Prepared for submission to JHEP

GPU-accelerated machine learning inference as a service for accelerator neutrino experiment computing

Michael Wang[1] **Tingjun Yang**[1] **Maria Acosta**[1] **Phil Harris**[2] **Burt Holzman**[1] **Wes Ketchum**[1] **Kyle Knoepfel**[1] **Jeff Krupa**[2] **Kevin Pedro**[1] **Nhan Tran**[1] **Marco Del Tutto**[1]

[1] *Fermi National Accelerator Laboratory*

[2] *MIT*

Abstract: Machine learning algorithms are becoming increasingly prevalent and powerful in the reconstruction of events in accelerator-based neutrino experiments. These sophisticated algorithms can be computationally expensive. At the same time, the data volumes of such experiments are also increasing and billions of neutrino events each including many machine learning inferences creates data processing computing challenges. We explore a computing model where heterogeneous computing GPU co-processors are made available as a web service, where they can be efficiently deployed and be elastically right-sized for a given processing task, which we call SONIC – Services Optimized for Network Inference with Coprocessors. We integrate GPU acceleration specifically for the ProtoDUNE reconstruction chain without disrupting the native compute workflow. We accelerate the most time-consuming task, track identification, by a factor of 30 which results in a 3$\times$ speed improvement of the total processing chain of CPUs and GPUs. For this given task, we only require deploying 1 GPU for every $\sim$30 CPU threads.

Contents

1 Introduction

Fundamental particle physics has pushed the bounds of computing for decades. As detectors become more sophisticated and granular, particle beams become more intense, and the data sets grow, the processing needs of the biggest fundamental physics experiments in the world are presented with massive computing challenges.

The Deep Underground Neutrino Experiment (DUNE), the future flagship neutrino experiment based at Fermi National Accelerator Laboratory, will conduct a rich program in neutrino and underground physics, including determination of the neutrino mass hierarchy and measurements of CP violation in neutrino mixing using a long baseline accelerator-based neutrino beam, detection and measurements of atmospheric and solar neutrinos, searches for supernova-burst neutrinos and other neutrino bursts from astronomical sources, and searches for GUT-scale physics in proton decay.

The detectors will consist of 4 modules, of which at least three are planned to be 10 kton Liquid Argon Time Projection Chambers (LArTPCs). Charged particles produced from neutrino or other particle interactions will travel through and ionize the argon, with ionization electrons drifted over many meters in a high electric field, and detected on planes of sensing wires or printed-circuit-board charge collectors. What results is essentially a high-definition image of a neutrino interaction, which naturally lends itself to applications of machine learning techniques designed for image classification, object detection, and semantic segmentation. Machine learning can also aid in other important applications, like noise reduction and anomaly/region-of-interest detection.

Due to the size and long readout times of the detectors, the data volume produced by the detectors will be very large: uncompressed continuous readout of a single module will be nearly 1.5 PB per second. Because that amount of data is impossible to collect and store (not to mention process), and because most of that data will not contain interactions of interest, a real-time data selection scheme must be employed to identify and store data containing neutrino interactions. With a limit on total bandwidth of 30 PB of data per year for all DUNE modules, that data selection scheme (and any accompanying compression) must effectively reduce the data rate by a factor of around 10^6.

In addition to applications in real-time data selection, accelerated machine learning inference that can scale to processing of large data volumes will be important for offline reconstruction and selection of neutrino interactions. A total data volume of 30 PB of raw data is anticipated to be collected per year, with individual event sizes of the order of a few GB, and extended readout events (associated, for example, with supernova burst events) that may be around 100 TB per module. It will be a challenge to efficiently analyze that dataset without transformations in computing models and technology that can handle data retrieval, transport, parallel processing, and storage in a cohesive manner. Similar computing challenges exist for a wide-range of existing neutrino experiments such as μBooNE [] and NOνA [].

In this paper, we focus on the inference of deep ML models as a solution for processing large datasets with the ProtoDUNE reconstruction workflow. For ProtoDUNE, machine learning (ML) inference is the most computationally intensive module in the full event processing chain and is run repeatedly on hundreds of billions of events. A growing trend to improve computing power has been the development of hardware that is dedicated to accelerating certain kinds of computations. Pairing a specialized coprocessor with a traditional CPU, referred to as heterogeneous computing, greatly improves performance. These specialized coprocessors utilize natural parallelization and provide higher data throughput. In this study, the coprocessors employed are GPUs (Graphical Processing Units) though the approach lends itself to even accommodating multiple types of coprocessors in the same workflow. ML algorithms, and in particular deep neural networks, are at the forefront of this computing revolution due to their high parallelizability and common computational needs.

To optimally integrate the GPUs into the neutrino event processing workflow, we deploy them *as a service*: in a client-server model where the primary job with the **clients** is spawned on CPUs, as is typically done in particle physics, and the ML model inference is performed on a GPU **server**. This approach, which is called SONIC [], is in contrast to a more traditional direct connection of a GPU to each and every single CPU node. The SONIC (Services Optimized for Network Inference with Coprocessors) approach allows a more flexible computing architecture for accelerating particle physics computing workflows which can be right-sized to a given task.

The rest of this paper is organized as follows. We first discuss related works which motivated and informed this study. In Section 2, we describe the tasks for ProtoDUNE event processing and the reconstruction task for which an ML algorithm has been developed. We detail how the GPU coprocessors are integrated into the neutrino software framework as a service on the client side and how we set up and scale out GPU resources in the cloud. In Section 3, we present the results which include single job and batch job multi-CPU/GPU latency and throughput measurements. Finally, in Section 4, we summarize the study and discuss further applications and studies to perform.

Related Work

Inference as a service was first employed for particle physics in Ref. [?]. This initial study utilized custom Field Programmable Gate Arrays (FPGAs) manufactured by Intel Altera and provided through the Microsoft Brainwave platform [?]. These FPGAs achieved low-latency, high-throughput inference for large convolutional neural networks such as ResNet-50 [?] using single-image batches.

Modern deep machine learning algorithms have been embraced by the neutrino reconstruction community because popular computer vision and image processing techniques map well to the neutrino reconstruction task and the detectors which collect the data. Please describe any previous and citable work on ML models in neutrino reconstruction.

2 Setup and methodology

In this study, we focus on one specific computing workflow, the ProtoDUNE reconstruction chain, to demonstrate the approach and power and flexibility of the SONIC approach. ProtoDUNE is a prototype detector for the DUNE (Deep Underground Neutrino Experiment) far detector. It is currently the largest LArTPC ever constructed and is vital for building the technology required for DUNE. This includes the reconstruction algorithms required for extracting physics objects for the LArTPC detector technology as well as the computing workflows needed.

In this section we will describe the ProtoDUNE reconstruction workflow and the ML model which is the current computing bottleneck. We will then describe the SONIC approach and how it was integrated into the LArTPC reconstruction software framework. Then we will describe how this approach can be scaled for even larger heterogeneous computing coprocessor workflows.

2.1 ProtoDUNE reconstruction

The workload used in this paper is the full offline reconstruction chain for the ProtoDUNE detector, which is a good representative of event reconstruction in present and future accelerator-based neutrino experiments. It begins by reconstructing optical hits from pulses produced by the photon detectors from scintillation light caused by the passage of ionizing particles through the liquid argon. These hits are grouped into flashes from which various parameters are determined, including time, spatial characteristics, and number of photoelectrons detected.

After the optical reconstruction stage, the workflow proceeds to the reconstruction of LArTPC wire hits. It begins by applying a deconvolution procedure that attempts to recover the original waveforms by disentangling the effects of electronics and field responses after noise mitigation. The deconvolved waveforms are then used to find and reconstruct wire hits, providing information like time and collected charge. Once the wire hits have been reconstructed, the 2D information provided by the hits in each plane are combined with that from the other planes in order to reconstruct 3D space points. This information is primarily used to resolve ambiguities caused by the complication of having the induction wires in one plane wrapping around into another plane.

The disambiguated collection of reconstructed 2D hits are then fed into the next stage consisting of a modular set of algorithms provided by the Pandora software development kit. This stage finds the high-level objects associated with particles, like tracks, showers, and vertices, and assembles them into a hierarchy of parent-daughter nodes that ultimately point back to the candidate neutrino interaction.

The final module in the chain, *EmTrackMichelId*, is a machine learning based algorithm that classifies reconstructed wire hits as being track-like, shower-like, or Michel electron-like. This algorithm begins by constructing 48×48 pixel images whose two dimensions are the time t and the wire number w in the plane. These images, referred to as patches, are centered on the peak time and wire number of the reconstructed hit being classified. The value of each pixel corresponds to the measured charge deposition in the deconvolved waveforms corresponding to the wire number and time interval associated with the row and column, respectively, of the pixel. Inference is performed on these patches using a convolutional neural network which will be described in more detail in the next section.

2.2 Benchmark models (Mike, Tingjun)

The network model employed by the *EmTrackMichelId* module of the ProtoDUNE reconstruction chain consists of a 2D convolutional layer followed by the two fully connected layers. The convolutional layer takes each of the 48×48 pixel patches described in Section 2.1, and applies 48 separate 5×5 filters to it, using stride lengths of 1, to produce 48 corresponding 44×44 pixel feature maps. These feature maps are then fed into the first fully connected (FC) layer consisting of 128 neurons, which is, in turn, connected to the second FC layer consisting of 32 neurons. Rectified Linear Unit (ReLU) activation functions are applied after the convolutional layer and each of the two FC layers. Dropouts are implemented between the convolutional layer and the first FC layer and between the two FC layers to help prevent overfitting. The second FC layer splits into two separate branches. The first terminates into three outputs that are constrained to a sum of one by a softmax activation function and the second terminates into a single output with a sigmoid activation function that limits its

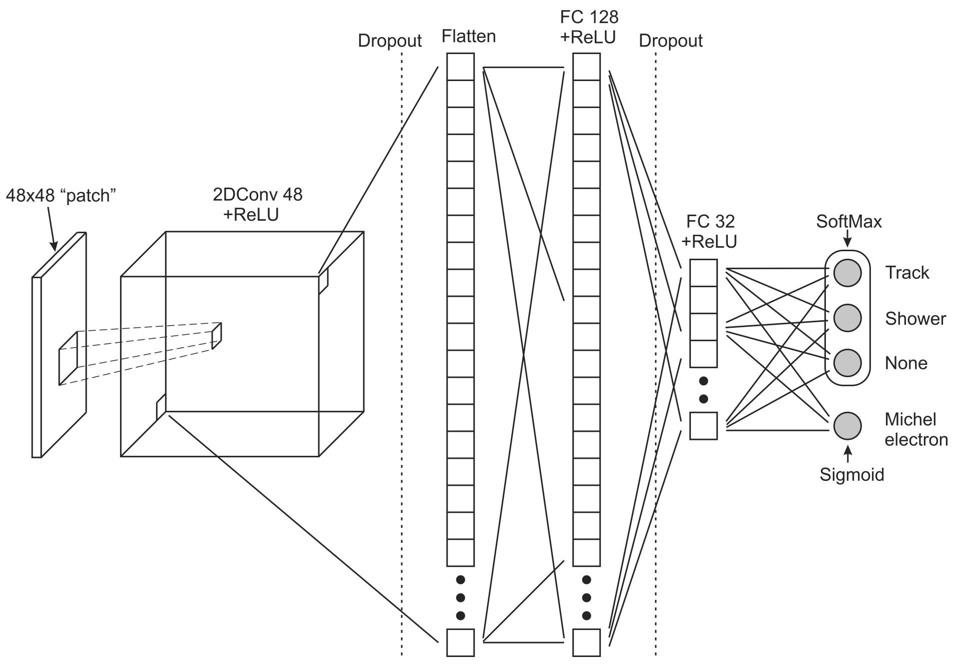


Figure 1. Architecture of the neural network used by the *EmTrackMichelId* module in the ProtoDUNE reconstruction chain.

value within a range of 0 to 1. The total number of trainable parameters in this model is 11,900,420.

2.3 Nvidia Triton inference server with LArSoft (Kevin, Mike, Kyle)

The Nvidia Triton inference server [**?**] provides a C++ client interface to send remote procedure calls (RPCs) to a server that provides one or more GPUs. These calls are based on gRPC [**?**]. The existence of a C++ client interface is key for integrating the service in the experiment software, which are also based on C++. In particular, the LArSoft [**?**] software is targeted as a common framework shared by many neutrino experiments. A new package, larrecodnn [**?**], hosts the tool that extracts the input from the neutrino event data format, transmits it in the proper format to be processed by the GPU server, and then receives the output.

This tool follows the SONIC (services for optimized network inference on coprocessors) [**?**] approach that is also in development for other particle physics applications. In this case, a synchronous, blocking call is used. The CPU usage of the workload, described in Section 2.1, is dominated by the neural network inference. Therefore, a significant increase in throughput can still be achieved by accepting the latency of the remote call while the CPU waits for the result. An asynchronous, non-blocking call would be slightly more efficient, as it would allow the CPU to continue with other work while the remote call was ongoing. However, this would require significant development in applications of task-based multithreading, as described in Ref. [**?**].

This approach has several advantages. Rather than requiring one coprocessor per CPU with a direct connection over PCIe, many worker nodes can send requests to a single GPU, as depicted in Fig. 3. This allows heterogeneous resources to be allocated and re-allocated based on demand, providing significant flexibility as well as cost reduction. The Triton software also handles load balancing for servers that provide multiple GPUs, further increasing the flexibility of the server. In addition, the Triton server can host multiple models from various machine learning frameworks. This reduces maintenance in the experimental code base, which would otherwise be required to integrate and support separate C++ APIs for every machine learning framework in use.

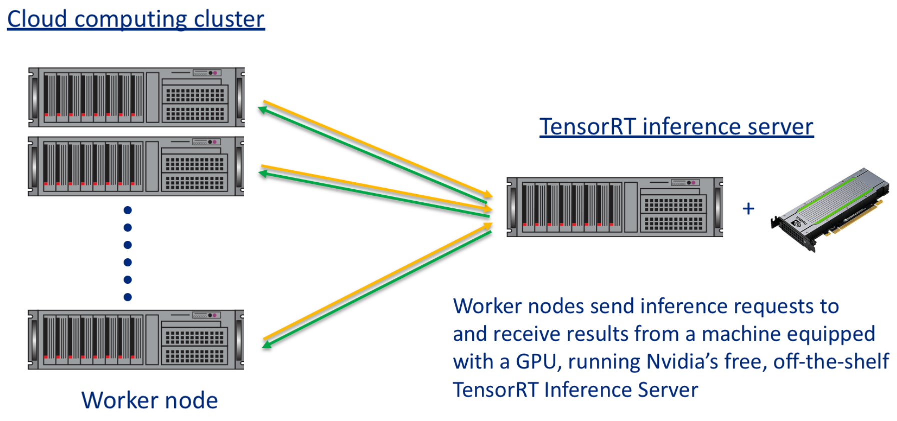

Cloud computing cluster
TensorRT inference server
+
Worker nodes send inference requests to and receive results from a machine equipped with a GPU, running Nvidia's free, off-the-shelf TensorRT Inference Server
Worker node

Figure 2. Client-server model

Can we get a description of what was needed to get this working in LArSoft?

2.4 Kubernetes scale out (Maria)

We performed tests on many different combinations of computing hardware, which provided us with a deeper understanding of networking limitations within Google Cloud and on-premises data centers. Even though the Triton Inference Server does not consume significant CPU power, the number of CPU cores provisioned for the node did have an impact on the maximum ingress bandwidth reached in our early tests.

To scale the GPU throughput in flexibly, we deployed a Google Kubernetes Engine cluster for server side workloads. The cluster was configured using a Deployment and ReplicaSet, which controlled Pods and their resource requests. Additionally, a load balancing service was deployed which distributed incoming network traffic among the Pods. We implemented Prometheus-based monitoring which provided insight on three aspects: system metrics for the underlying virtual machine, Kubernetes metrics on the overall health and state of the cluster, and inference-specific metrics gathered from the Triton Inference Server via a built-in Prometheus publisher. All metrics were visualized through a Grafana instance, also deployed within the same cluster.

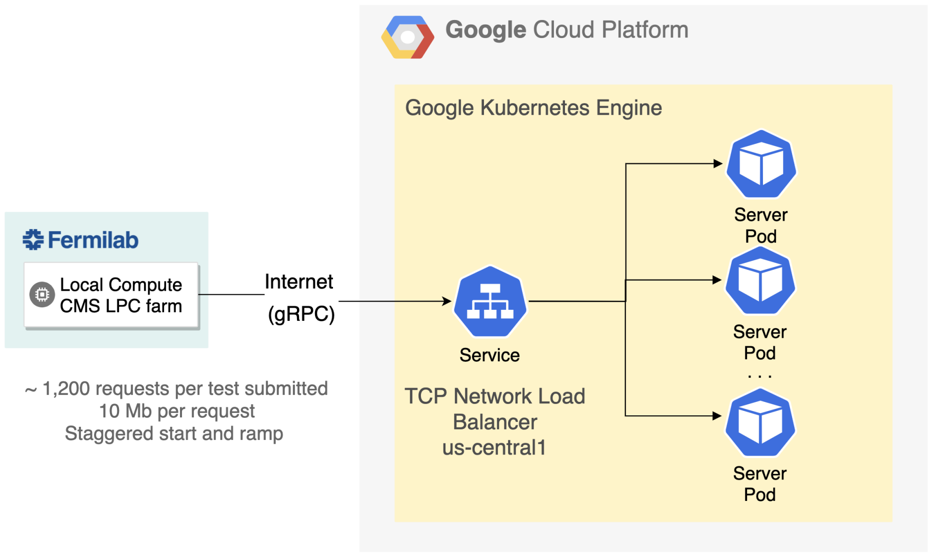


Figure 3. Base GKE setup

We kept the Pod to Node ratio at 1:1 throughout the studies, with each Pod running an instance of Triton Inference Server (v20.02-py3) from Nvidia Docker repository. Pod hardware requests aim to maximize the use of allocatable vCPU and memory, and use all GPUs available to the container.

In this scenario, it can be naively assumed that a small instance n1-standard-2 with 2 vCPUs, 7.5 GB of memory, and different GPU configurations [1, 2, 4, 8] would be able to handle the workload, which would be distributed evenly on GPUs. After performing several tests, we found that horizontal scaling would allow us to increase our ingress bandwidth since Google Cloud imposes a hard limit on network bandwidth at 2 Gbit/s per virtual CPU (vCPU) up to a theoretical maximum of 16 Gbit/s for each virtual machine [**?**].

Given these parameters, we found that the ideal setup for optimizing ingress bandwidth was to provision multiple Pods on 16-vCPU machines with fewer GPUs per Pod. For GPU-intensive tests, we took advantage of having a single point of entry through Kubernetes load balancing and provisioning multiple identical Pods behind the scenes, where the sum of the GPUs attached to each Pod is the total GPU requirement.

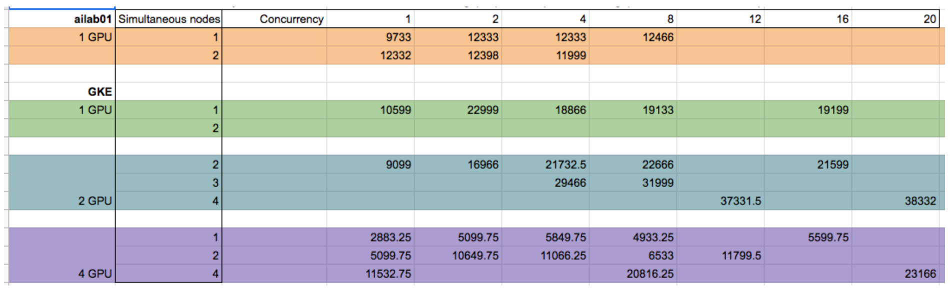
3 Results

Given the setup we describe in the previous section for machine learning acceleration as a service, we measure the performance and compare against the default CPU-only workflow in ProtoDUNE.

First, we measure the performance of the GPU server using standardized tools provided by NVidia to baseline the expected throughput of the GPU server for our model. Then we measure the latency for a single client side job within the LArSoft software framework including GPU acceleration as a service and in the baseline CPU-only workflow. Finally, we scale out the workflow to have multiple CPU clients using the GPU server and develop a model for scaling out to large workloads.

3.1 Server side performance

- describe perf client
- make table of measured throughputs
- describe basic features



ailab01	Simultaneous nodes	Concurrency	1	2	4	8	12	16	20
1 GPU	1		9733	12333	12333	12466			
	2		12332	12398	11999				
GKE									
1 GPU	1		10599	22999	18866	19133		19199	
	2								
	2		9099	16966	21732.5	22666		21599	
	3				29466	31999			
2 GPU	4						37331.5		38332
	1		2883.25	5099.75	5849.75	4933.25		5599.75	
	2		5099.75	10649.75	11066.25	6533	11799.5		
4 GPU	4		11532.75			20816.25			23166

Figure 4. Processing comparison

3.2 Single client measurements (Mike)

Tests of network latency and inference latency.

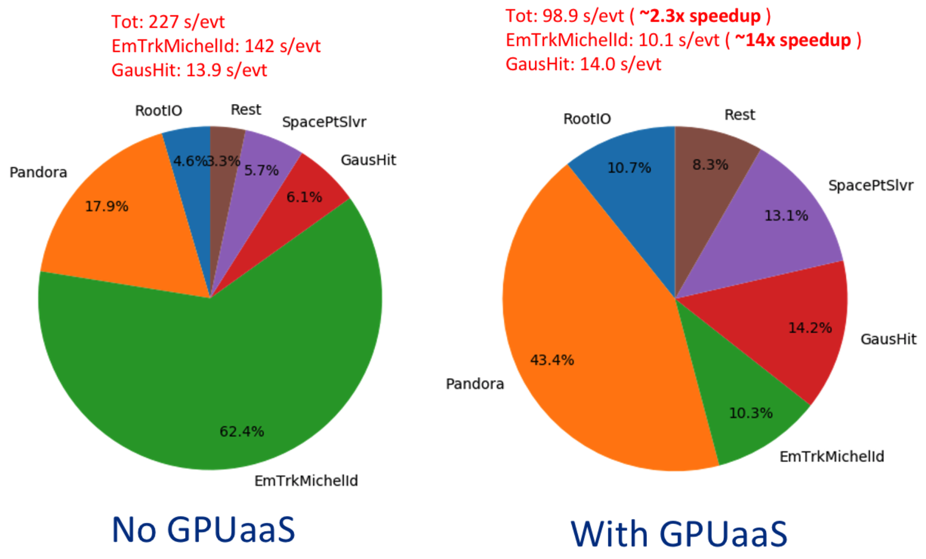


Figure 5. Processing comparison

3.3 Multiclient scaling (Mike)

3.3.1 Simplified model of throughput

We assume blocking modules and zero communication latency to start with. We define p as the fraction of the event which can be accelerated such that the total time of a CPU-only job is trivially defined as:

$$t^{\text{tot}}_{\text{CPU}} = (1-p) \times t^{\text{tot}}_{\text{CPU}} + p \times t^{\text{tot}}_{\text{CPU}} \tag{3.1}$$

We replace the ”accelerate-able” module by the latency on the GPU:

$$t^{\text{tot}}_{\text{SONIC,ideal}} = (1-p) \times t^{\text{tot}}_{\text{CPU}} + t_{\text{GPU}} \tag{3.2}$$

This is in the ideal scenario when the GPU is always available for the CPU job. The value of t_{GPU} is fixed *unless* the GPU is saturated with requests. We define this condition as how many GPU requests can be made while a single CPU is processing an event. So the GPU saturation level conditions is defined as:

$$\frac{N_{\text{CPU}}}{N_{\text{GPU}}} > \frac{t^{\text{tot}}_{\text{SONIC,ideal}}}{t_{\text{GPU}}} \tag{3.3}$$

Here, $t^{\text{tot}}_{\text{SONIC,ideal}}$ is equivalent to Eq. A.2 and is the processing time assuming there is no saturated GPU. If we consider now two conditions, saturated and unsaturated GPU, we can compute the total latency in each case:

$$t^{\text{tot}}_{\text{SONIC}} = (1-p) \times t^{\text{tot}}_{\text{CPU}} + t_{\text{GPU}}, \text{ if } \frac{N_{\text{CPU}}}{N_{\text{GPU}}} < \frac{t^{\text{tot}}_{\text{SONIC,ideal}}}{t_{\text{GPU}}} \tag{3.4}$$

$$t^{\text{tot}}_{\text{SONIC}} = (1-p) \times t^{\text{tot}}_{\text{CPU}} + t_{\text{GPU}} \left[1 + \left(\frac{N_{\text{CPU}}}{N_{\text{GPU}}} - \frac{t^{\text{tot}}_{\text{SONIC,ideal}}}{t_{\text{GPU}}}\right)\right], \text{ if } \frac{N_{\text{CPU}}}{N_{\text{GPU}}} > \frac{t^{\text{tot}}_{\text{SONIC,ideal}}}{t_{\text{GPU}}} \tag{3.5}$$

Plugging in Eq. A.2, you can rewrite the above as:

$$t^{\text{tot}}_{\text{SONIC}} = (1-p) \times t^{\text{tot}}_{\text{CPU}} + t_{\text{GPU}}, \text{ if } \frac{N_{\text{CPU}}}{N_{\text{GPU}}} < \frac{t^{\text{tot}}_{\text{SONIC,ideal}}}{t_{\text{GPU}}} \tag{3.6}$$

$$t^{\text{tot}}_{\text{SONIC}} = t_{\text{GPU}} \times \frac{N_{\text{CPU}}}{N_{\text{GPU}}}, \text{ if } \frac{N_{\text{CPU}}}{N_{\text{GPU}}} > \frac{t^{\text{tot}}_{\text{SONIC,ideal}}}{t_{\text{GPU}}} \tag{3.7}$$

3.4 Validation

So, now let’s plug in some numbers for the protoDUNE scenario. The total CPU time with no GPU is $t^{\text{tot}}_{\text{CPU}} = 330\ s$. The accelerate-able model is `EMMichelTrackID` such that $p = 0.67$ and $p \times t^{\text{tot}}_{\text{CPU}} = 220\ s$. When we run the module on the GPU, we find that $t_{\text{GPU}} = 10\ s$. Plugging these values into Eq. A.2

$$t^{\text{tot}}_{\text{SONIC,ideal}} = 120\text{s} \tag{3.8}$$

If we consider both conditions, we find:

$$t^{\text{tot}}_{\text{SONIC}} = 120\text{s, if } \frac{N_{\text{CPU}}}{N_{\text{GPU}}} < 12 \tag{3.9}$$

$$t^{\text{tot}}_{\text{SONIC}} = 110 \text{ s} + 10 \text{ s} \left[1 + \left(\frac{N_{\text{CPU}}}{N_{\text{GPU}}} - 12\right)\right] \text{, if } \frac{N_{\text{CPU}}}{N_{\text{GPU}}} > 12 \tag{3.10}$$

For example, if $N_{\text{CPU}} = 32$ and $N_{\text{GPU}} = 1$, then $t^{\text{tot}}_{\text{SONIC}} = 320s$, but that's not what we see. Even in the single GPU scenario, this already does not seem to scale. Why??

4 Summary and Outlook (Nhan)

On HPC?

Acknowledgments

Acknowledgments

A Throughput and Latency Calculations

We assume blocking modules and zero communication latency to start with. We define p as the fraction of the event which can be accelerated such that the total time of a CPU-only job is trivially defined as:

$$t_{\mathrm{CPU}} = (1-p) \times t_{\mathrm{CPU}} + p \times t_{\mathrm{CPU}} \tag{A.1}$$

We replace the "accelerate-able" module by the latency on the GPU:

$$t_{\mathrm{ideal}} = (1-p) \times t_{\mathrm{CPU}} + t_{\mathrm{GPU}} \tag{A.2}$$

This is in the ideal scenario when the GPU is always available for the CPU job. The value of t_{GPU} is fixed *unless* the GPU is saturated with requests. We define this condition as how many GPU requests can be made while a single CPU is processing an event. So the GPU saturation level conditions is defined as:

$$\frac{N_{\mathrm{CPU}}}{N_{\mathrm{GPU}}} > \frac{t_{\mathrm{ideal}}}{t_{\mathrm{GPU}}} \tag{A.3}$$

Here, t_{ideal} is equivalent to Eq. (A.2) and is the processing time assuming there is no saturated GPU. There are two conditions, unsaturated and saturated GPU, which correspond to $\frac{N_{\mathrm{CPU}}}{N_{\mathrm{GPU}}} < \frac{t_{\mathrm{ideal}}}{t_{\mathrm{GPU}}}$ and $\frac{N_{\mathrm{CPU}}}{N_{\mathrm{GPU}}} > \frac{t_{\mathrm{ideal}}}{t_{\mathrm{GPU}}}$, respectively. We can compute the total latency to account for both cases:

$$t_{\mathrm{SONIC}} = (1-p) \times t_{\mathrm{CPU}} + t_{\mathrm{GPU}} \left[1 + \max\left(0, \frac{N_{\mathrm{CPU}}}{N_{\mathrm{GPU}}} - \frac{t_{\mathrm{ideal}}}{t_{\mathrm{GPU}}}\right)\right] \tag{A.4}$$

A small diagram illustrating the additional latency from a saturated GPU in Eq. (A.8) is given in Fig. 6. For the case of $N_{\mathrm{CPU}} = 5$ and $N_{\mathrm{GPU}} = 2$, we show that the latency for CPU1 is either 2 or 3 clocks. This can, for example, explain where there is such a spread in the total time for a given job to finish.

Plugging in Eq. (A.2) for t_{ideal}, the saturated case simplifies to:

$$t_{\mathrm{SONIC}} = t_{\mathrm{GPU}} \times \frac{N_{\mathrm{CPU}}}{N_{\mathrm{GPU}}} \tag{A.5}$$

A.1 Validation

So, now let's plug in some numbers for the protoDUNE scenario. The total CPU time with no GPU is $t_{\mathrm{CPU}} = 330\ s$. The accelerate-able model is `EMMichelTrackID` such that $p = 0.67$ and $p \times t_{\mathrm{CPU}} = 220\ s$. When we run the module on the GPU, we find that $t_{\mathrm{GPU}} = 10\ s$. Plugging these values into Eq. (A.2)

$$t_{\mathrm{ideal}} = 120\mathrm{s} \tag{A.6}$$

If we consider both conditions, we find:

$$t_{\mathrm{SONIC}} = 120\mathrm{s}, \text{ if } \frac{N_{\mathrm{CPU}}}{N_{\mathrm{GPU}}} < 12 \tag{A.7}$$

$$t_{\mathrm{SONIC}} = 110\ \mathrm{s} + 10\ \mathrm{s}\left[1 + \left(\frac{N_{\mathrm{CPU}}}{N_{\mathrm{GPU}}} - 12\right)\right], \text{ if } \frac{N_{\mathrm{CPU}}}{N_{\mathrm{GPU}}} > 12 \tag{A.8}$$

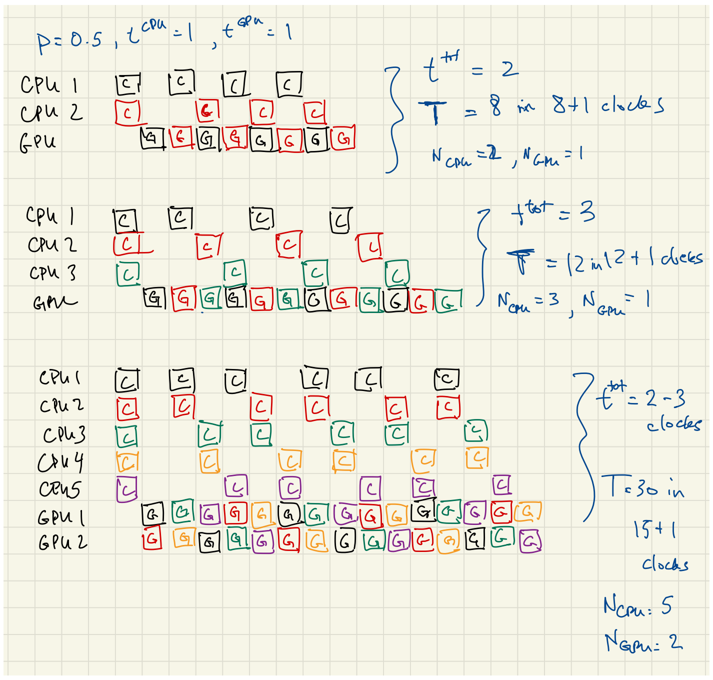


Figure 6. Concurrency mapping for $p = 0.5$ and $(1 - p) \times t_{\text{CPU}} = 1$ and $t_{\text{GPU}} = 1$

For example, if $N_{\text{CPU}} = 32$ and $N_{\text{GPU}} = 1$, then $t_{\text{SONIC}} = 320s$, but that's not what we see. Even in the single GPU scenario, this already does not seem to scale. Why??

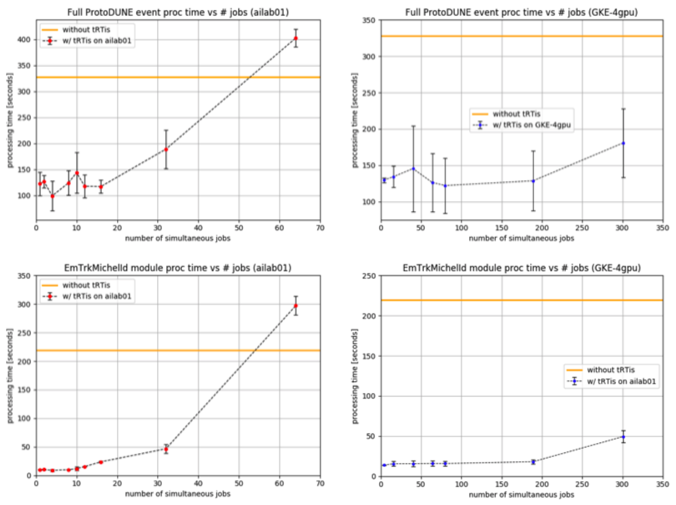


Figure 7. Results from tests by Mike/Maria