Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY

Multi-Threading and Acceleration in Wire-Cell Toolkit

Haiwang Yu (BNL) for the Wire-Cell Team

2023-03-03

LArSoft Multi-threading and Acceleration Workshop

Wire-Cell

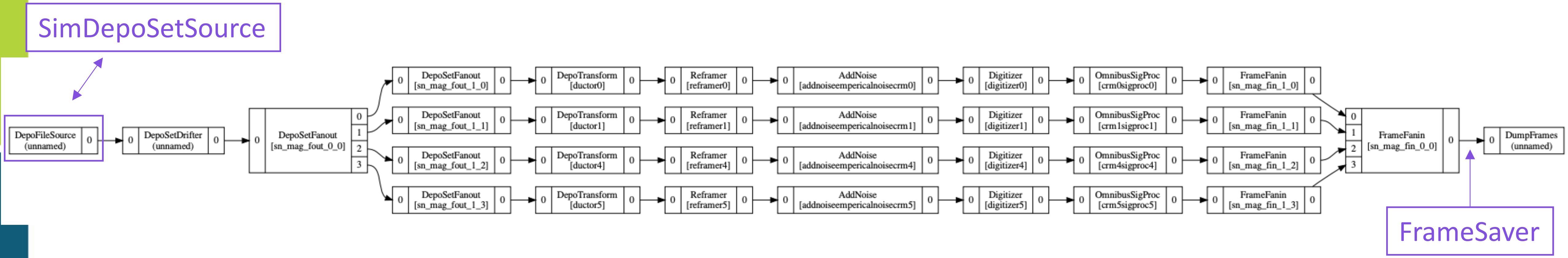
Multi-Threading and Acceleration in Wire-Cell Toolkit

Goal: to balance computing resources

Techniques in Wire-Cell Toolkit (WCT)
- Task level multi-threading – CPU/Memory
  - targeting Sim+SigProc
  - real benefit?
  - research → production
- Portable parallelization – if we have GPU
- PyTorch – ML/GPU
- ZIO: zero-MQ based distributed computing framework – between nodes
- IDFT – targeting one alg.

Brookhaven National Laboratory

wire-cell-toolkit and larwirecell



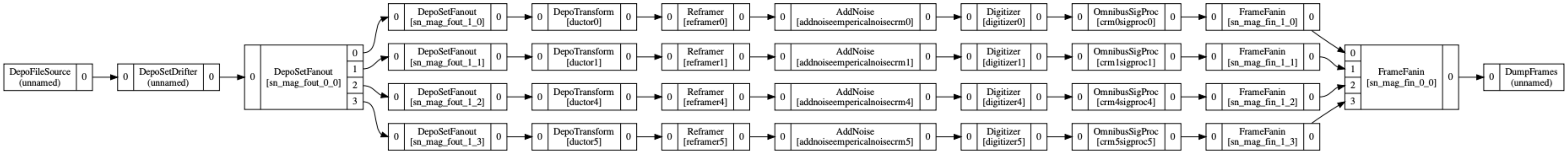
wire-cell-toolkit (wct): https://github.com/WireCell/wire-cell-toolkit
configurations:

- centric: cfg in the wct repo
- experiment: e.g., https://github.com/DUNE/dunereco/tree/develop/dunereco/DUNEWireCell

larwirecell: https://github.com/LArSoft/larwirecell

- Links LArSoft and wct
- If run the example above in LArSoft:
  - DepoFileSource → SimDepoSetSource:IArtEventVisitor
  - add FrameSaver:public IArtEventVisitor after 'FrameFanin'

Use TBB engine for WCT jobs



in jsonnet
wire-cell -t 4 -c wct.jsonnet ...

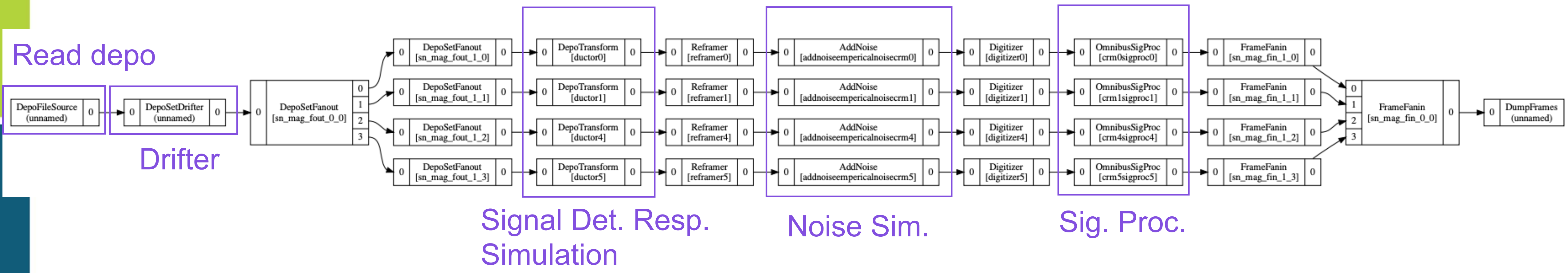
```
local graph = g.pipeline([wcls_input.deposet, setdrifter, bi_manifold, retagger, wcls_output.sp_signals, sink]);

local app = {
    type: 'TbbFlow', //Pgrapher, TbbFlow
    data: {
        edges: g.edges(graph),
    },
};
```

in fhicl
lar -j 4 -c wcls.fcl ...
no Legacy services

```
tpcrawdecoder : {
  module_type : WireCellToolkit
  wcls_main: {
    tool_type: WCLS
    # apps: ["Pgrapher"]
    apps: ["TbbFlow"]
```

Brookhaven National Laboratory

Task for benchmarking



Processed CRUs

3	7	11	15	19	23
3					
2					
2					
1	5				
1	5				
0:0	4:0	8	12	16	20
0:1	4:1				

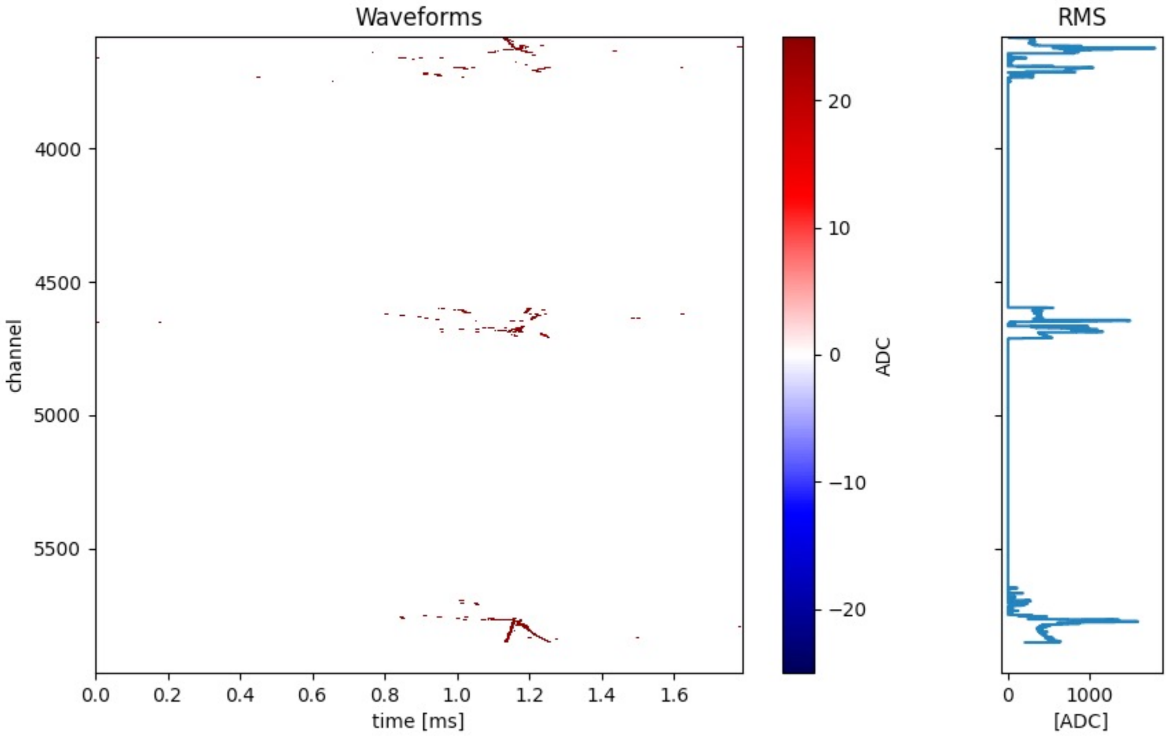
…

Simulation + Signal Processing for DUNE-FD2-VD
- input: energy depos from genie + G4 using LArSoft v09_65_02d00
- both Sim and SigProc handles **dense data**
- and are **major time-consumers in many production campaigns**
- no communications needed cross CRUs/APAs

Sample SigProc result

Only 1 CRU in the 0th event got a neutrino interaction

Most of the time the workflow simulates noise and performs SigProc



lar vs. wct

Tested on dunegpvm09 with 4 cores and 10GB memory? No other active users at testing time.

- my local machine is not setup to run the latest LArSoft currently

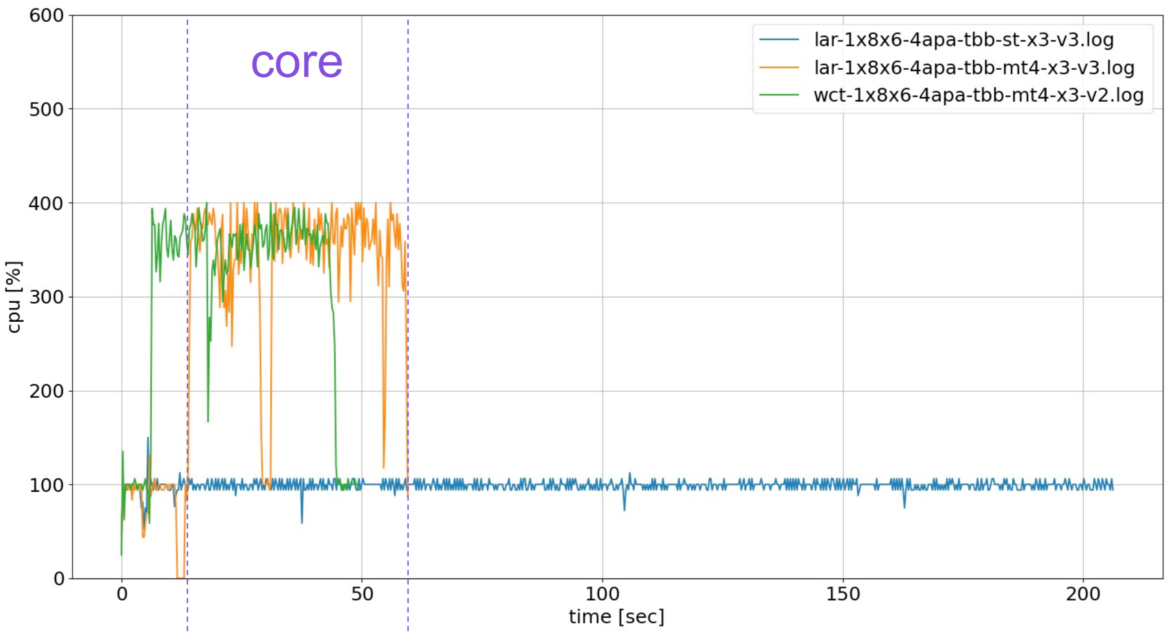
To demonstrate WCT MT can be used in production

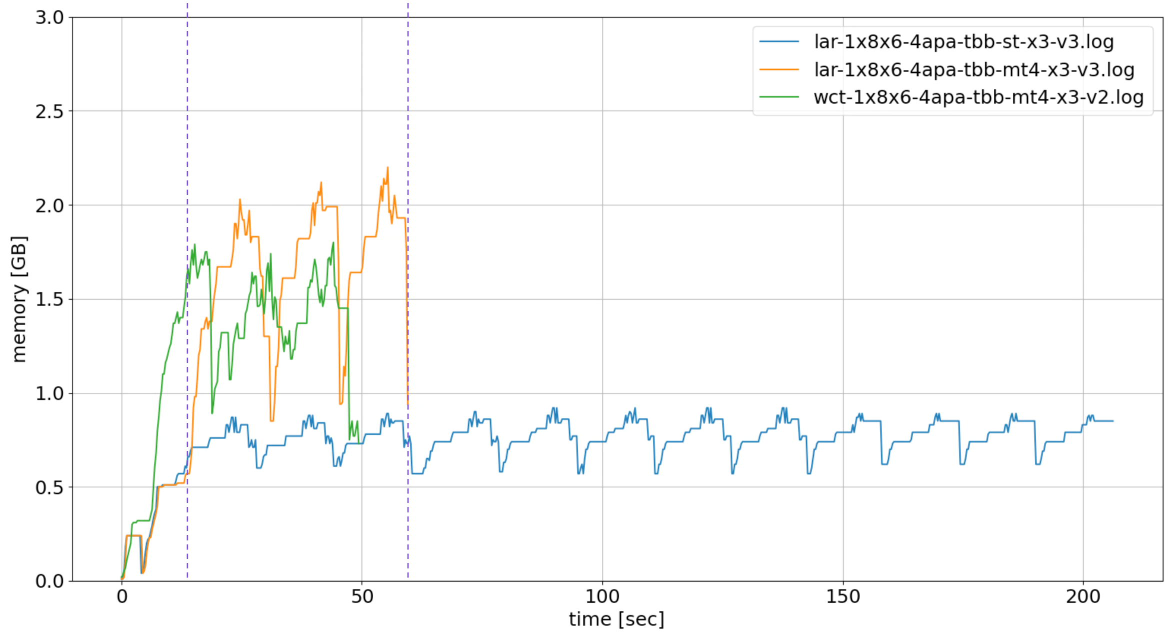
CPU:

- speed up for core: 4.1 (lar, fluctuation?)
- seems lar has a bit more overhead and is a bit slower but hard to say with fluctuations (similar results running twice)

Memory:

- MT uses more and is peakier.
- wct uses a bit less memory
- 0.92 → 0.54 GB (per core)



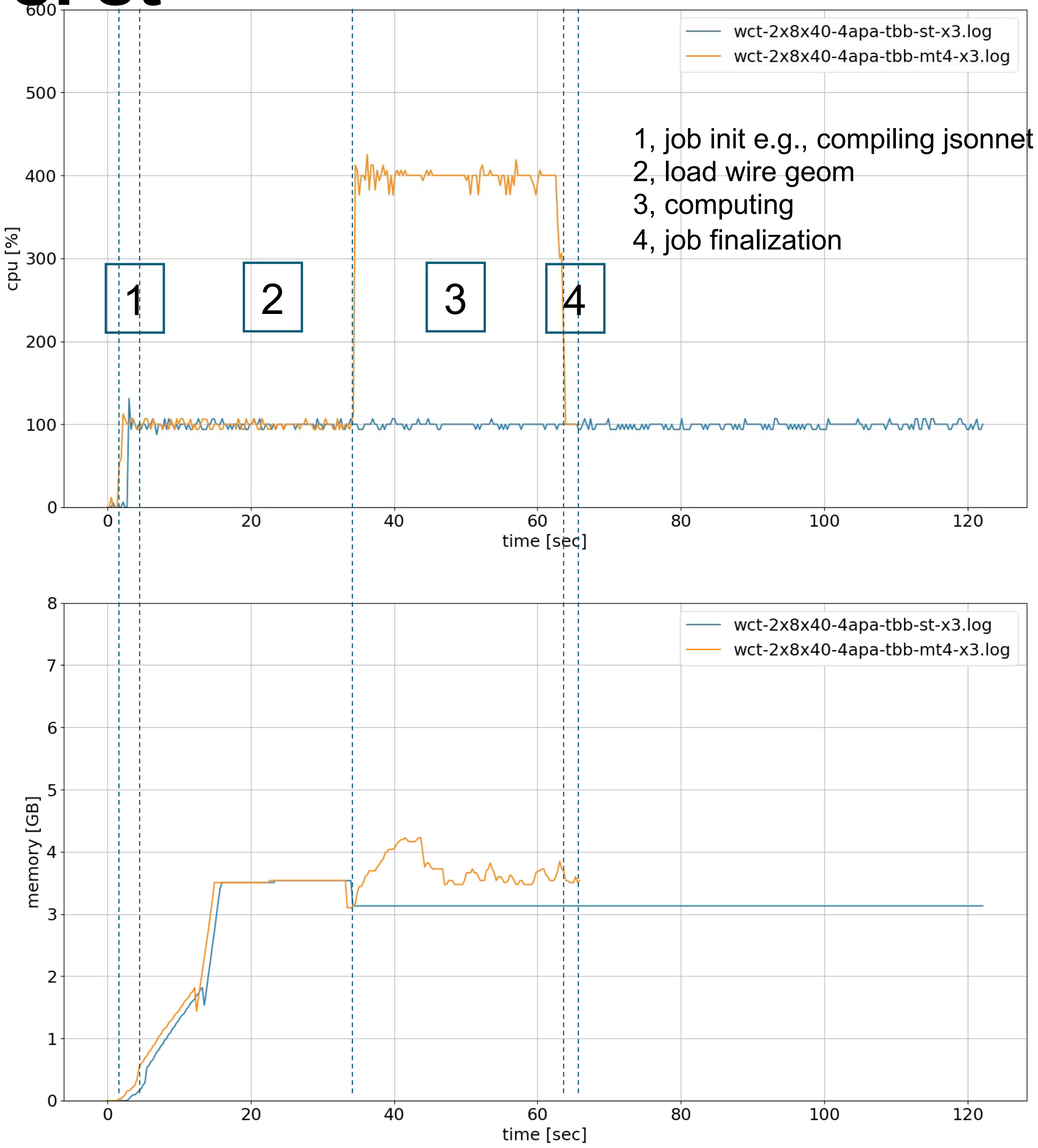


Brookhaven National Laboratory

2x8x40-4apa (full 10kt) mt4 vs. st

core speed up: 2.8 (dynamic CPU frequency scaling?)
per core mem: 3.8 -> 1

- longer ST section to initiate the wire geom.
- real benefit if
    - **MT portion of a job is large**
    - **Shared memory between threads is large**



Portable Parallelization

M. Lin etc, ACAT2022

HEP-CCE and Portable Parallelization Strategies

- **Kokkos:** a C++ abstraction layer (library) that supports parallel execution for different **host** and **accelerator** architectures.
- **SYCL:** a **specification** for a cross-platform C++ abstraction layer.
- **OpenMP/OpenACC**: Directive-based programming models for different **host** and **accelerator** architectures
- **Alpaka:** C++ abstraction layer similar to Kokkos
- **std::par:** language-based parallelism from C++ Standard
- **HIP:** originally an abstraction layer for CUDA and ROCM. Being extended to also support OneAPI.

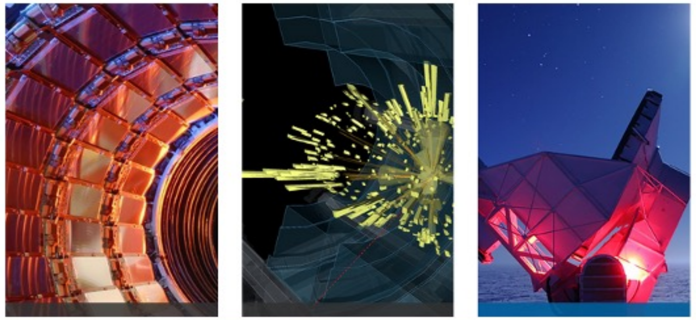
ATLAS EXPERIMENT

CMB-S4 Next Generation CMB Experiment

CMS Compact Muon Solenoid

DUNE

LSST DESC Dark Energy Science Collaboration



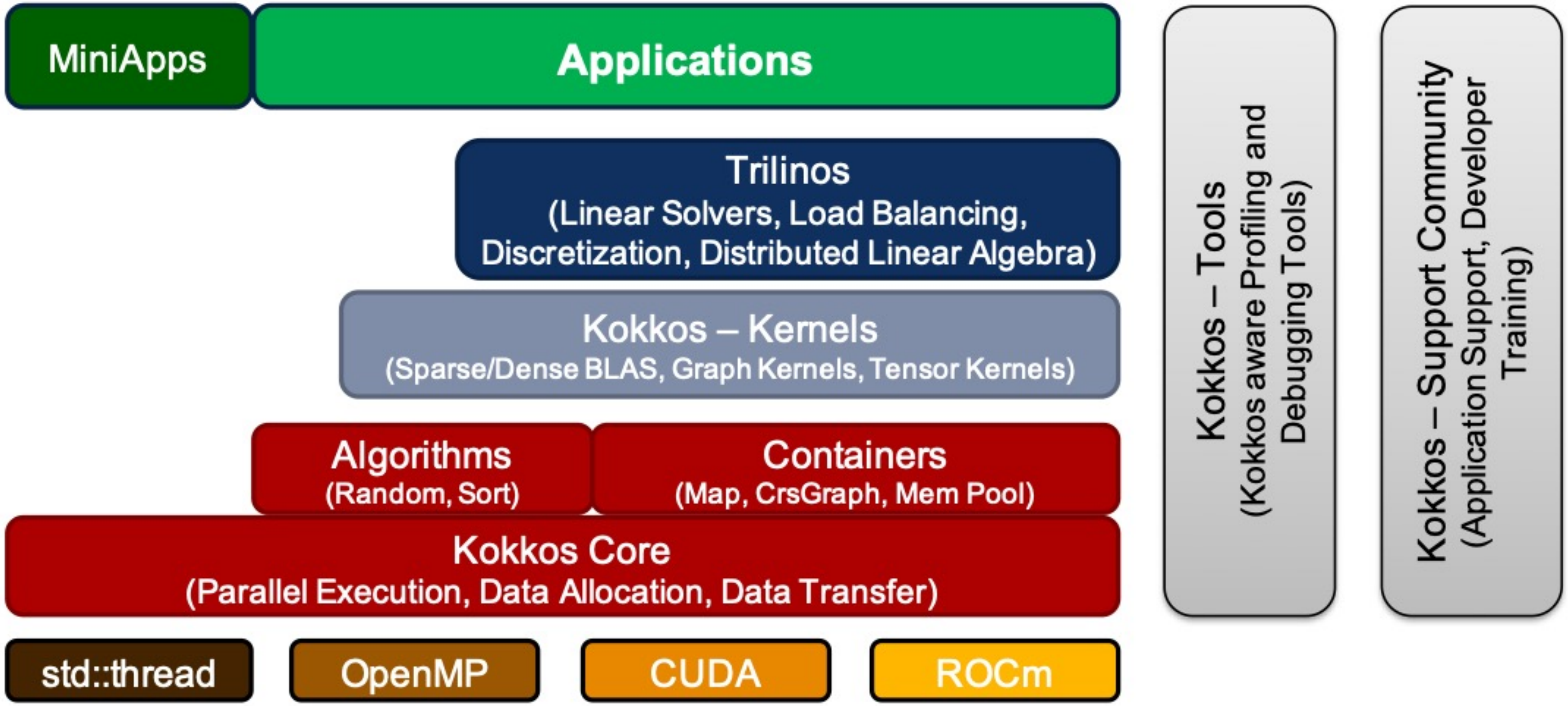
More details: CCE-PPS Overview Poster on **Wednesday:** https://indi.to/k7Bsk

HEP-CCE involves four US labs, six experiments.
Salman Habib (ANL) PI, Paolo Calafiura (LBNL) co-PI

U.S. DEPARTMENT OF ENERGY Office of Science | Argonne NATIONAL LABORATORY | BERKELEY LAB Bringing Science Solutions to the World | Brookhaven National Laboratory | Fermilab | HEP-CCE

3

Example: Kokkos

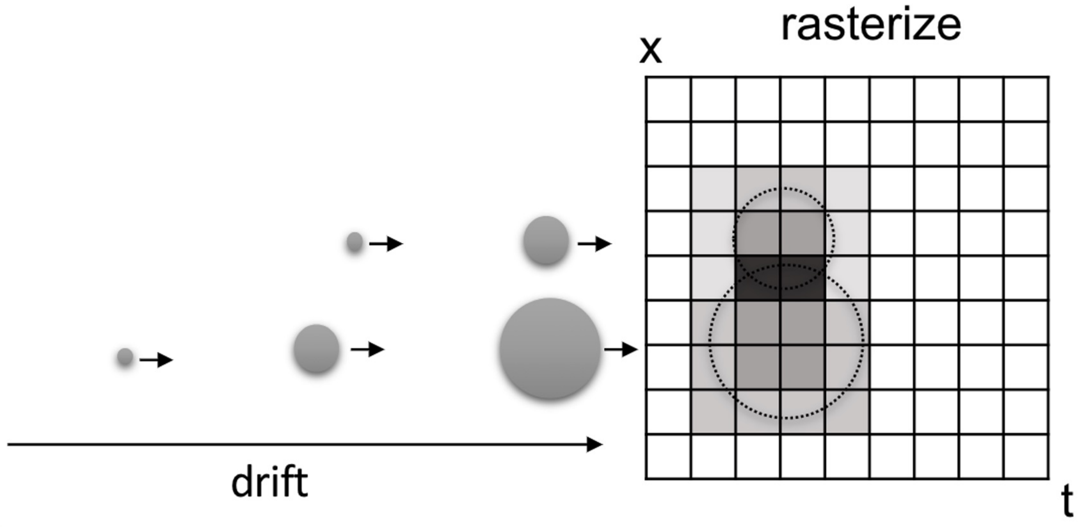


Wire-Cell Simulation Major Steps

Three major steps of LArTPC simulation with Wire-Cell - a representative workflow

1. Rasterization: depositions ⟶ patches (small 2D array, ~20×20)
    - # depo ~100k for cosmic ray event
2. Scatter adding: patches ⟶ grid (large 2D array, ~10k×10k)
3. FFT: convolution with detector response

rasterization and scatter adding



Convolution theorem:
convolution in time/space domain

$$M(t,x) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} R(t-t', x-x') \cdot S(t', x') dt' dx' + N(t,x),$$

$$\downarrow$$

multiplication in frequency domain

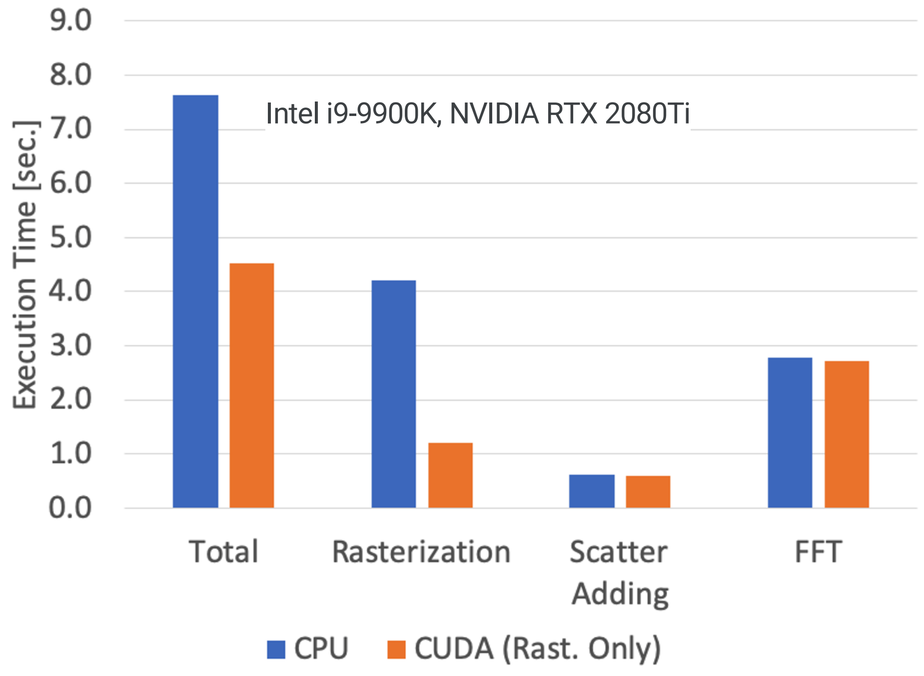
$$S(t,x) \xrightarrow{FT} S(\omega_t, \omega_x),$$
$$M(\omega_t, \omega_x) = R(\omega_t, \omega_x) \cdot S(\omega_t, \omega_x),$$
$$M(\omega_t, \omega_x) \xrightarrow{IFT} M(t,x).$$

Brookhaven National Laboratory

Initial CUDA porting

First CUDA porting focused on the Rasterization step:

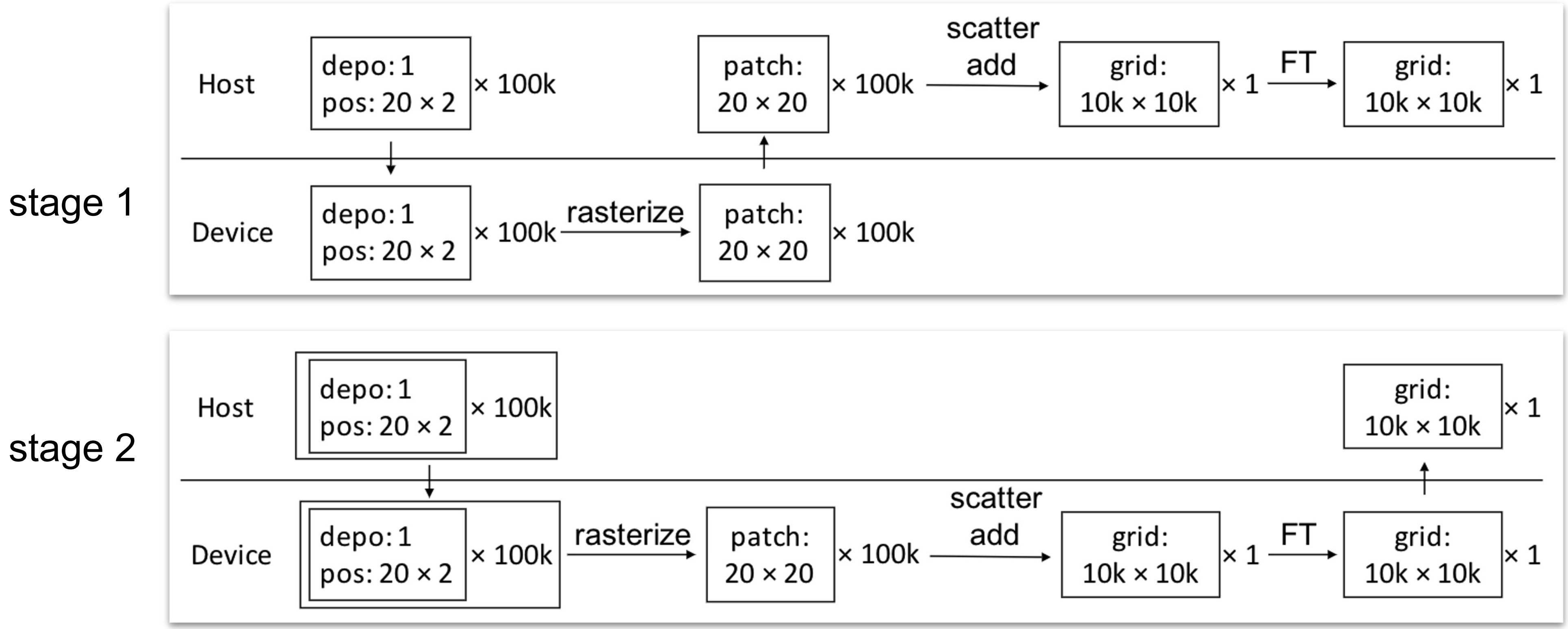
- 3× speedup for the Rast. step
  - parallelization at single patch level
  - RNG factored out ⟶ random number pool
- simulation results statistically consistent with CPU version



Kokkos Porting

Two stage porting strategy

1. partial porting - port only step 1, rasterization
2. full porting
   a. more workloads for parallelization
   b. batched device-host data transfer



Brookhaven National Laboratory

wire-cell-gen-kokkos

Benchmarking: The same code was tested on three different architectures: 24-core AMD Ryzen Threadripper 3960X for reference CPU implementation (CPU-ref) and Kokkos with OpenMP backend running 48 threads (Kokkos-OMP48), NVIDIA V100 GPU for Kokkos-CUDA and AMD Raedon Pro VII for Kokkos-HIP.

Computation [secs]	CPU-ref	Kokko-CUDA	Kokkos-HIP	Kokkos-OMP48
Rasterization	10.45	0.05	0.04	0.15
ScatterAdd	1.14	0.0006	0.007	0.013
FFT	5.44	0.71	2.50	13.3
Total Time	18.04	0.99	2.77	13.7

Table 1: Timing for the main computational tasks on different architectures averaged over 10 runs each.

- FFT is not parallelized for CPU-ref or Kokkos-OMP48. The slowdown may be due to implementation difference. Detailed investigation is ongoing.
- We get an overall speedup of $18\times$ on V100, and $7\times$ on Raedon Pro VII.
- The GPUs are still under utilized and can be shared by several parallel processes to gain further speedup using *e.g.* CUDA MPS.

[1] Z. Dong, K. Knoepfel, M. Lin, B. Viren, H. Yu and K. Yu, vCHEP 2021, arXiv: 2104.08265
[2] Z. Dong, K. Knoepfel, M. Lin, B. Viren, H. Yu and K. Yu, ACAT 2021 poster, arXiv:2203.02479

Next for WCT-PPS

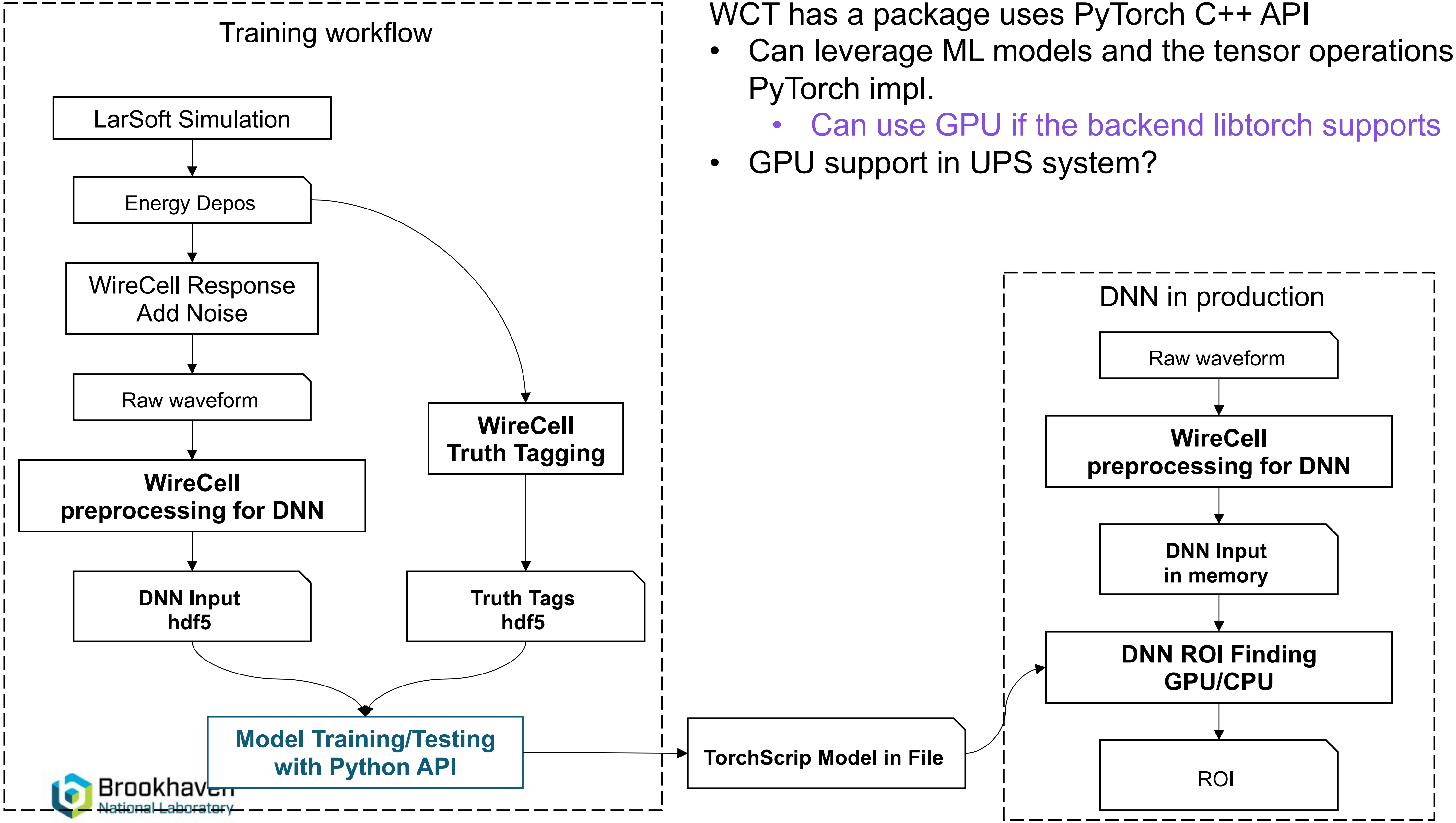
- More validation of the Kokkos wct-gen porting results

- Port wct-sigproc to Kokkos
    - deconvolution – easier to port
    - ROI finding
        - traditional – heuristic logics – harder to port
        - DNN – handled by ML backends – validation needed

Brookhaven National Laboratory

PyTorch: AI/ML and more

WCT has a package uses PyTorch C++ API
- Can leverage ML models and the tensor operations from PyTorch impl.
  - Can use GPU if the backend libtorch supports
- GPU support in UPS system?



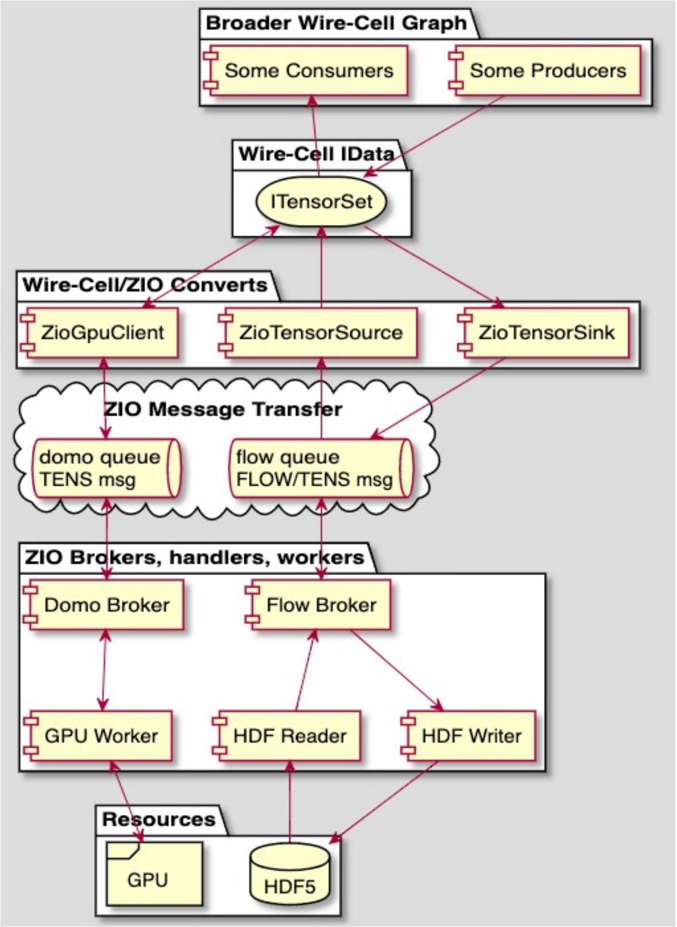
Distributed computing with ZIO

ZeroMQ based distributed computing framework by B. Viren

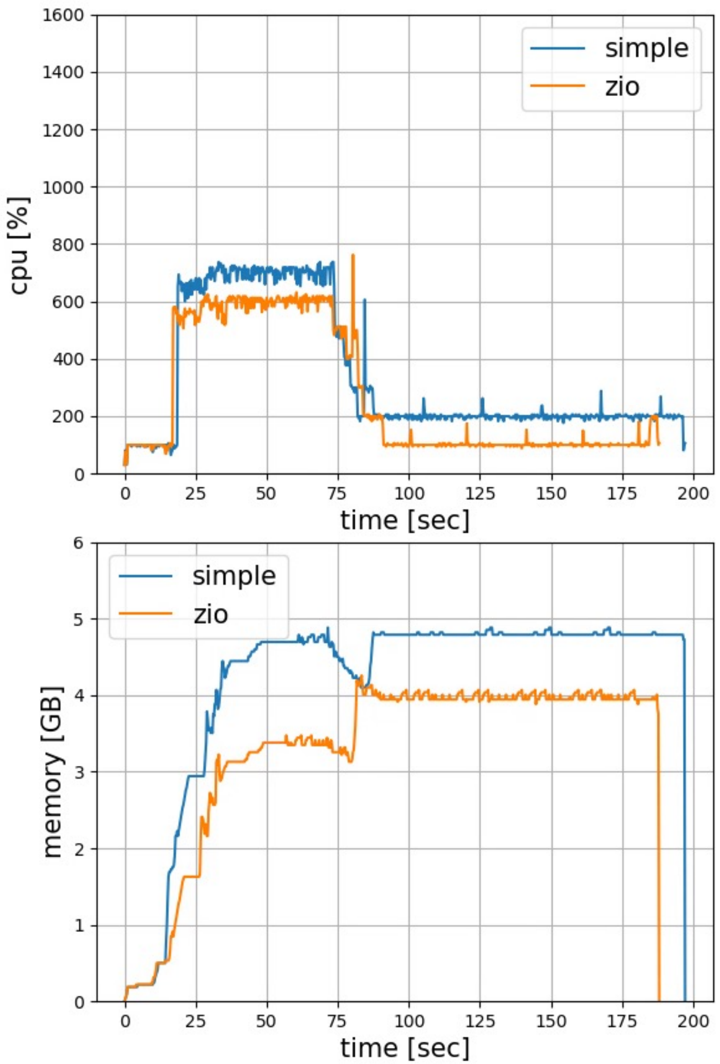
Can further balance computing resources between nodes

Mechanically working tested in an DNN inferencing task

https://brettviren.github.io/zio/whytos.html



Example running PyTorch model inferencing
Host resource uses:



- running with ***lar***
- chain: frame ⟶ nf ⟶ sp ⟶ dnn-roi ⟶ frame
- tcp protocol
- worker(idle): 2.8 GB RAM, 1.6GB VRAM

IDFT: interface for multiple FFT impl.

- IDFT: easy to swap FFT implementations.
- Data copying between CPU/GPU needed
- Using IDFT alone seems not the optimal way to speed things up
    - useful add-on when having idling GPU cycles
- Still under development

FftwDFT

```
[21:15:51.634] I [ timer   ] Timer: WireCell::Gen::DepoTransform : 16.32 sec
[21:15:51.634] I [ timer   ] Timer: WireCell::Sio::FrameFileSink : 4.51 sec
[21:15:51.634] I [ timer   ] Timer: WireCell::Gen::DepoSetDrifter : 0.74 sec
[21:15:51.634] I [ timer   ] Timer: WireCell::Sio::DepoFileSource : 0.49 sec
[21:15:51.634] I [ timer   ] Timer: WireCell::Gen::Digitizer : 0.31 sec
[21:15:51.634] I [ timer   ] Timer: WireCell::Gen::AddNoise : 0.21 sec
[21:15:51.634] I [ timer   ] Timer: WireCell::Gen::Reframer : 0.03 sec
[21:15:51.634] I [ timer   ] Timer: WireCell::Gen::DepoSetFanout : 0 sec
[21:15:51.634] I [ timer   ] Timer: WireCell::Gen::FrameFanin : 0 sec
[21:15:51.634] I [ timer   ] Timer: WireCell::Gen::Retagger : 0 sec
[21:15:51.634] I [ timer   ] Timer: Total node execution : 22.609999937936664 sec
[21:15:51.635] D [   io    ] <FrameFileSink:> closing frames-pr173-cpu.tar.bz2 after 2 calls
```

cuFftDFT

```
[21:14:02.248] I [ timer   ] Timer: WireCell::Gen::DepoTransform : 13.09 sec
[21:14:02.248] I [ timer   ] Timer: WireCell::Sio::FrameFileSink : 6.01 sec
[21:14:02.248] I [ timer   ] Timer: WireCell::Gen::DepoSetDrifter : 0.89 sec
[21:14:02.248] I [ timer   ] Timer: WireCell::Gen::AddNoise : 0.67 sec
[21:14:02.248] I [ timer   ] Timer: WireCell::Sio::DepoFileSource : 0.54 sec
[21:14:02.248] I [ timer   ] Timer: WireCell::Gen::Digitizer : 0.35 sec
[21:14:02.248] I [ timer   ] Timer: WireCell::Gen::Reframer : 0.03 sec
[21:14:02.248] I [ timer   ] Timer: WireCell::Gen::FrameFanin : 0 sec
[21:14:02.248] I [ timer   ] Timer: WireCell::Gen::Retagger : 0 sec
[21:14:02.248] I [ timer   ] Timer: WireCell::Gen::DepoSetFanout : 0 sec
[21:14:02.248] I [ timer   ] Timer: Total node execution : 21.580000398680568 sec
```

Summary

Wire-Cell Toolkit Multi-Threading and Acceleration:

- Production ready: MT, PyTorch
- Need more work: Kokkos, IDFT, ZIO

Metrics for acceleration?

- Throughput per time – traditional productions
  - related to resource availability
- Absolute speed - prompt processing
  - e.g., supernova neutrino burst pointing

Brookhaven National Laboratory

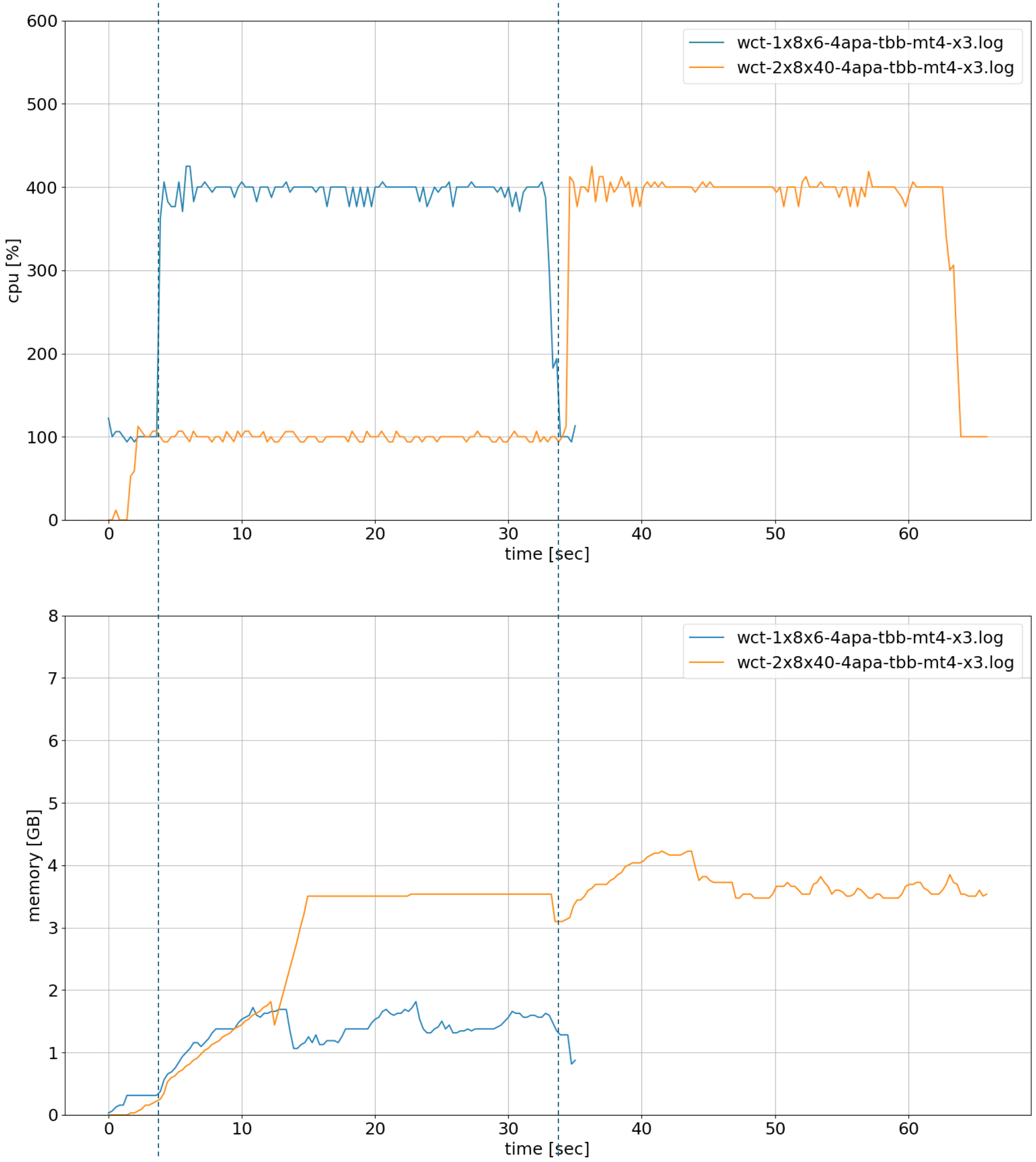
backups

Variations in the configuration

what to compare		
executable	lar (WCT as plugin of LArSoft)	wct (Wire-Cell Toolkit)
detector	1x8x6-*apa	2x8x40-*apa (full 10kt)
wct engine	pgr (Pgrapher)	tbb
thread	st	mt*
event	x*	
cfg	nofanin	mag/frametap (I/O related)

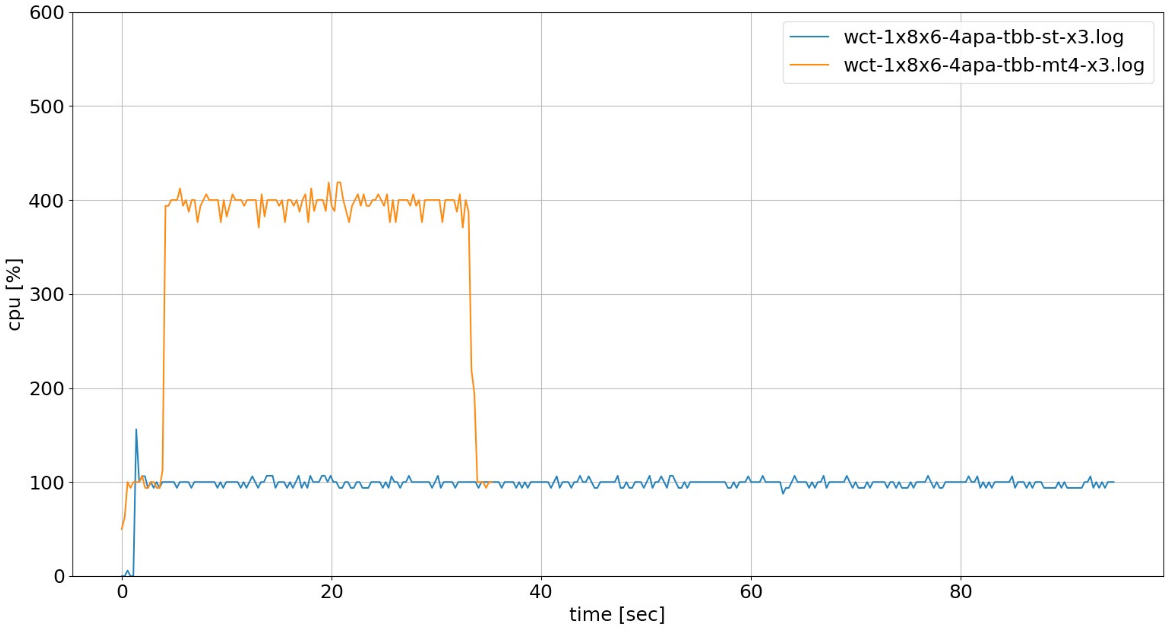
Brookhaven National Laboratory

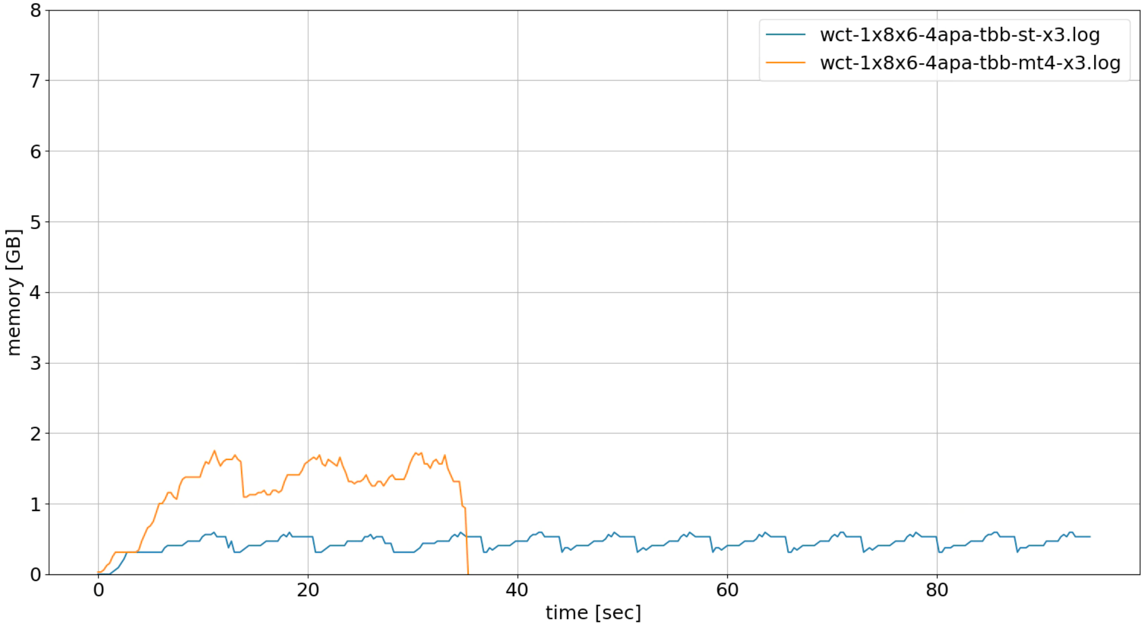
1x8x6 vs. 2x8x40



1x8x6-4apa mt4 vs. st

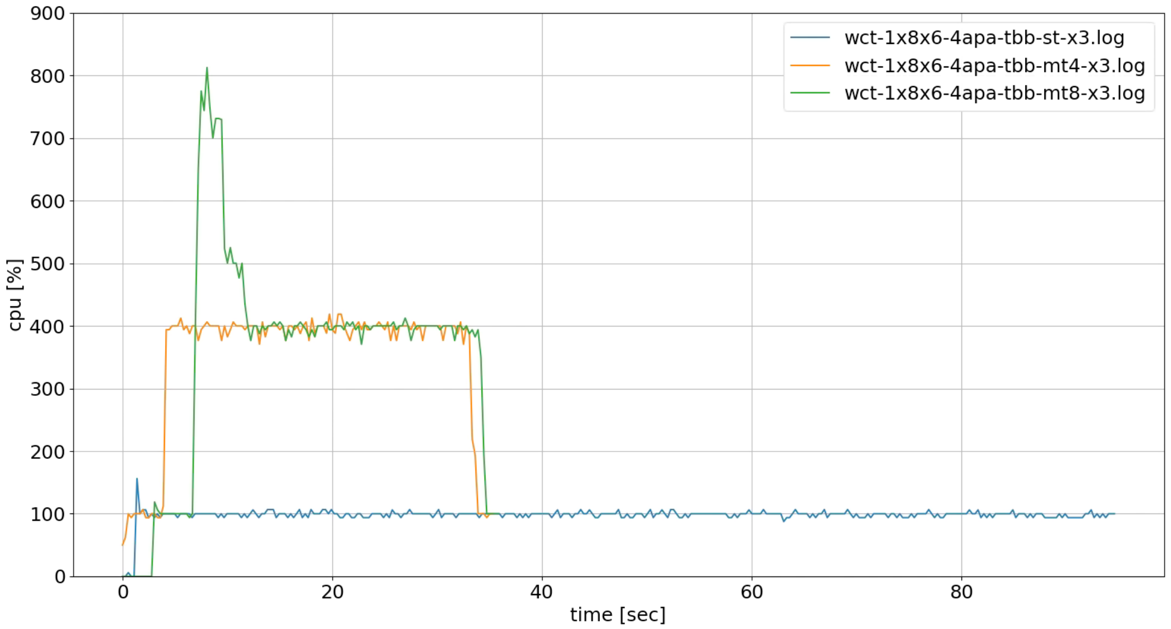
core speed up: 3
per core mem: 0.6 -> 0.43

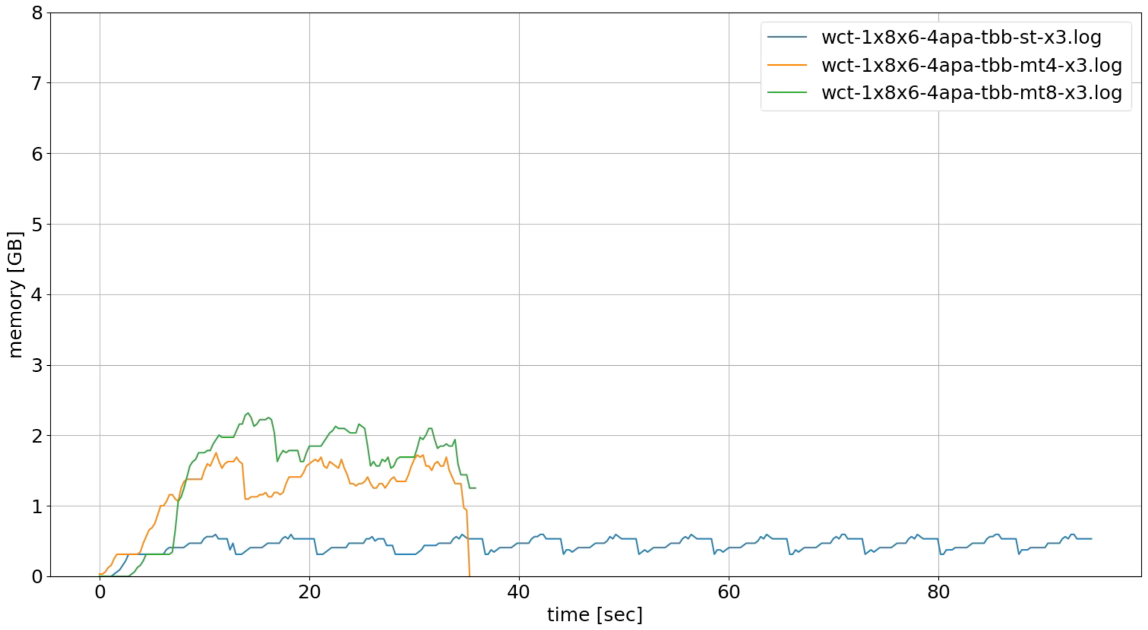




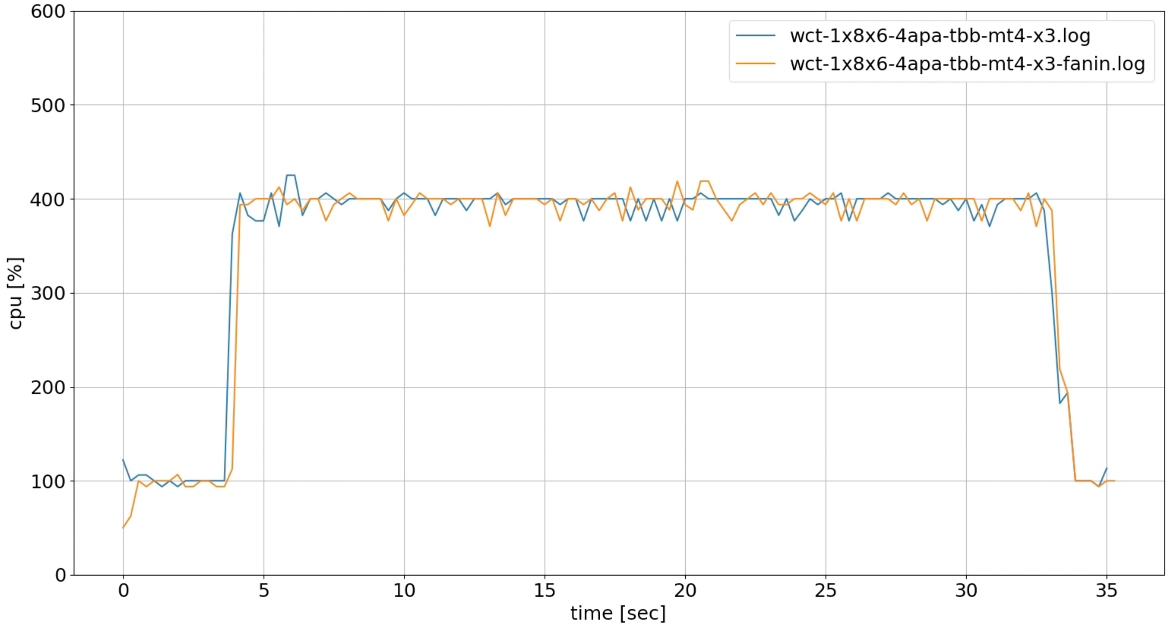
extra threads

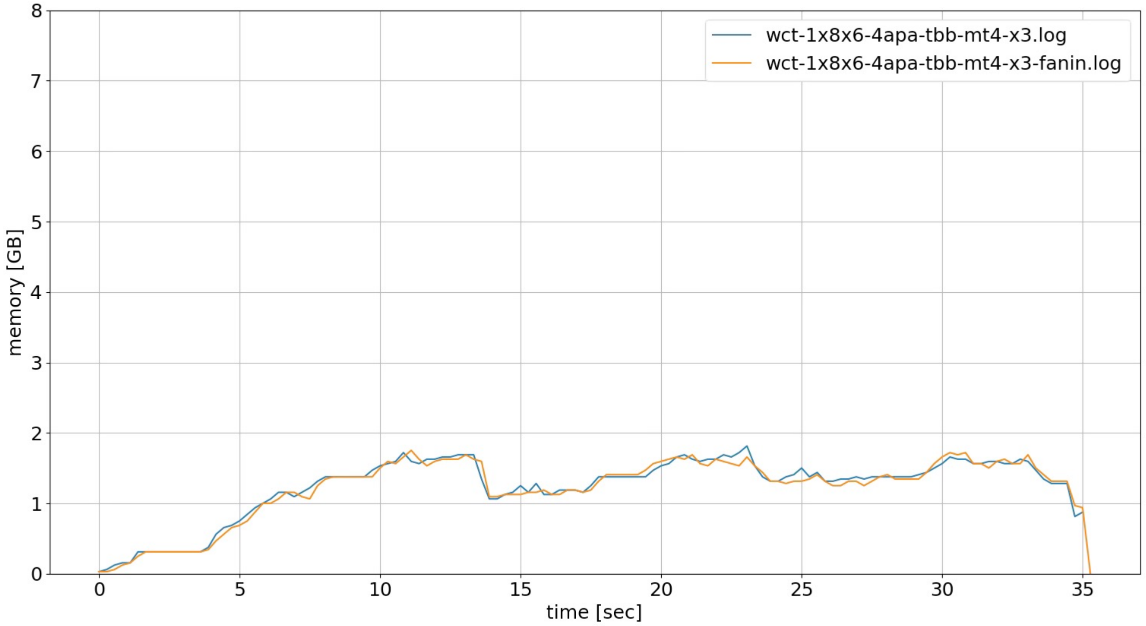
reproducible with a second run





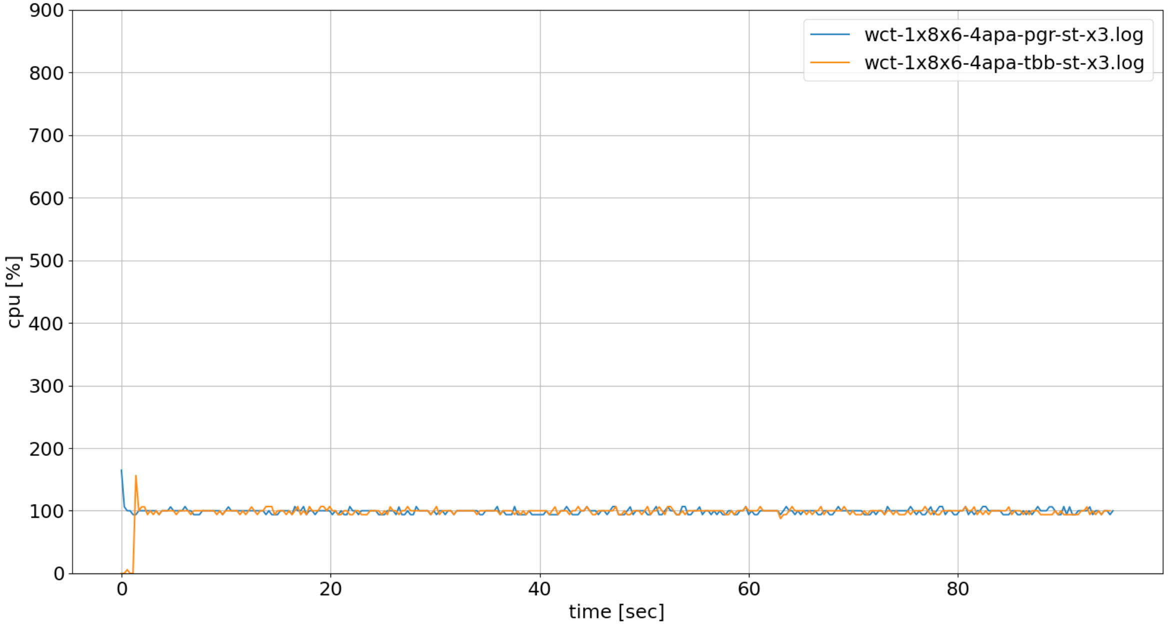
fanin vs. nofanin



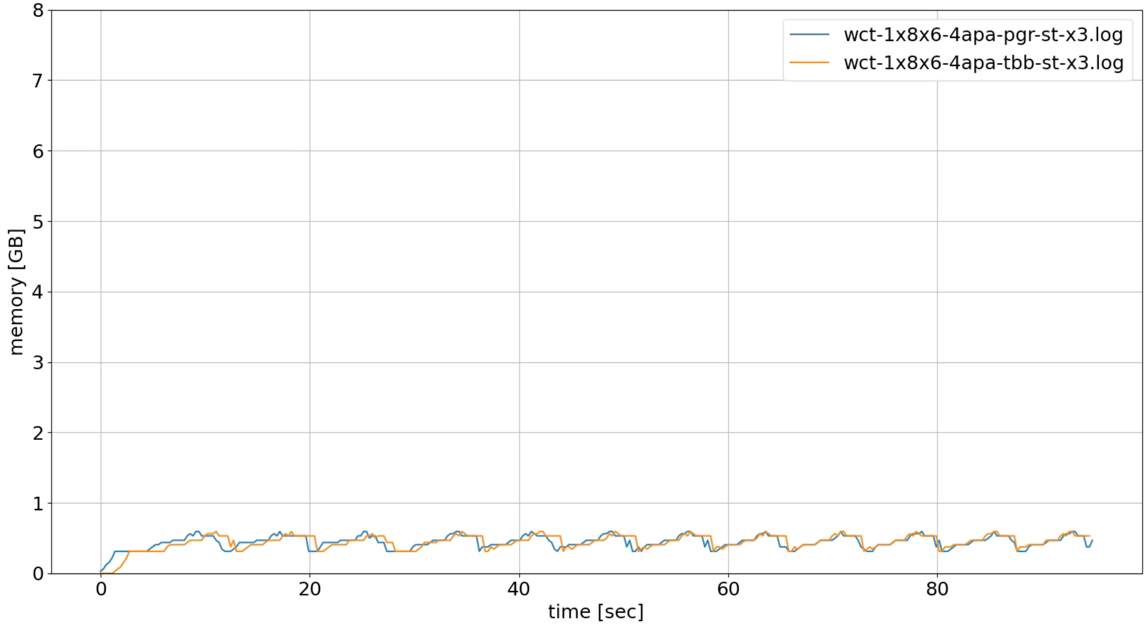


Brookhaven National Laboratory

tbb vs. pgr st



900
800
700
600
500
400
300
200
100
0
cpu [%]
0
20
40
60
80
time [sec]
wct-1x8x6-4apa-pgr-st-x3.log
wct-1x8x6-4apa-tbb-st-x3.log



8
7
6
5
4
3
2
1
0
memory [GB]
0
20
40
60
80
time [sec]
wct-1x8x6-4apa-pgr-st-x3.log
wct-1x8x6-4apa-tbb-st-x3.log