SciDAC4 developments

Giuseppe Cerati (FNAL)

FNAL Frameworks Workshop

June 5, 2023

Project Goals

SciDAC
Scientific Discovery through Advanced Computing

- "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**

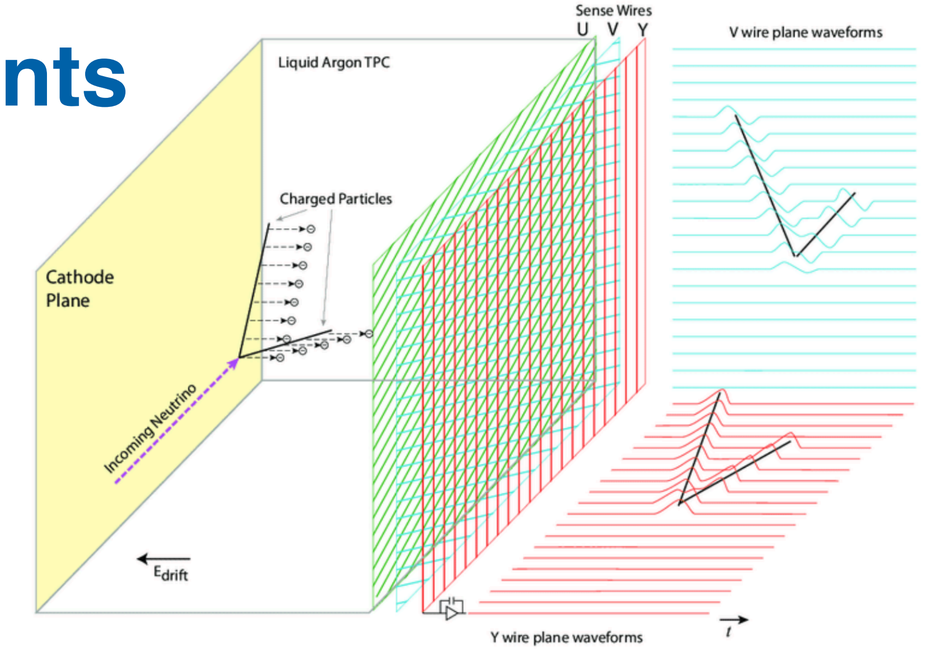
- Focus on two areas:
  - Novel parallel algorithm for charged particle **tracking in CMS**
  - Pioneer similar techniques for **reconstruction in LArTPC detectors**

- 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

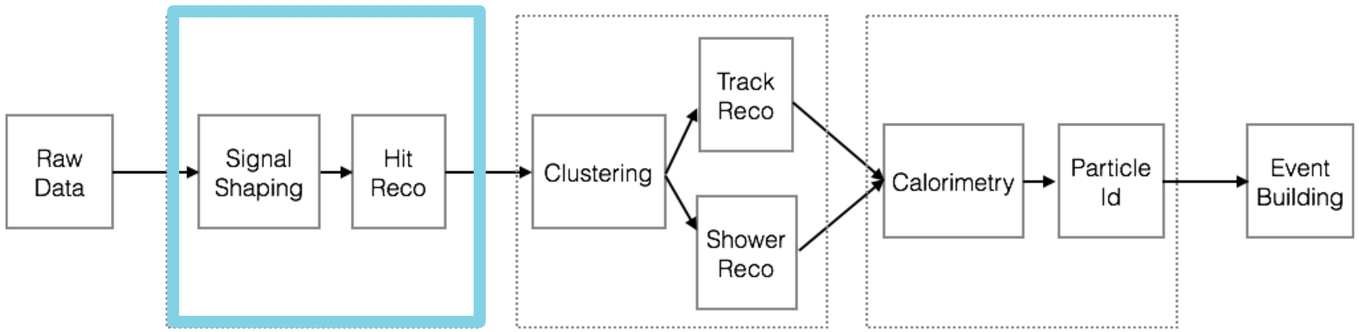
Fermilab

Reconstruction for LArTPC ν experiments

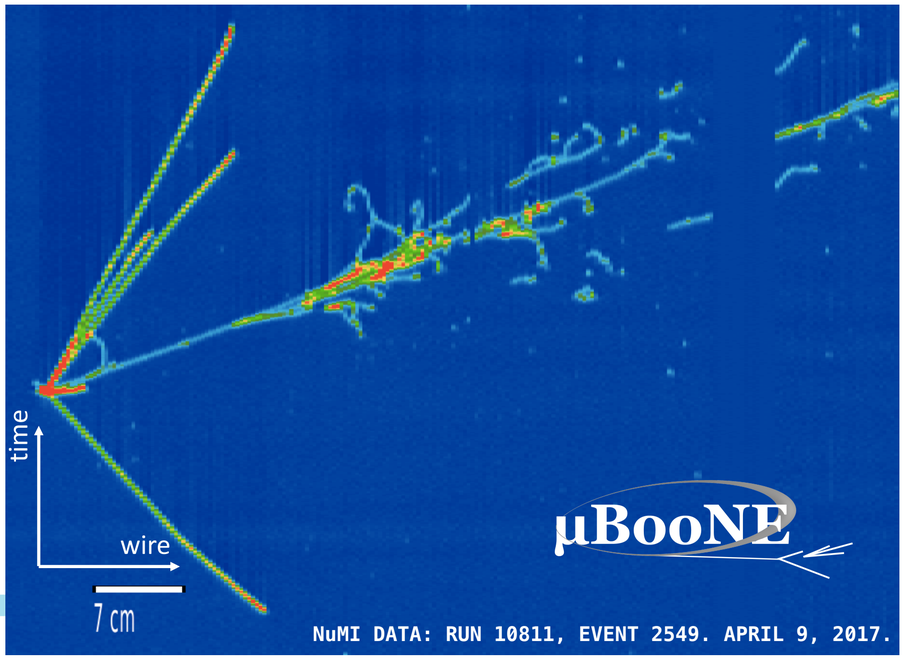
- Charged particles produced in neutrino interactions ionize the argon, ionization electrons drift in electric field towards anode planes
- Sense wires detect the incoming charge, producing beautiful detector data images



- 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 ~5x bigger, DUNE Far Detector O(100)x bigger
- LArTPC detectors are modular in nature ➔ **parallelism**!



Typical reconstruction chain for LArTPC experiments



- Parallelization of Hit Finder Algorithm

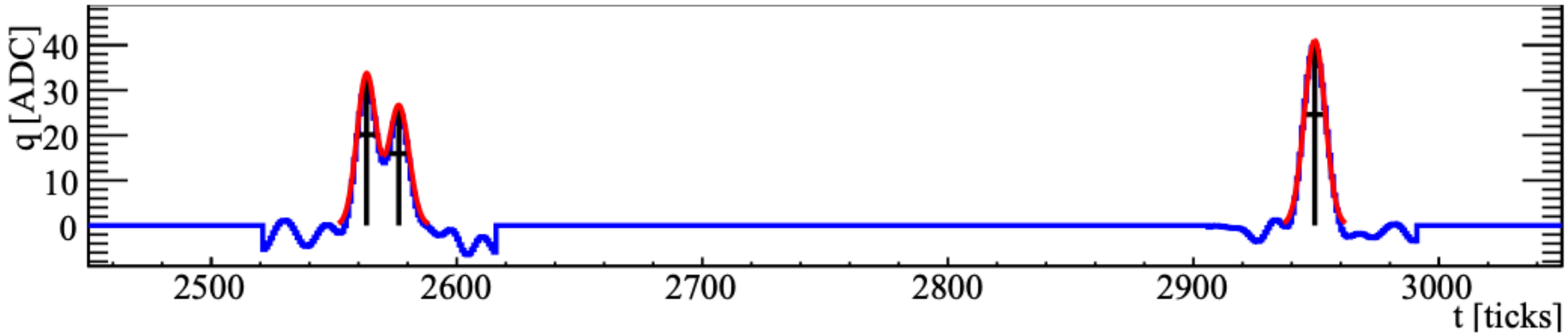
Initial Study: Hit Finding

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 [physics.ins-det]

- MicroBooNE: ~8k wires readout at 2 MHz, deconvolved wire signals are **Gaussian pulses**
- Hit finding: **identify** pulses and determine their **peak** position and **width**
- It ~~can~~ 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



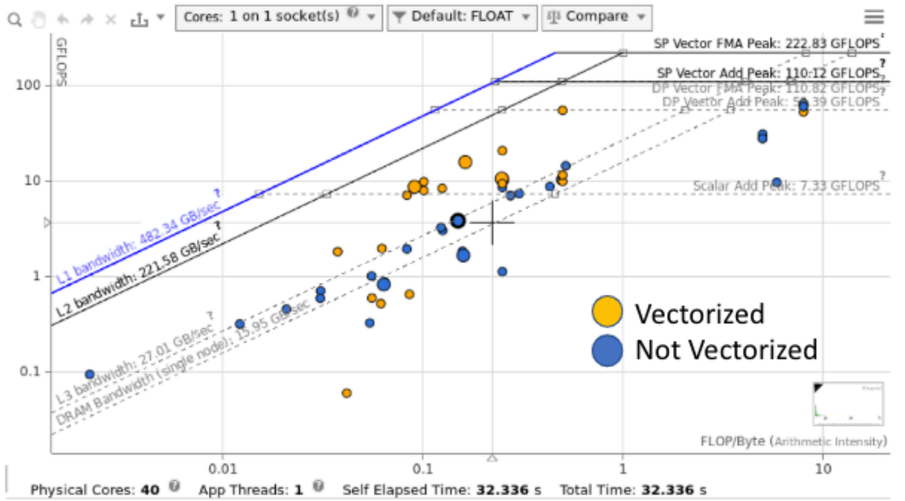
Fermilab

Standalone Implementation

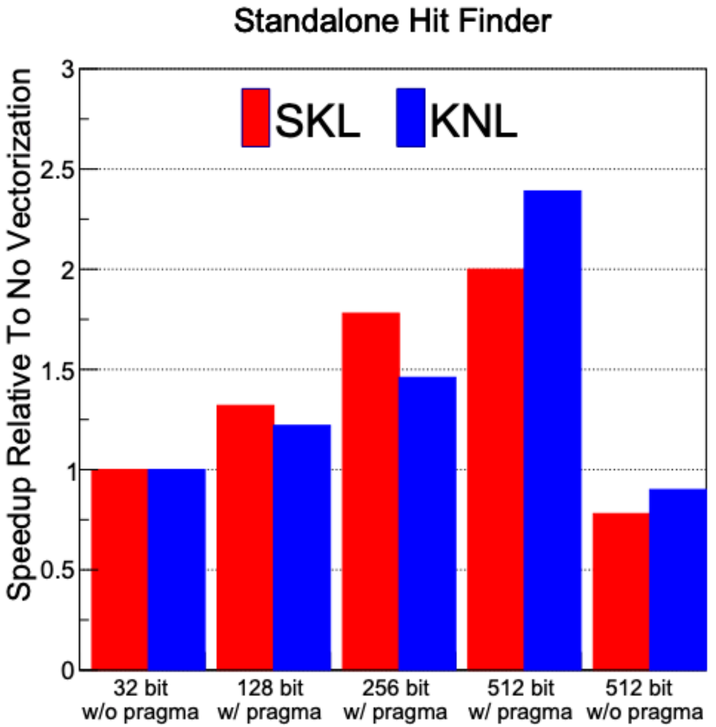
- Replicated LArSoft hit finder as **standalone code** for testing and optimization
- Replaced Gaussian fit based on Minuit+ROOT with a local implementation of **Levenberg-Marquardt minimization**
  - gradient descent when far from minimum and Hessian minimization when close to it
    - implementation 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

Roofline analysis



- 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

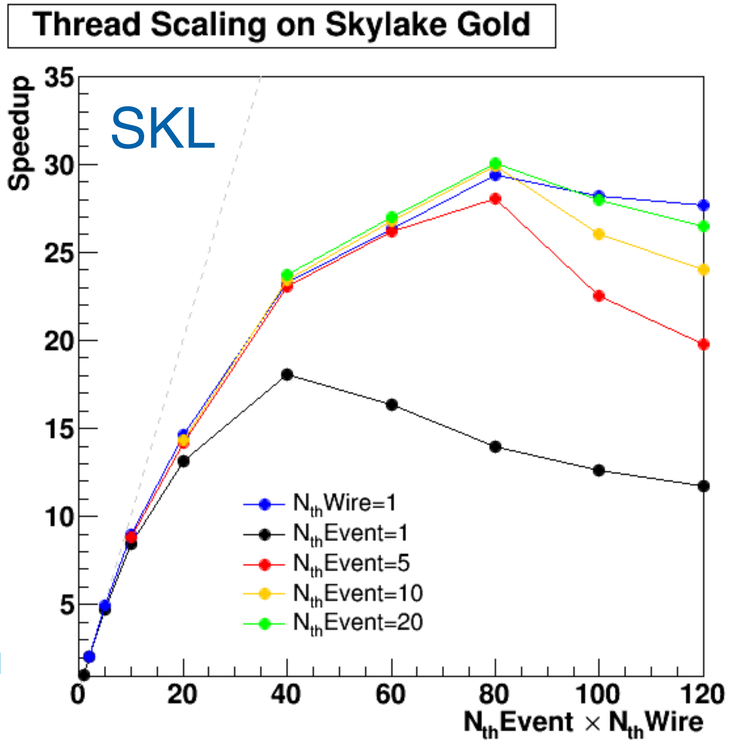


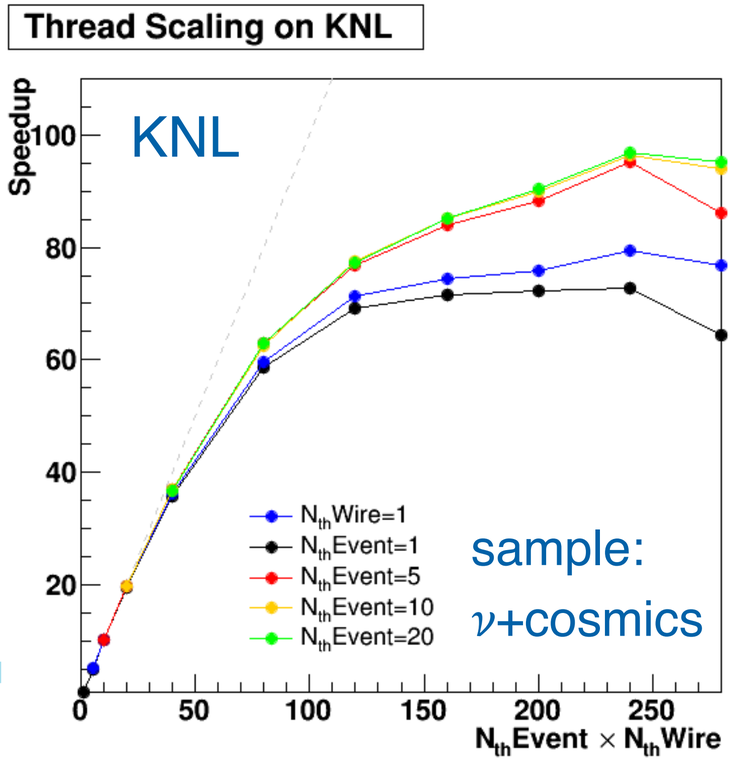
sample: ν+cosmics Vectorization performance

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)





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

Multithreading implementation in modules

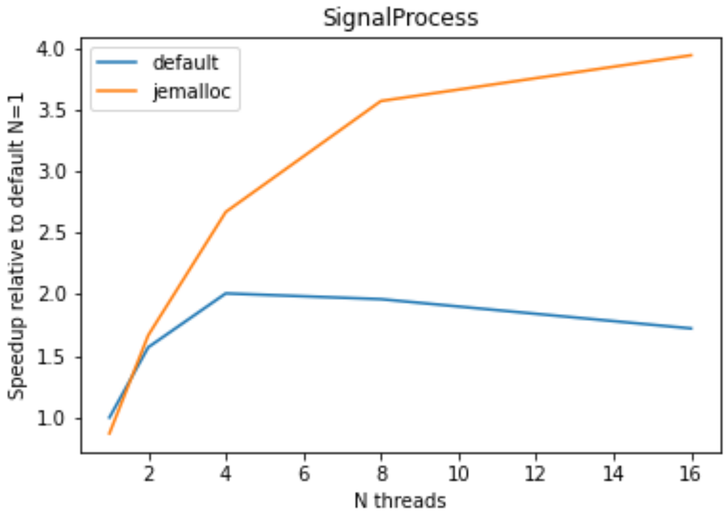
- Signal processing modules in ICARUS are multithreaded:
  - MCDecoderICARUSTPCwROI, Decon1DROI, ROIFinder, GausHitFinder
  - Credit to T. Usher for most of the work on the first 3 modules!
  - We focused on making services thread safe within event boundaries and to finalize the implementation of the the above modules
- Results in the next slides show scaling results obtained during development
  - tested without (default) and with jemalloc library for memory allocations
  - not meant to be a “final/optimized” version, goal is to demonstrate functionality
- These modules now run multithreaded in production workflows on the grid

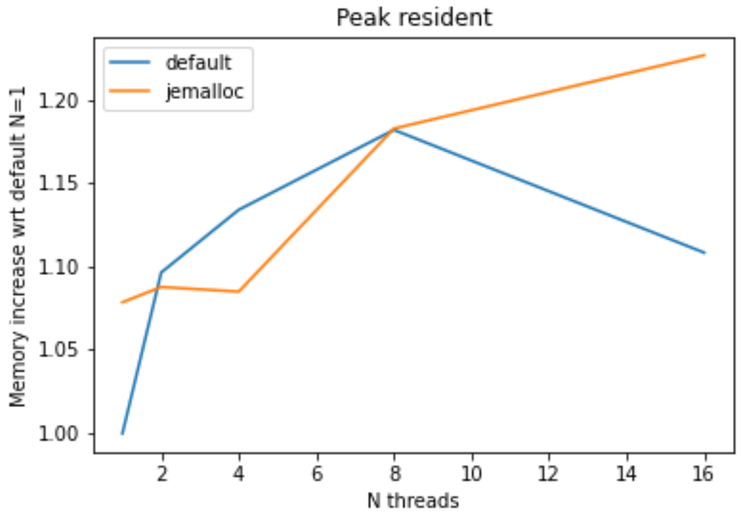
Scaling results

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



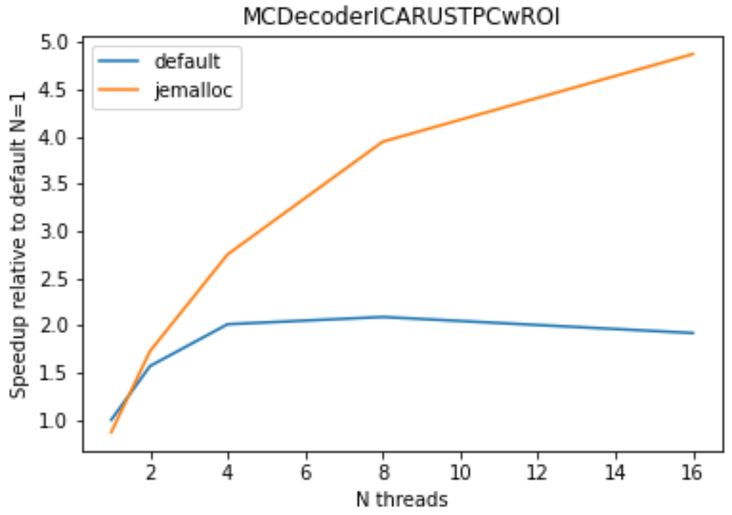


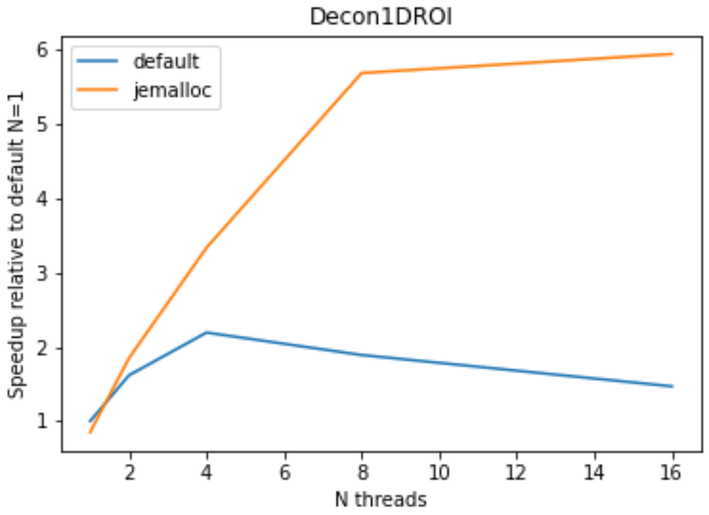
memory usage increase

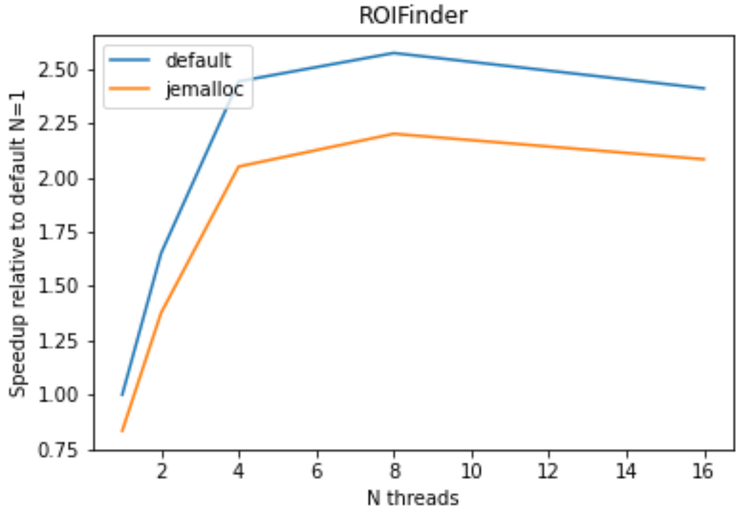
Fermilab

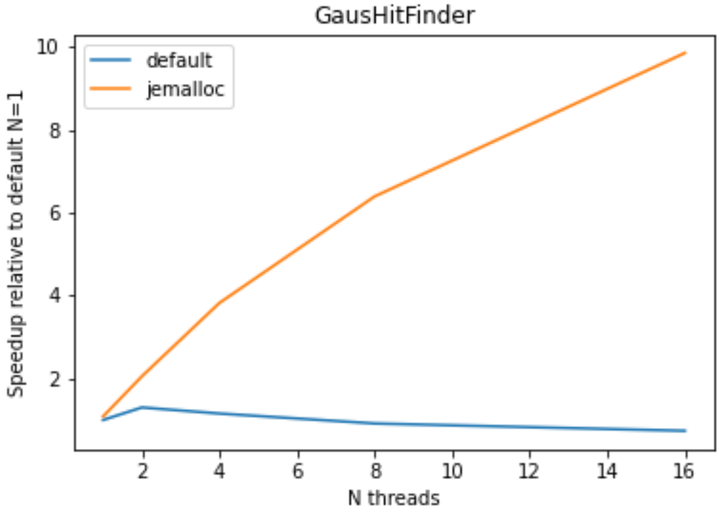
Scaling of individual modules

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, larvecutils, 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

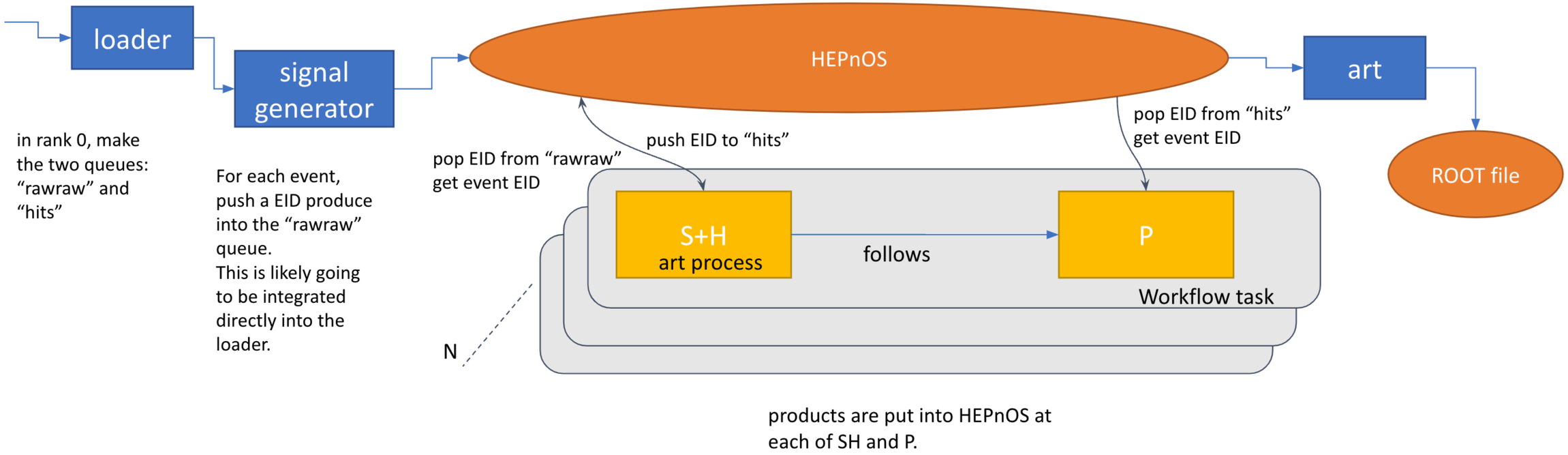
HEPnOS

- 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
  - HEPnOS internally uses argobots threads: had to avoid conflicts with TBB by ensuring all argobots calls are from the same OS thread

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!

HEPnOS Preliminary Results

Fermilab

Background: File-based workflows for grid-computing

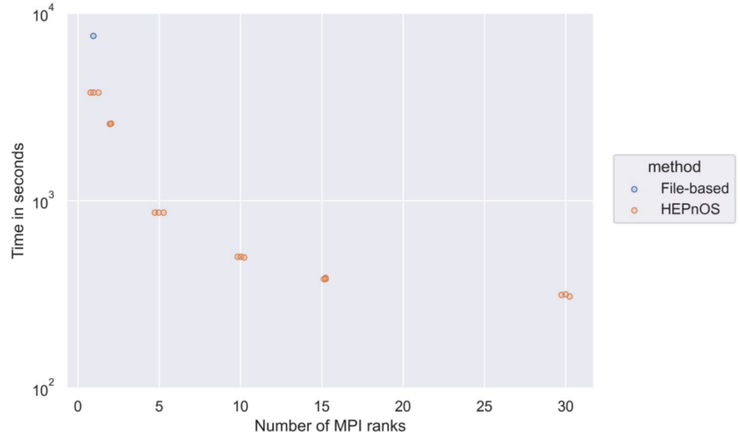
- Current workflow uses a **file-based workflow** suited for **grid-computing:**
  - Input is a set of files, each containing ~30 events, ~10-20 Gb (for ICARUS).
  - Run each task (set of analysis modules) in the analysis pipeline as a **process.**
  - Size of a task is limited by available time, memory, disk space for output files.
  - Use **files to pass the results** between tasks.
- ICARUS uses the *art* framework to analyze events. Each *art* instance can concurrently process multiple events (each harnessing multiple-threads).
- Typically, one *art* process runs on a node and analyzes all the events in a file, producing one output file containing the **data products produced by the analysis and ones produced by preceding analysis**.
- Thus, we **typically have fewer *art* processes than files!**

[Sajid Ali @ CHEP23]

Fermilab

Preliminary evaluation

- Used 1 KNL node on Theta, with 64 (x86_64) cores for the analysis and 1 KNL node as the server for the *HEPnOS* workflow.
- Used MPI to launch multiple *art* processes.
- We observe a scaling efficiency of 75% at 10 MPI ranks and 66% at 15 MPI ranks.



Bonus Topic: Code Portability

M. Kwok @ CHEP23

Portability: Software landscape

HEP-CCE

- Rapidly changing ~O(month) portability solutions
  - New features/compiler supports/New backend
- Different approaches:
  - Compiler pragma-based approach
  - Libraries
  - Language extension
- HEP-CCE: Joint effort of major U.S. National labs involved in HEP
  - Investigate different portability solutions in HEP context

Software →

Hardware ↓

	CUDA	Kokkos	SYCL	HIP	OpenMP	alpaka	std::par
NVIDIA GPU			intel/llvm compute-cpp	hipcc	nvc++ LLVM, Cray GCC, XL		nvc++
AMD GPU			openSYCL intel/llvm	hipcc	AOMP LLVM Cray		
Intel GPU			oneAPI intel/llvm	CHIP-SPV: early prototype	Intel OneAPI compiler	prototype	oneapi::dpl
x86 CPU			oneAPI intel/llvm computecpp	via HIP-CPU Runtime	nvc++ LLVM, CCE, GCC, XL		
FPGA				via Xilinx Runtime	prototype compilers (OpenArc, Intel, etc.)	prototype via SYCL	

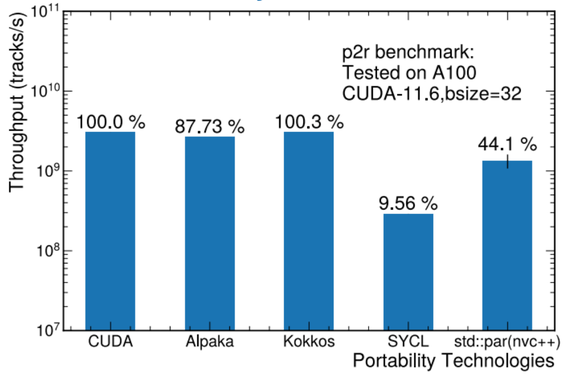
Stay tuned tomorrow for! HEP-CCE result

Fermilab

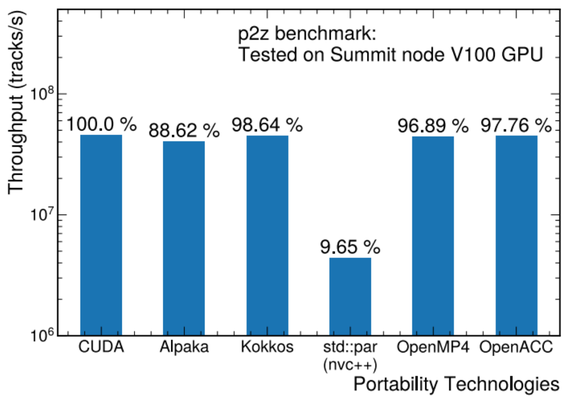
GPU Results - NVIDIA

- p2r's measurement more sensitive changes to kernel execution
  - p2z measurement is sensitive to overheads related to data movement
- Kokkos and Alpaka both managed to produce close-to-native performance
- Unclear what is causing the slowdown in SYCL/std::par in p2r versions
  - Profiling shows significant branching in SYCL version

p2r: NVIDIA GPU (A100)
Kernel-only



p2z: NVIDIA GPU (V100)
Data movement + kernel



Fermilab

Several options on the table, with similar performance (although application-dependent).

Early adoption of one approach can be beneficial for a framework, but would be good to be able to easily change as tools mature.

Fermilab

Conclusions

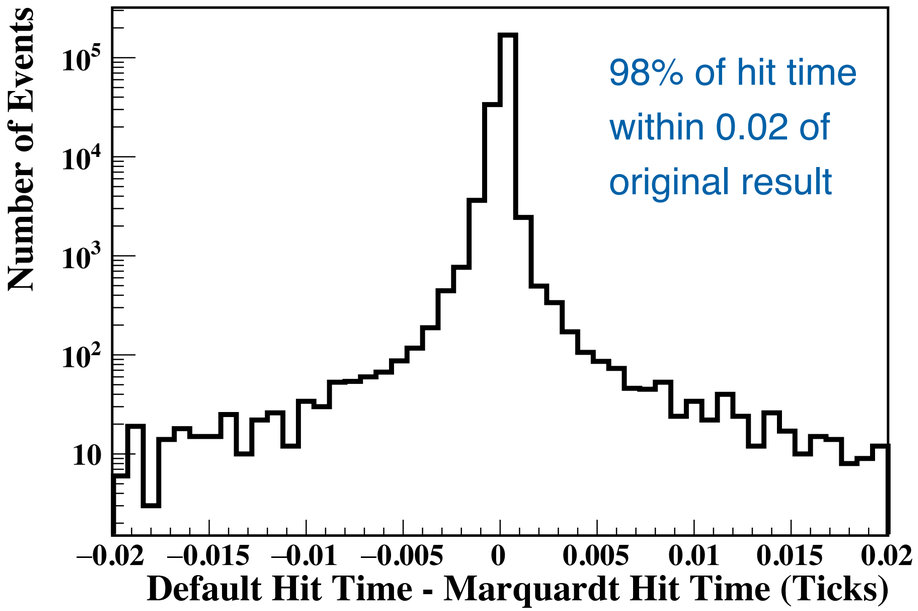
- SciDAC4 projects have been on the forefront of developments in various areas such as:
  - Vectorization and Multithreading
  - HPC workflows
  - Code portability tools

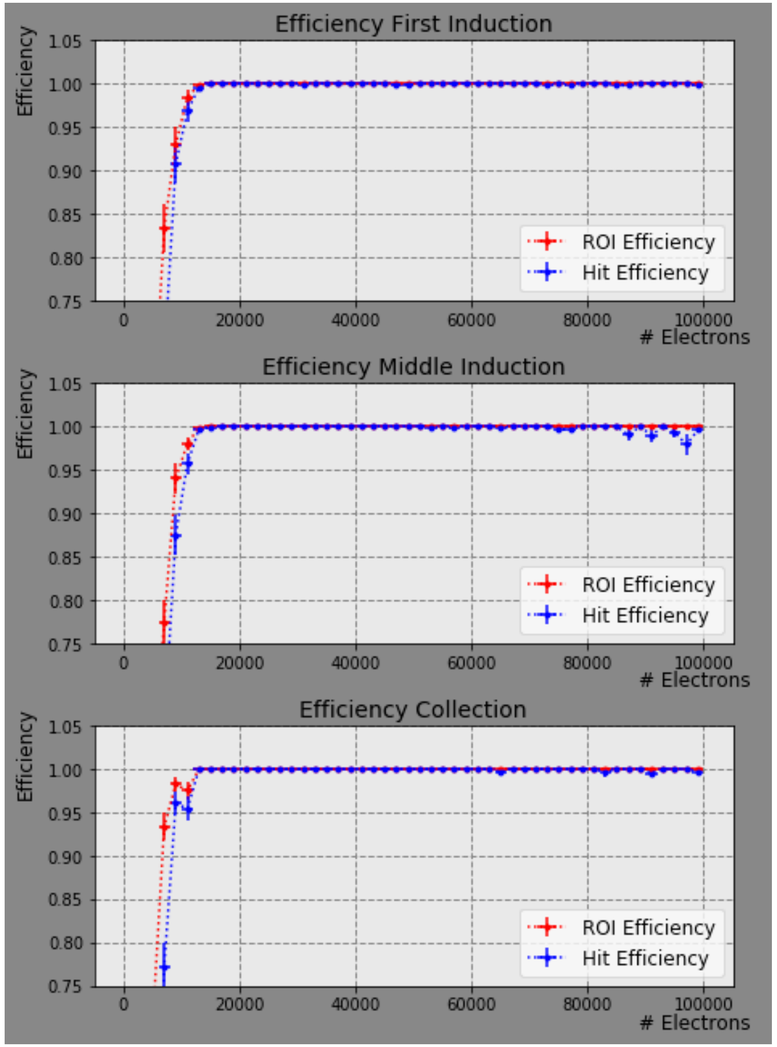
- Take home messages for framework developments are:
  - Optimal vectorization may require specific compilation options
  - Memory-intensive workflows best exploit multithreading solutions not requiring more data
    - This typically means parallelization within the event. Also, jemalloc help MT performance.
  - Object stores can improve scaling at HPC by removing file boundary restrictions
  - Code portability solutions are becoming mature and need to be supported

Backup

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





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