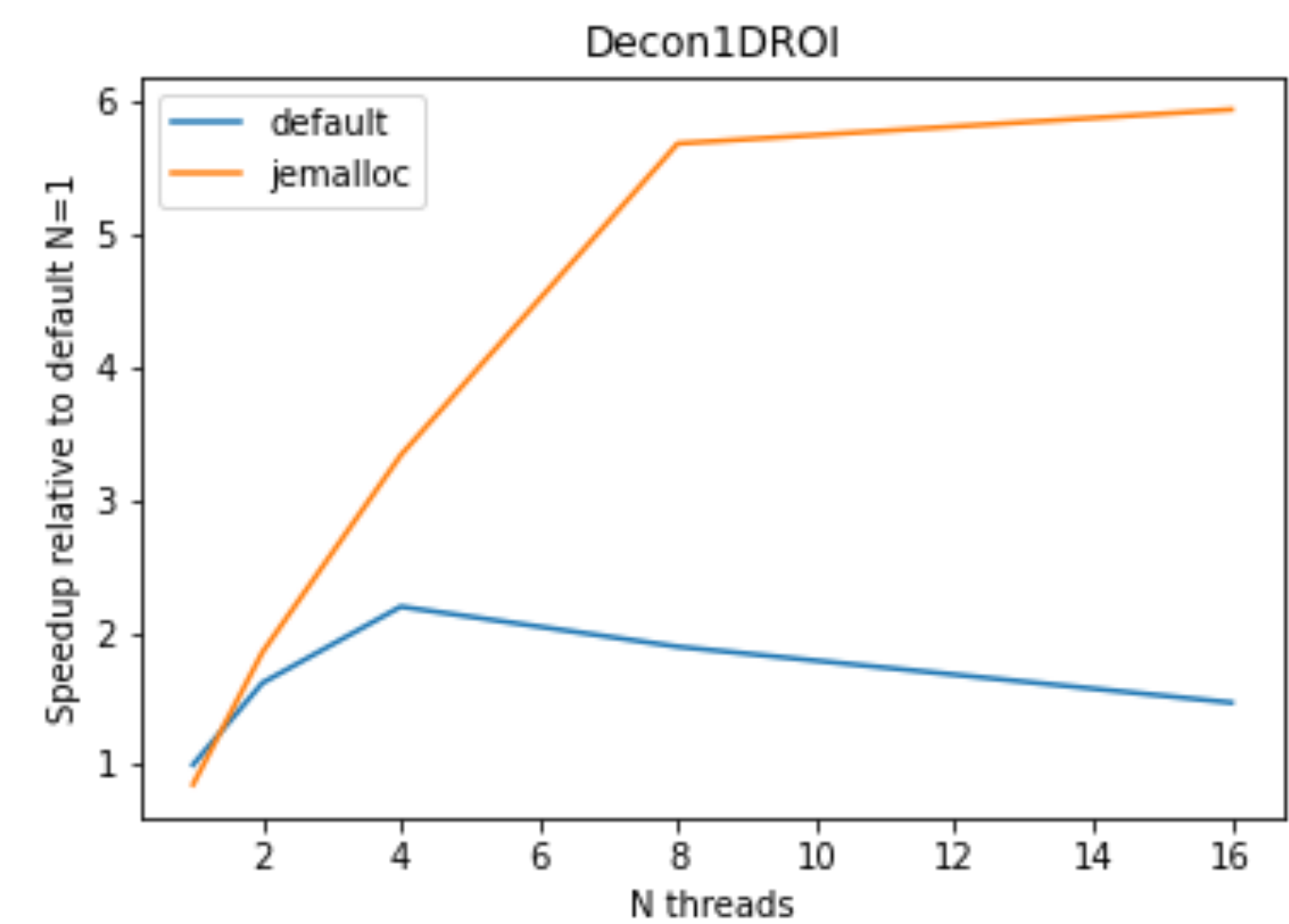
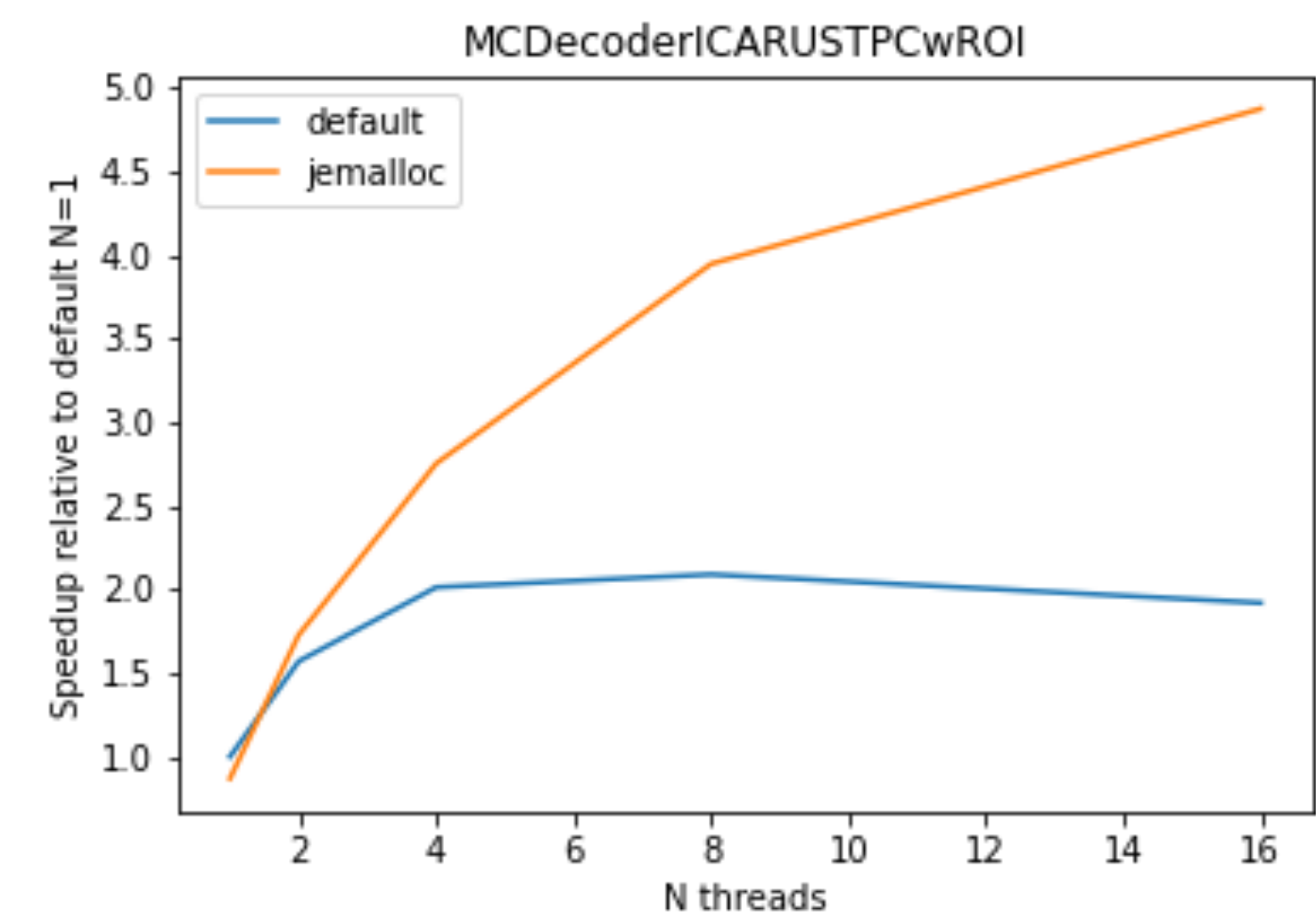
CPU time
speedup



memory usage
increase

Scaling of individual modules

Out of the box, not necessarily optimized/tuned.



- Running at HPC

LArTPC Reconstruction on HPC

- Work is ongoing to develop a **reconstruction workflow for HPC** centers.
 - Initial targets are **ICARUS** 1D signal processing and **Theta@ALCF**
- Goal is an **efficient** utilization of HPC resources
 - parallel architectures (SIMD and many-cores, also GPUs)
 - high bandwidth interconnects between nodes
- Workflow developed in collaboration with HEP-on-HPC SciDAC project
 - <https://computing.fnal.gov/hep-on-hpc/>
 - J. Kowalkowski, S. Sehrish, M. Paterno, S. Ali (FNAL), T. Peterka, O. Yildiz (ANL)



Spack builds

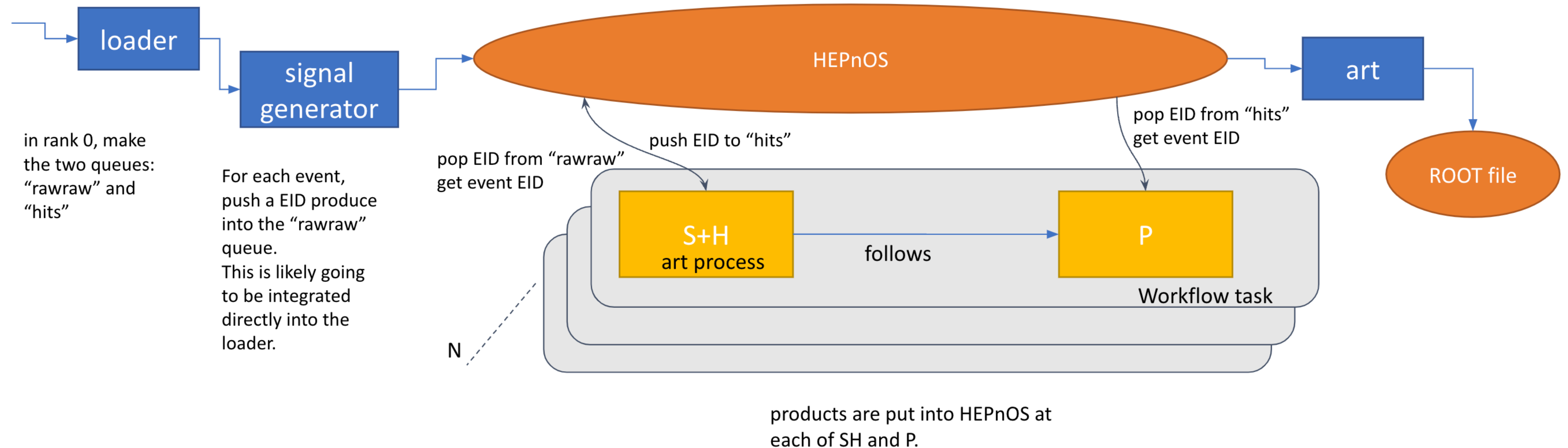


- LArSoft/art migrating away from home-brewed UPS-based tools for release builds
 - See e.g. talks at <https://indico.fnal.gov/event/51092/>, <https://indico.fnal.gov/event/51726/>
- Targeting modern HPC-friendly tools such as Spack (see spack.readthedocs.io)
- Spack provides a simple way to customize compilation at package level
 - icc and AVX-512 are needed for optimal vectorization speedups in hit finder
- A new package, larvecutls, was created containing vectorized code
 - right now only MarqFitAlg, but more can be added in the future
- Migration work still ongoing, but building icaruscode releases with Spack is possible!
 - still requires manual work to produce a recipe, currently using icaruscode v09_37_01_02p04
 - recent releases after cetbuildtools migration, so it may not be difficult to propagate recipe
 - thank you to FNAL Spack team: P. Gartung, C. Green, M. Mengel, S. White

HEPnOS

- HEPnOS is a distributed data service for managing HEP data.
 - distributed: available to all nodes on a machine, through memory (not reading files)
 - data service: independent of user applications; works with domain concepts (datasets, runs) not artifacts (files)
- Features:
 - Accelerates access by retaining data in the system (in memory) throughout analysis process.
 - Uses SciDAC Institute technologies to get optimal use of interconnects at ASCR facilities
 - Provides for large scale, run-time configurable, parallelism
 - global view of data, removes limitations from filesystem
 - Supports workflow load balancing across a large machine
- For this workflow what we did is:
 - We built a consistent software stack: both ICARUS code and HEPnOS using same compiler and flags
 - Implemented the ability to store and load the required data types in HEPnOS
 - Developed HEPnOS art input source and output module
 - IO capabilities limited to selected data products, not full metadata

Workflow layout



- Running signal processing (S), hit finding (H), and cluster3D+Pandora (P) reconstruction
 - S in multi-threaded, H is vectorized and multi-threaded, P is serial
- MPI-wrapper allows to execute an art/LArSoft instance in each rank, running S, H then P
- HEPnOS servers communicate with art/LArSoft via MPI
 - Each server supports as many ranks as allowed by memory available on node
- Ongoing tests on Theta with different ranks per node and different threads per rank. Stay tuned!

Conclusions

- Developed first vectorized and multi-threaded algorithm in LArSoft: hit finder
- Extended usage of multi-threading to other modules of 1D ICARUS signal processing
- Tests show that usage of jemalloc is critical for good thread scaling
- Ongoing work to prototype a LArSoft-based workflow for efficient usage of HPC resources - ongoing tests on Theta