

# DUNE SPNG Updates

# Different machines with different NVIDIA GPUs for tests

Looking at the CUDA architectures

<https://developer.nvidia.com/cuda/gpus>

| Machine    | GPU          | GPU Memory   | Driver Version | CUDA Version | CUDA Architecture supported |
|------------|--------------|--------------|----------------|--------------|-----------------------------|
| Perlmutter | A100         | 40 GB, 80 GB | 580.178.04     | 13.2         | sm_80                       |
| SGPU       | L40s         | 40 GB        | 610.43.02      | 13.3         | sm_89                       |
| Local      | RTX 3500 ADA | 12 GB        | 580.173.02     | 13.0         | sm_89                       |
| wcgpu0     | RTX 4090     | 24 GB        | 555.42.06      | 12.5         | sm_89                       |

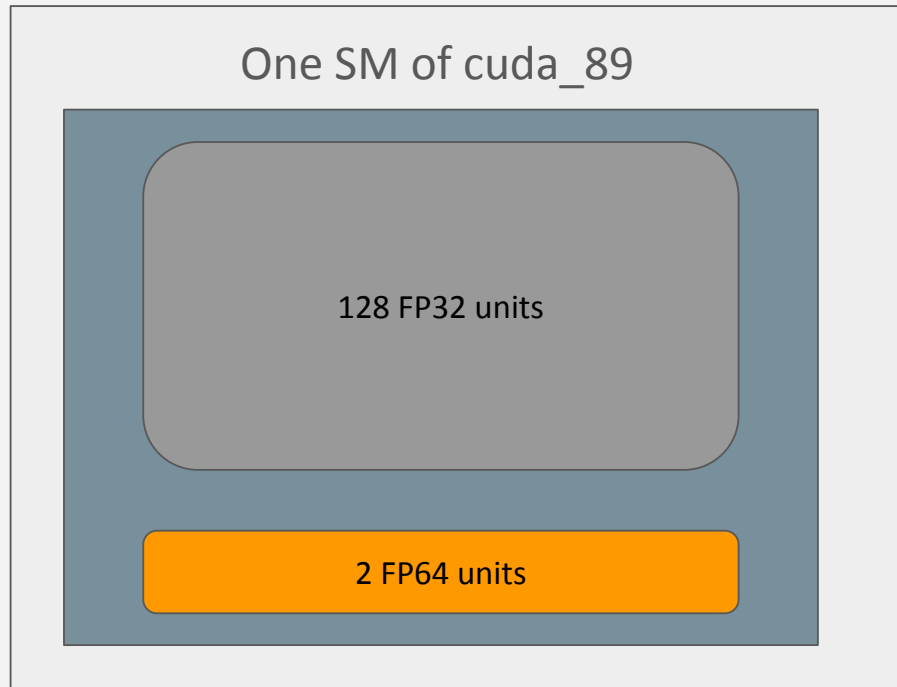
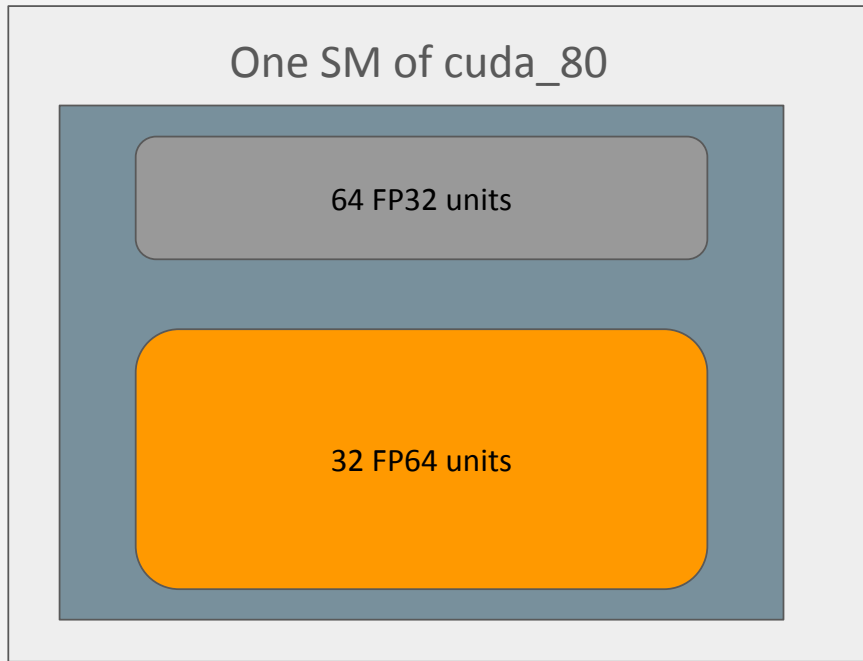
- sm\_80 and sm\_89 represent two different NVIDIA GPU Compute capabilities.
- More on this later.

# Hardware and Performance

| Parameter          | A100 40 GB | L40S   | RTX 4090 | RTX 3500 Ada |
|--------------------|------------|--------|----------|--------------|
| Architecture       | Ampere     | Ada    | Ada      | Ada          |
| SM count           | 108        | 142    | 128      | 40           |
| FP32/CUDA cores    | 6,912      | 18,176 | 16,384   | 5,120        |
| FP64 units per SM  | 32         | 2      | 2        | 2            |
| Total FP64 units   | 3,456      | 284    | 256      | 80           |
| FP32 peak (TFLOPS) | 19.5       | 91.6   | 82.6     | ~23          |
| FP64 peak (TFLOPS) | 9.7        | ~1.43  | ~1.29    | ~0.36        |

SM (Streaming Multi-processors) is an independent execution unit that schedules GPU threads and execute their instructions.

# Simplification of Streaming MultiProcessor (SM)



One A100 SM provide 32 FP64 computation lanes and 64 FP32 computation lanes.

# Warps and SM



- `cuda_80` → 32 FP64 arithmetic lanes → Can launch 32 threads of FP64 calculation in one cycle
- `cuda_89` → 2 FP64 arithmetic lanes → Can launch 2 threads of FP64 calculation in one cycle
- Higher Clock frequencies and more SM can compensate (to an extent) for the small FP64/cycle calculation in `cuda_89` gpus

An NVIDIA can launch multiple SMs in an execution cycle.

One SM can schedule 32 threads (one warp) in one cycle.

# Performance tests in Perlmutter machine

## Ultimate tests planned (Building and running workflow in different machines)

| Machine    | Tests                     | Status   |
|------------|---------------------------|----------|
| Perlmutter | Container<br>(podman-hpc) | done     |
| Perlmutter | Container (shifter)       | WIP      |
| Perlmutter | Native build              | done     |
| Polaris    | Container                 | Not done |
| Polaris    | Native build              | Not done |
| OLCF       | Container                 | Not done |
| OLCF       | Native Build              | Not done |

# NERSC Architecture

## System Specifications

| Partition | # of nodes | CPU                              | GPU                                   | NIC                                 |
|-----------|------------|----------------------------------|---------------------------------------|-------------------------------------|
| GPU       | 1536       | 1x <a href="#">AMD EPYC 7763</a> | 4x <a href="#">NVIDIA A100</a> (40GB) | 4x <a href="#">HPE Slingshot 11</a> |
|           | 256        | 1x <a href="#">AMD EPYC 7763</a> | 4x <a href="#">NVIDIA A100</a> (80GB) | 4x <a href="#">HPE Slingshot 11</a> |
| CPU       | 3072       | 2x <a href="#">AMD EPYC 7763</a> | -                                     | 1x <a href="#">HPE Slingshot 11</a> |
| Login     | 40         | 2x <a href="#">AMD EPYC 7713</a> | 1x <a href="#">NVIDIA A100</a> (40GB) | -                                   |

- Each GPU Node
  - 1 AMD EPYC CPU → 64 physical cores, 128 SMT hardware threads (2 threads per core) , 256 DDR4 memory
  - 4 A-100 GPUs (40 GB each unless asked for 80 GB ones )

# Wire-cell-toolkit build in Perlmutter (Container)

- Uses NERSC based image ([docker.io/nersc/pytorch:24.08.01](https://docker.io/nersc/pytorch:24.08.01))
  - Pytorch 2.5
  - Python 3.10
  - CUDA Toolkit → 12.6
- Uses CUDA toolkit and pytorch from base image and spack build for remaining dependencies
  - Boost → 1.90
  - Eigen → 5.0.1
  - FFTW → 3.3.11
  - fmt/spdlog → 8.1.1/1.10.0
  - Go Jsonnet → 0.22.0
  - Intel Tbb → 2023.1.0
  - JsonCPP → 1.9.6
  - Zlib-ng → 2.3.3
  - Wire-cell-data → 0.2.0
- Both dependency container and container with wire-cell-toolkit (spng-branch) [here](#)
- Container is built locally
  - Build in perlmutter login-node failed twice due to memory issue
  - Later realized that should have initiated the build in a interactive node with required memory allocation

# Wire-cell-toolkit build in Perlmutter (native)

- Files needed are [here](#) (WIP & not stable)
- Available dependencies (via module)
  - gcc → 13.2
  - cudatoolkit → 12.9
  - pytorch → 2.8.0
  - cmake → 3.30.2
- Remaining dependencies in spack-environment
  - Boost, Eigen, FFTW
  - Fmt, spdlog, go-jsonnet, jsoncpp, tbb, bzip2

# Native vs Container Build

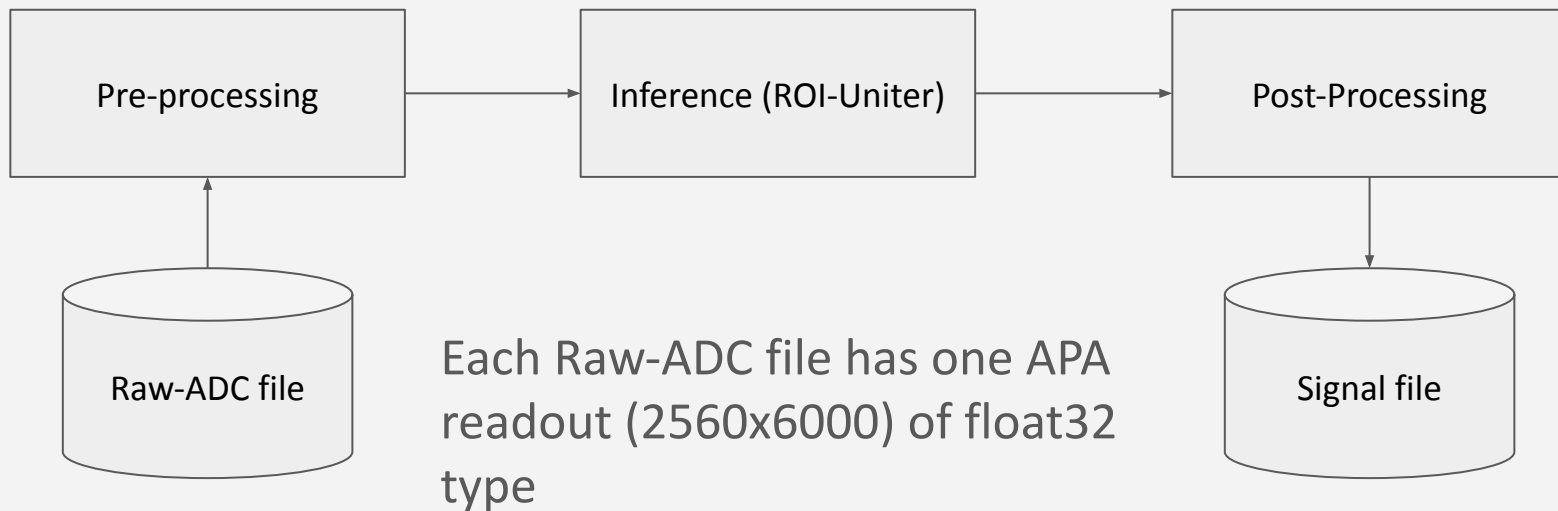
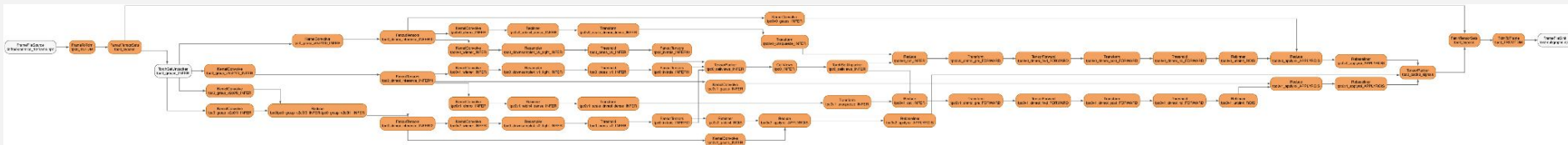
- Workflow → adc-to-spng.jsonnet
- Input → 10 ADC frames
- 1 GPU node (used 1 GPU and 1 CPU core), Pgraph engine (serial)
  - Numbers are average of 3 trials each

| Category                       | Native | Container |
|--------------------------------|--------|-----------|
| Total run-time (wall-sec)      | 29.23  | 31.91     |
| Total wire-cell run (wall-sec) | 21.845 | 21.183    |
| Input Read (wall-sec)          | 6.87   | 7.338     |
| Output Write (wall-sec)        | 4.917  | 4.936     |
| TensorForward (wall-sec)       | 3.516  | 3.496     |

- Container overhead is a one-time overhead of warming up the container (which is about ~1.7 seconds)

All tests in native build from  
here on

# Profiling and Performance of SPNG workflow



# OSP and SPNG comparison with/without GPU sharding

- 1 GPU node  $\rightarrow$  4 GPUs (A100/40 GB), 1 CPU (64 cores, 128 threads)
- 4 APA adc  $\rightarrow$  4 APA signals

| WC Processes | OSP - 1 GPU | SPNG - 1 GPU | SPNG - 2 GPU | SPNG - 4 GPU |
|--------------|-------------|--------------|--------------|--------------|
| 1            | 33.9        | 7.08         | 7.43         | 7.08         |
| 2            | 18.07       | 4.65         | 5.49         | 5.04         |
| 4            | 9.33        | 3.81         | 4.49         | 4.17         |

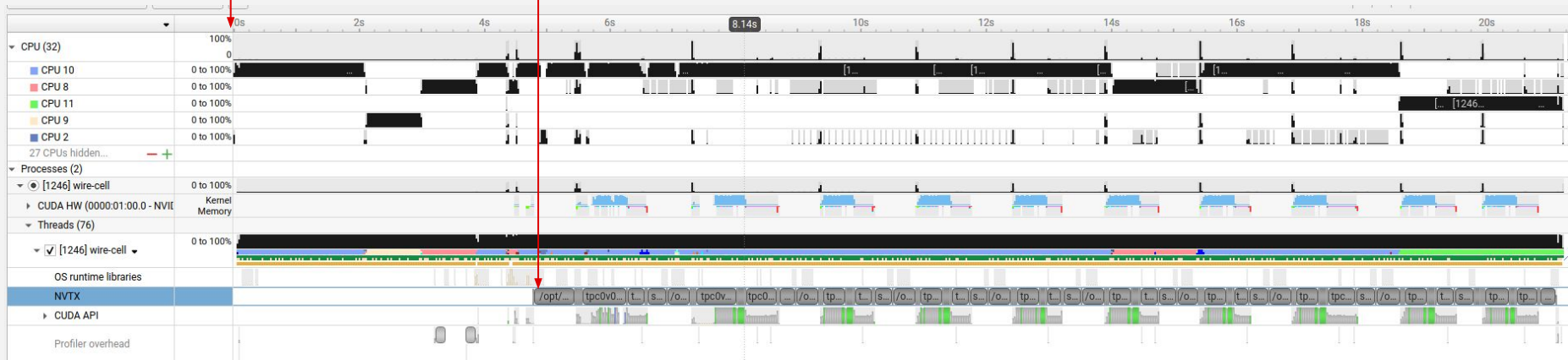
WC=1,2,4 means running 1,2 and 4 WC  
APA graphs concurrently  
1,2,4 GPU means 4-APA graph shared  
across 1,2 and 4 GPUs respectively.

Time includes graph execution time. Does not  
include initialization, configuration steps etc.

# From a local profiling report (PGraph engine)

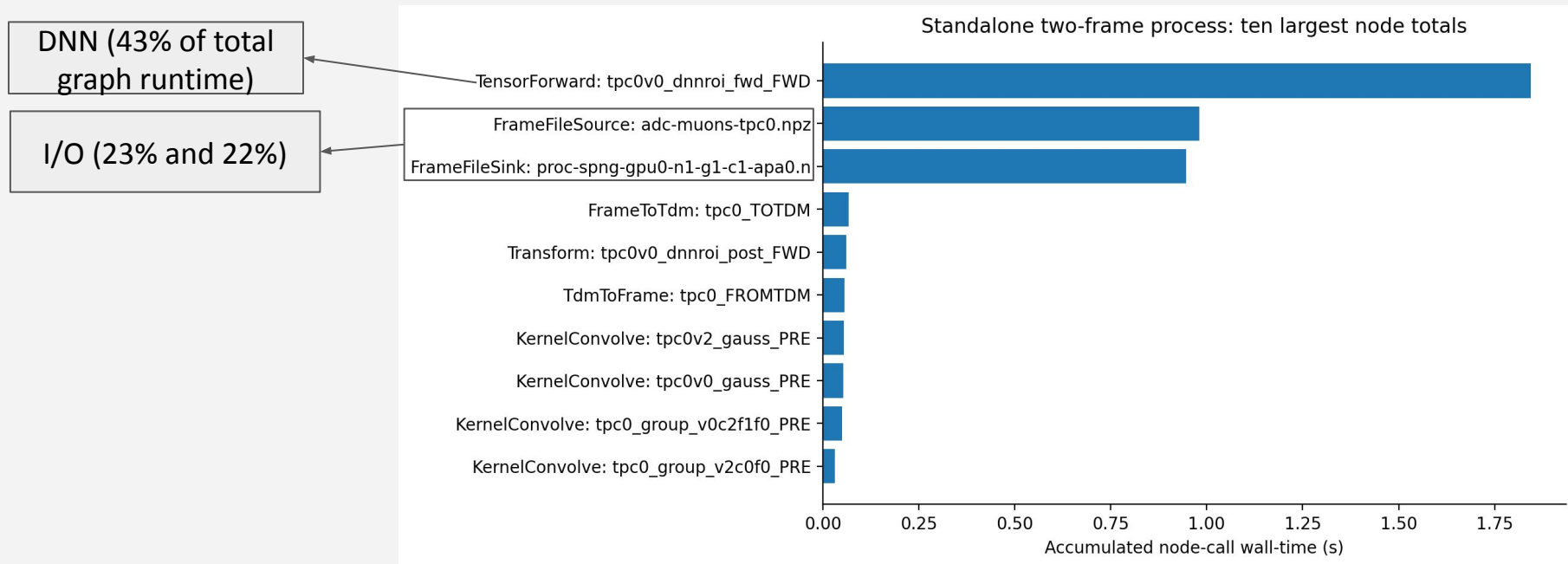
WC Process starts here

Graph starts here



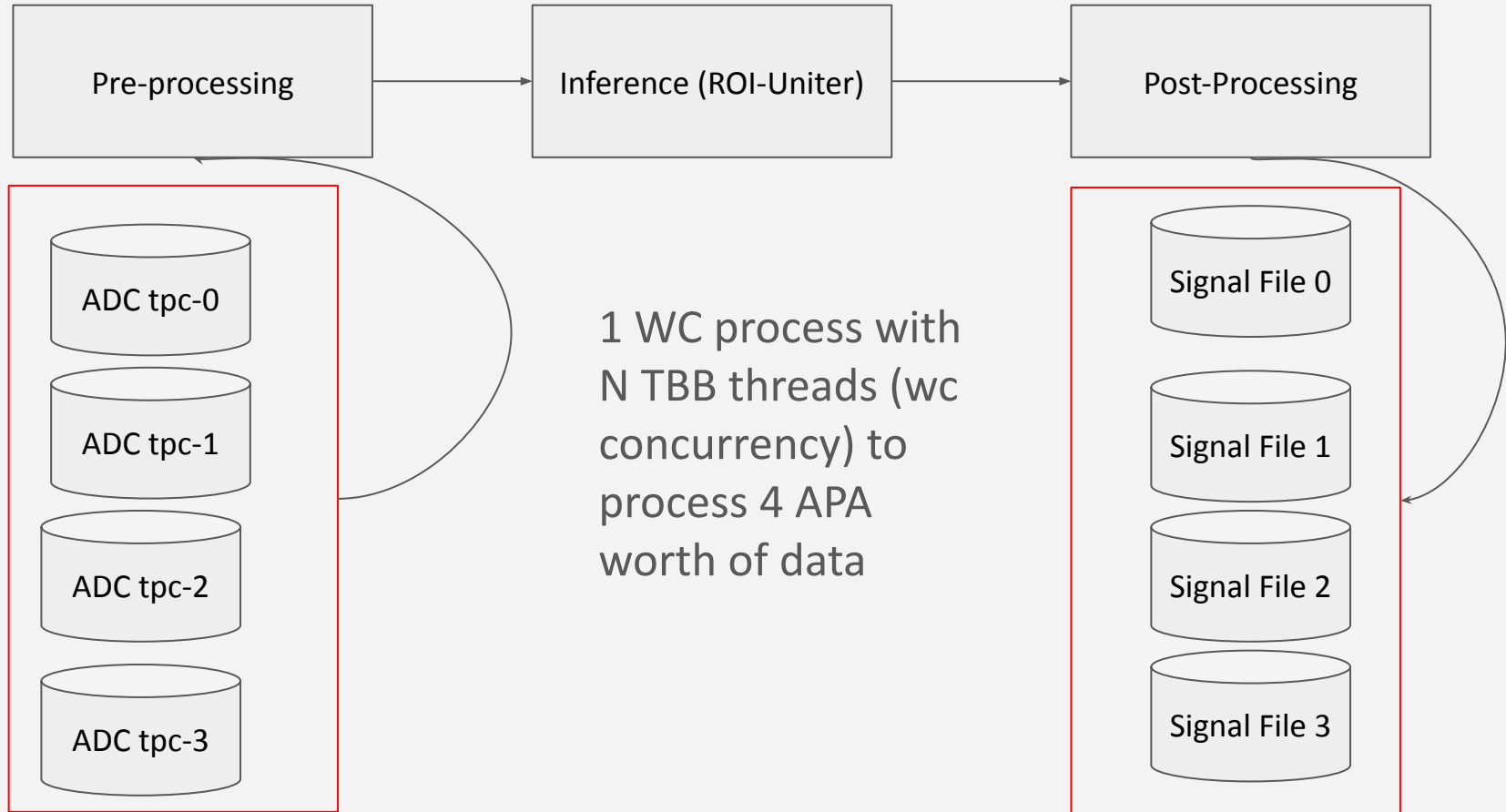
# Per Node Breakdown for a WC Graph (1 APA)

- 1 A-100 GPU, 1 CPU core, 2 ADC inputs, 2 Signal outputs

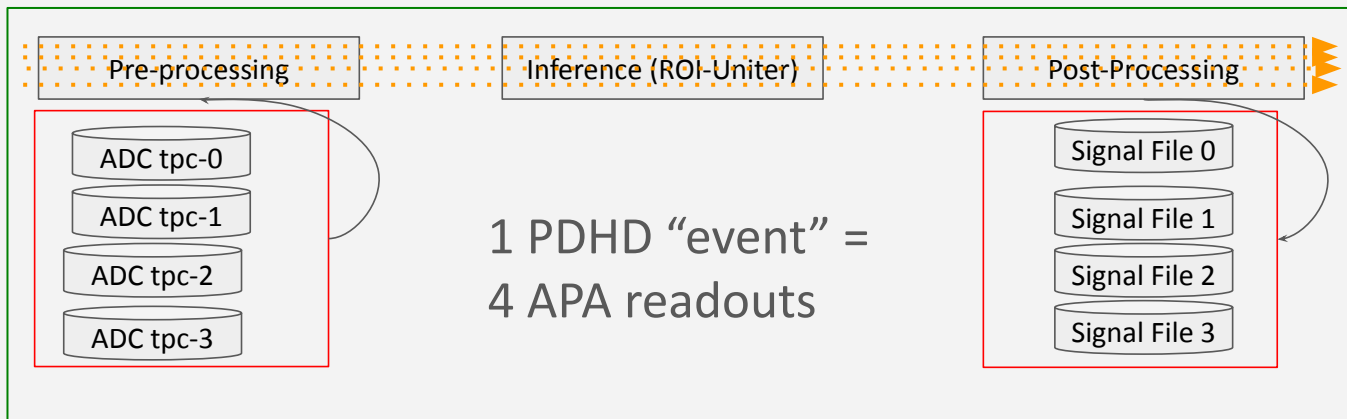


Only showing 10 largest contributors

# My understanding of Tbb concurrency and WC Processes

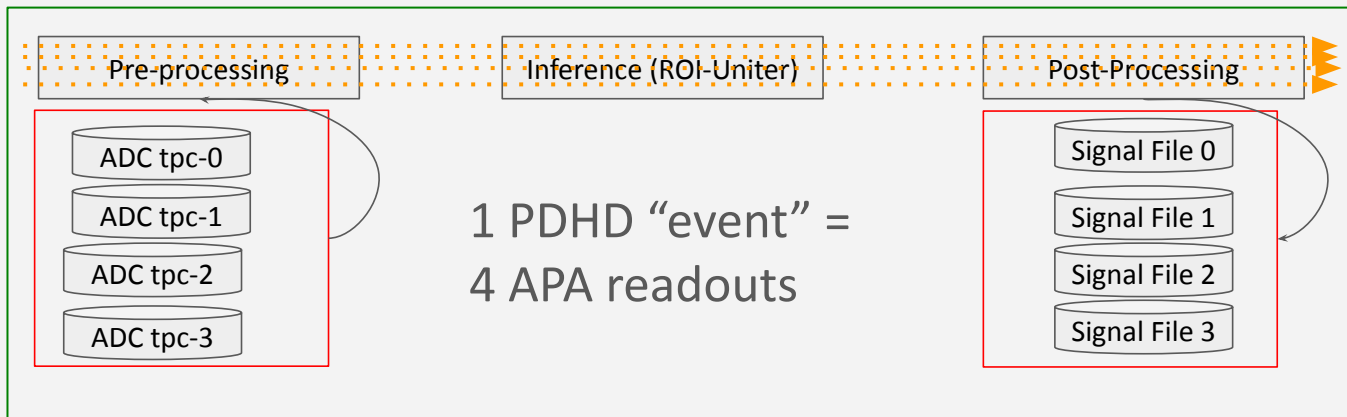


# Clarifying the terminology



WC Process 0

TBB (wc\_concurrency) =4



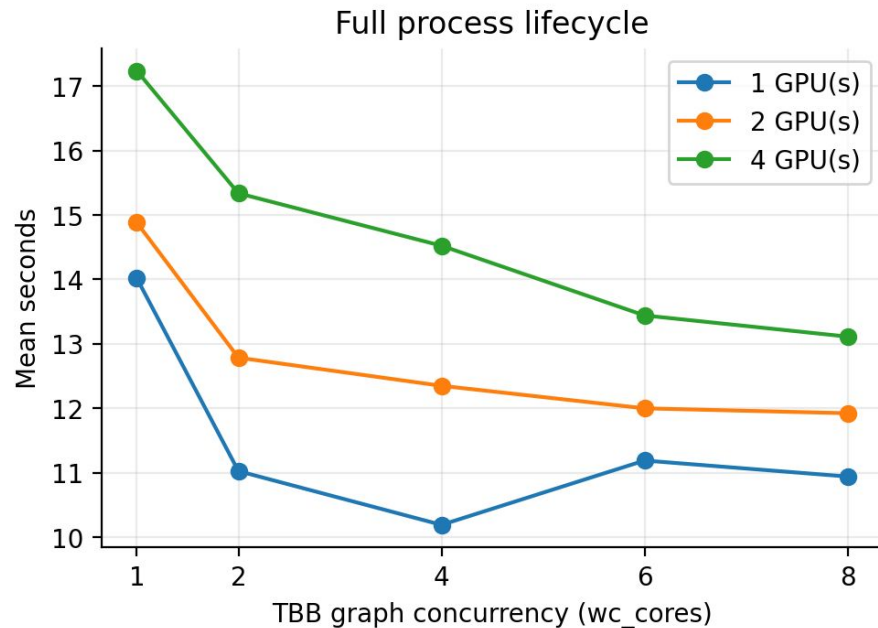
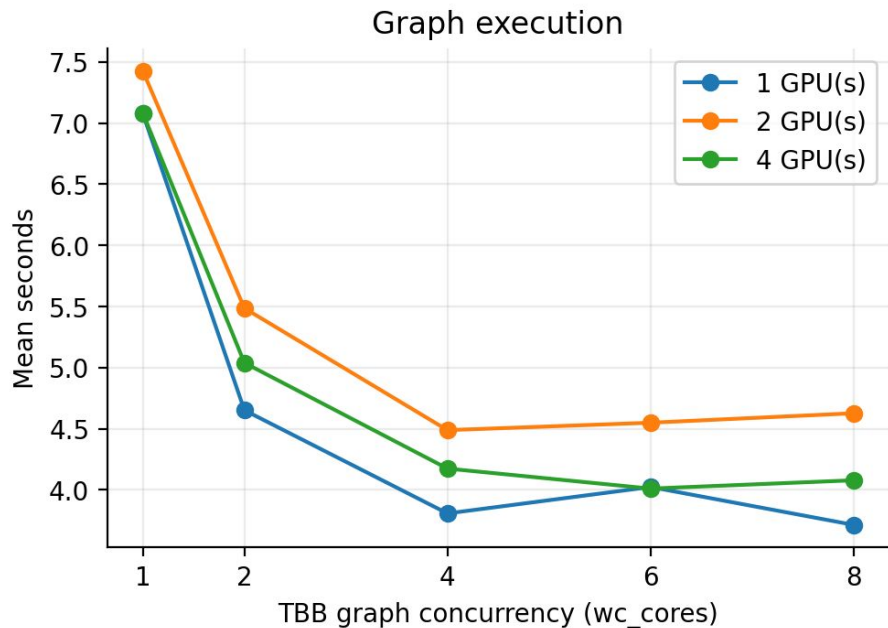
TBB is the wire-cell  
graph level  
concurrency

WC Process 1

TBB (wc\_concurrency) =4

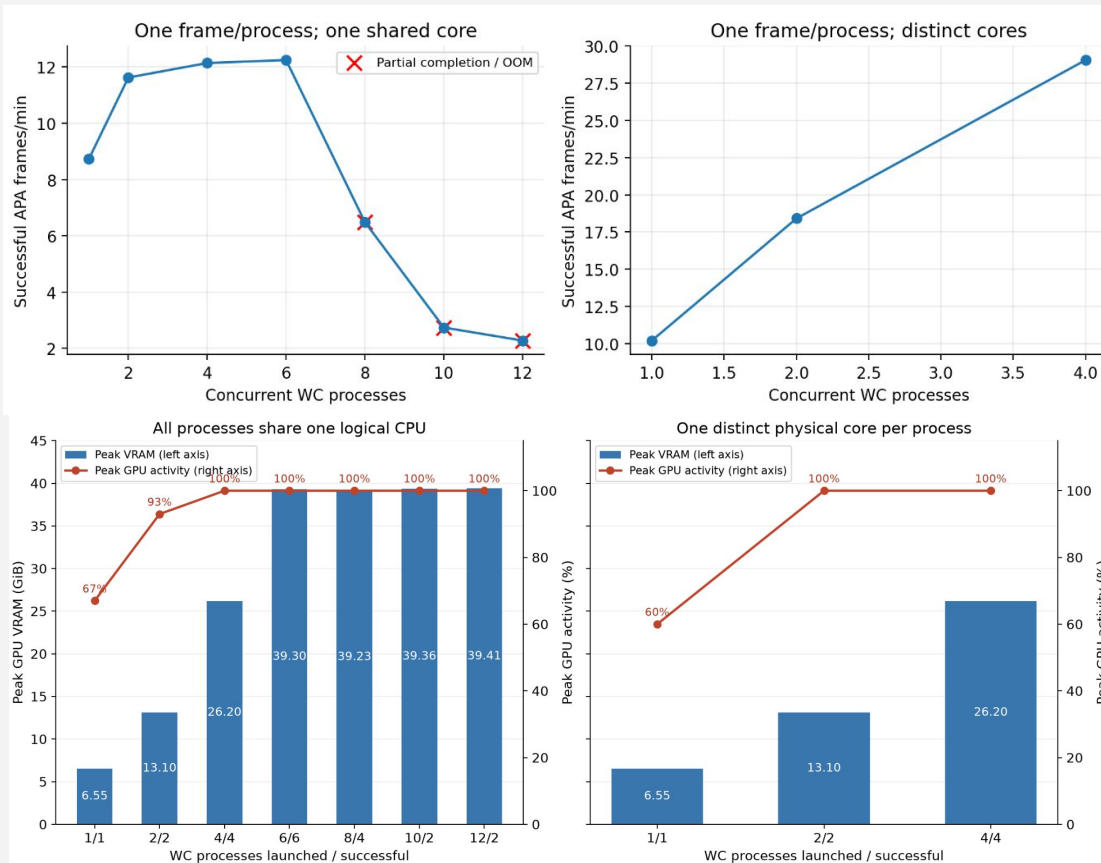
# 1 Wire-cell (WC) Process, Different Tbb concurrency

- 1 GPU node (upto 4 gpus + 1 cpu), 1 WC Process, 4 APAs



Tbb divides the WC process among N graph workers (TBB graph concurrency)

# Shared and distinct core processing (1 GPU → A100/40 GB)



Each wc processes 1 APA (Tested with 4 instances of APA 0 ADCs. PDHD has APA 0,1,2,3 for one instance)

One shared core → 1 wc process with 1 core, 2 wc process with 1 core etc

Distinct core → 1 wc process with 1 core, 2 wc processes with 2 cores etc

✗ means not all processes were successful due to CUDA Out of Memory error

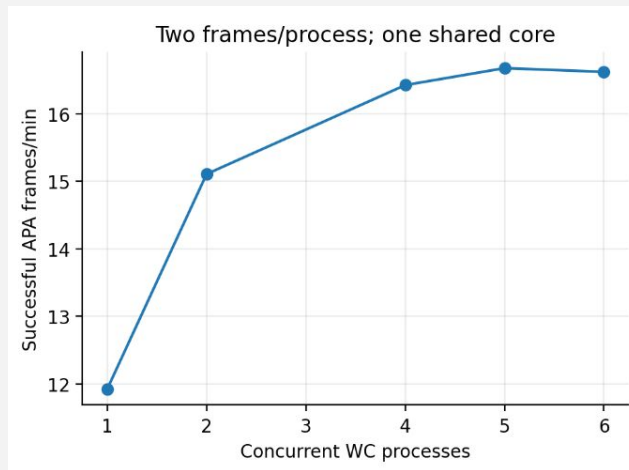
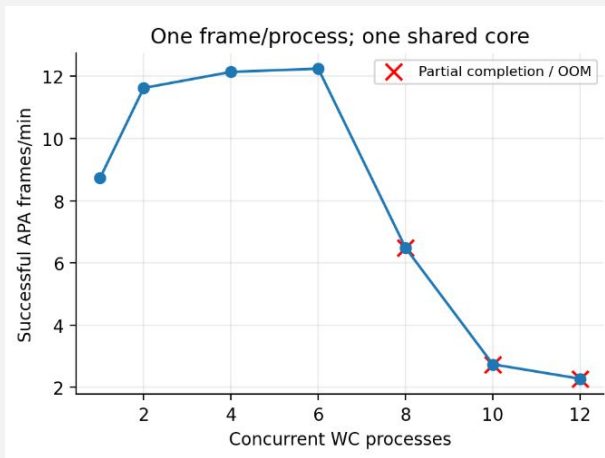
1 CPU core managing all processes (left)  
1 CPU core managing 1 process (right)

**In both cases, GPUs are being shared by multiple processes. Tbb concurrency (wc\_cores) set at 4**

For 1 GPU node, 64 cores available.

Better to allocate individual core to individual process

# 1 vs 2 APAs per process (one shared core)



Tests done with APA 0 of  
PDHD ADCs only

One process includes the overhead of warming up that shows smaller number of APAs processed per minutes in the left plot (one APA/process).

## Varying number of CPU cores for 4 WC Processes (1 GPU)

| WC Process | GPUs | CPU Cores | Wall(s) | Core-time (CPU-s) | Peak GPU (%) | Peak VRAM/GPU (GiB) | Lifecycle GPU average (%) | APA frames/min |
|------------|------|-----------|---------|-------------------|--------------|---------------------|---------------------------|----------------|
| 4          | 1    | 1         | 26.23   | 24.5              | 100          | 26.2                | 9.29                      | 18.3           |
| 4          | 1    | 2         | 15.66   | 27.33             | 100          | 26.2                | 18.87                     | 30.65          |
| 4          | 1    | 4         | 10.38   | 32.35             | 100          | 26.2                | 30.89                     | 46.25          |

- 4 APA (0,1,2,3) ADC readout = 1 PDHD readout
- 4 wc processes/per cpu core sharing 1 gpu chosen over scaling test in next slide

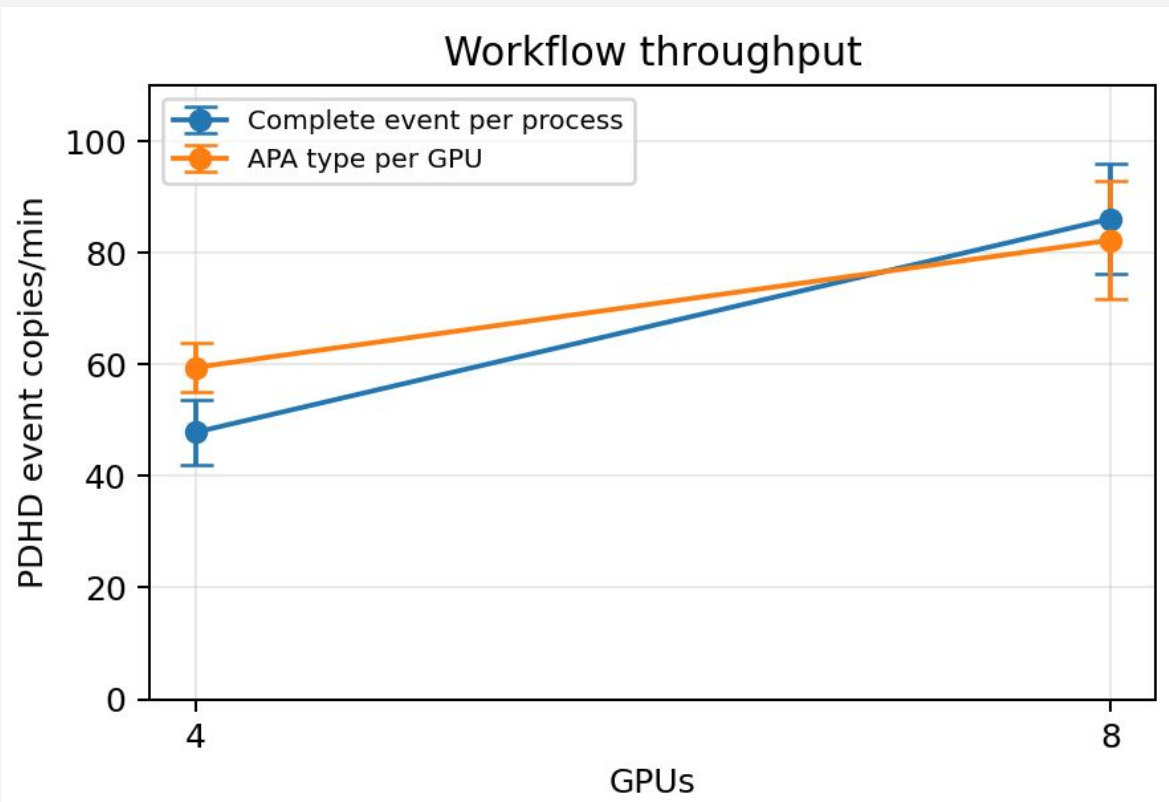
# Throughput for different ways of processing PDHD “event”

- 1 PDHD event = 4 APA ADCs
- Test in the following slide done with:
  - wc\_concurrency (tbb concurrency = 4)
  - Wire-cell Process per GPU = 4
  - 4 cpu cores per GPU
  - 1 GPU Node = 4 A100 GPU, 1 CPU (64 cores)
- **Complete event per process** → Each WC processes 1 PDHD event (4 APA ADC readouts) in 1 GPU
  - WC Process 0 → event 0 (APA 0, 1, 2, 3) in GPU 0 } 1 PDHD event
- **APA type per GPU** → Each WC processes 4 APA specific ADC readouts in 1 GPU
  - WC Process 0 → APA 0 (event 0, 1, 2, 3) in GPU 0
  - WC Process 1 → APA 1 (event 0, 1, 2, 3) in GPU 1
  - WC Process 2 → APA 2 (event 0, 1, 2, 3) in GPU 2
  - WC Process 3 → APA 3 (event 0, 1, 2, 3) in GPU 3

}

4 PDHD event

# Throughput for different ways of processing PDHD “event”



- APA type per GPU does not scale as well as Complete event per process type.
- In both cases, peak GPU VRAM per GPU ~ 26 GiB
- Peak GPU Utilization ~ 100%