Introduction

• Computing projections outpace CPU growth, interest in ML and high latency algorithms only increasing

- Coprocessors (GPU, FPGA, …) offer possible solution

- Speedups at large:
  - Batch size and/or complexity
Direct Connect

• Simplest form of coprocessor implementation is *direct connect*

• Scaling this up can be difficult

  • What if I have multiple hardware types, algorithms, many CPUs (cluster)?

• User CPU needs some knowledge of coprocessor details
On-Demand Computing

• As a user, I just want my workflow to run quickly

• On-demand computing
  • Client communicates with server CPU, server CPU communicates with coprocessor
  • Many existing tools available from industry, cloud
As-a-service Computing

...as-a-service (aaS)

Pros:
- Simple support for mixed hardware
- Scaleable
- Throughput optimizations for multiple-client
- Simple client-side

...directly

Pros:
- Simple connections
- Reduced network load
As-a-service Computing

• In principle, as-a-service can be used for any algorithm

• Simply send all inputs to server, server returns outputs

• Just need server able to accept requests and communicate with GPU or FPGA

• Can provide large speed up w.r.t traditional computing model
MLaaS

- Many tools from industry focus on machine learning inference
  - Small number of inputs relative to large number of operations and CPU compute time
  - No knowledge of hardware language (HLS/RTL/CUDA) required
- Growing component of computing workflows
- CMS is exploring ML alternatives to many traditional algorithms
  - FACILE (HCAL reco)
  - PF, MET, clustering, tagging
HCAL energy regression currently performed by MAHI algorithm (fit for in-time pulse energy)

Good candidate for ML solution

Same inputs as MAHI, no corrections/calibrations necessary

Dense network, ~2000 parameters (small)

Similar performance to MAHI

Working with CMS HCAL DPG to validate/integrate
Setup

- For fast inference we focus on gRPC protocol
- Open source remote procedure call (RPC) system developed by Google

1. Runs the inference
Setup

- For fast inference we focus on gRPC protocol
  - Open source remote procedure call (RPC) system developed by Google

1. Formats inputs
2. Sends asynchronous, non-blocking gRPC call (CMSSW ExternalWork)
3. Interprets response
Setup

- For fast inference we focus on gRPC protocol
  - Open source remote procedure call (RPC) system developed by Google

1. Formats inputs
2. Sends asynchronous, non-blocking gRPC call (CMSSW ExternalWork)
3. Interprets response

1. Initializes model on coprocessor
2. Receives and schedules inference request
3. Sends inference request to accelerator
4. Outputs and send results
5. Monitors network/device utilization

Tools (backup)

- Use NVidia triton inference server for GPU + Customized GCP Kubernetes
- Wrote our own FPGA gRPC inference server
SONIC Client

- SONIC (Services for Optimized Network Inference on Coprocessors)
  - Simple
  - Available in CMSSW
  - No knowledge of coprocessor specs needed

- CMSSW (core): https://github.com/cms-sw/cmssw/tree/master/HeterogeneousCore/SonicCore

- Development: https://github.com/hls-fpga-machine-learning/SonicCMS

- Nvidia triton external and client are also now available in CMSSW
  - Docker-ized test setup & example producer
SONIC Client

```cpp
#include "HeterogeneousCore/SonicCore/interface/SonicEDProducer.h"
#include "FWCore/Framework/interface/MakerMacros.h"

class MyProducer : public SonicEDProducer<Client> {
public:
  explicit MyProducer(edm::ParameterSet const& cfg) : SonicEDProducer<Client>(cfg) {
    // for debugging
    setDebugName("MyProducer");
  }
  void acquire(edm::Event const& iEvent, edm::EventSetup const& iSetup, Input& iInput) override {
    // convert event data to client input format
  }
  void produce(edm::Event& iEvent, edm::EventSetup const& iSetup, Output const& iOutput) override {
    // convert client output to event data format
  }
  static void fillDescriptions(edm::ConfigurationDescriptions & descriptions) {
    edm::ParameterSetDescription desc;
    Client::fillPSetDescription(desc);
    // add producer-specific parameters
    descriptions.add("MyProducer", desc);
  }
};

DEFINE_FWK_MODULE(MyProducer);
```

User writes a standard producer, server address included as a config

- Nvidia triton external and client are now available in CMSSW
- Docker-ized test setup & example producer

User writes a standard producer, server address included as a config
other very challenging, computationally intensive algorithms (including a directed graph CNN) are within a factor of a few in size with respect to the typical particle physics models used for top tagging are often several orders of magnitude smaller than those used for neutrino event classification. As such, it is customary to report three metrics for the performance of the network on the top tagging dataset: model accuracy, area under the ROC curve (AUC), and background rejection power at a fixed signal efficiency.

Fig. 3: A comparison of QCD (left) and top (right) jet images averaged over 5,000 jets.

Table 1: Comparison of QCD and top jets using Brainwave. The best performance is a model accuracy of 92.6%, an area under the ROC curve (AUC) of 98.2%, and a background rejection power at a fixed signal efficiency of 30%.

Fig. 4: The ROC curves showing the performance of different models on the top tagging dataset.

**Benchmarks**

**FACILE**

**DeepCalo**

**ResNet**

**ECAL cluster regression**

**(ATLAS algorithm)**
Benchmarks

- Test 3 different example ML algorithms
  - FACILE (batch 16000)
  - DeepCalo (batch 10)
  - ResNet (batch 10)

<table>
<thead>
<tr>
<th>GPU/FPGA aaS</th>
<th>Gain w.r.t. CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 ms (GPU)</td>
<td>8x (GPU)</td>
</tr>
<tr>
<td>0.2 ms (FPGA)</td>
<td>80x (FPGA)</td>
</tr>
<tr>
<td>0.1 ms (GPU) in progress (FPGA)</td>
<td>750x</td>
</tr>
<tr>
<td>1-2 ms (GPU/FPGA)</td>
<td>500x</td>
</tr>
</tbody>
</table>
Gains

Where do we gain from coprocessors?

- FACILE
- DeepCalo
- ResNet

Batch size/network bandwidth vs. Algorithm complexity

Small gain vs. Large gain
1 GPU Server

- Inference performed in CMS workflow
- Larger models saturate with fewer clients, lower throughput
- Range of performance for GPUs

![Graphs showing throughput vs. simultaneous processes for FACILE, DeepCalo, and ResNet models with different batch sizes.](image)
Multi-GPU Server

- High bandwidth, long distance (MIT and Google Cloud US-central)
- Linear scaling with # of GPUs
- Throughput saturates at ~60 Gbps (8000 events/s)
Dynamic Batching

- Allows server to wait for requests to build up
- Most beneficial for small-batch algorithms
- Can extend event-by-event processing to multi-event processing
  - Transparent to user
- Single-line change to server configuration

```
dynamic_batching {
  preferred_batch_size: [ 100 ]
}
```

Can also specify max wait time
Dynamic Batching

- 60x throughput gain in this case
- 10k events/s for 1M weight model
- ~1000 simultaneous clients to saturate single GPU
FPGA Server Design

- Same workflow developed for FPGA coprocessors
  - gRPC base (Triton calls), same config as for running on GPU
- **FACILE**: hls4ml (Alveo U250 & AWS f1)
- **DeepCalo**: hls4ml (ongoing work)
- **ResNet**: Xilinx ML Suite (AWS f1)
- **ResNet**: Microsoft Azure ML Studio (Azure Stack Edge)
- Many design settings to optimize
FPGA Server

- With small FACILE network, major speedup w.r.t. GPU (500 evt/s)
- Limitation from CPU
- For larger ResNet, comparable or slightly better throughput w.r.t GPU
• Used FACILE in CMS HLT workflow to test as-a-service model in realistic computing environment

  • Replaced current (non-ML) MAHI algo

• Use of cloud resources allows at-scale test
Coprocessor Scalability

- 10% reduction in computing time operating as-a-service

  - Consistent with fraction of time spent on HCAL local reco w.r.t total HLT time
  
  → Maximal achievable reduction for this single algorithm

- No increase in latency until 300/1500 clients (GPU/FPGA)

  - Single device can service 300/1500 HLT instances
Coprocessor Scalability

- Factor of 5 improvement between FPGA over GPU for HLT less than >10x shown earlier
- Running on AWS, network bandwidth is limited to 25 Gbps
- Corresponds to a maximal throughput of ~2500 events/s
- Consistent with HLT saturation at 1500 processes
aaS Configuration

- As-a-service paradigm opens up many design options
  - Servers for each algorithm
  - Server capable of managing multiple algorithms
  - Mixed architecture servers
Summary

• As-a-service computing has many existing tools that we can leverage to address computing challenges
  • Very cohesive with ML usage

• Extremely simple for end user

• Papers detailing GPUaaS (2007.10359) and FPGAaaS (submitted to H²RC’20)

• Award from internet2: https://www.internet2.edu/news/detail/17957/

• Many more possibilities for improvement
Next Steps

- **At-scale testing:**
  - Tests of full HLT, more accelerated algorithms
    - Many possibilities for speed-ups with heterogeneous arch
  - Offline production workflow at scale
  - Tests with DOE HPCs
    - Perhaps more realistic for HEP usage

- **Algorithm development:**
  - Clustering with ML (full ML chain from detector to physics objects)
BACKUP
As-a-service Computing

- GPU-to-CPU replacement ratio:
  \[ F^\text{eq}_{\text{GPU}} = \frac{X - S - L}{Y} \]

- How many clients can one GPU serve while maintaining throughput?

\[ X: \text{CPU algo time} \]
\[ S: \text{I/O packaging overhead} \]
\[ Y: \text{GPU algo time} \]
\[ T: \text{transfer time} \]
\[ L: \text{rescheduling time } [f(Y+T)] \]
8 GPU/FPGA Server

- Similar performance between GPU and FPGA
  - ~150 evt/s
Tools

Our tools for prototyping CMS reconstruction as-a-service
1. Google Cloud/Amazon Web Services/Microsoft Azure
2. T2/T3 clusters
3. local server/accelerator hardware

Towards abstraction:
on-premises, in the cloud, oh my!

Building a network of heterogeneous resources in the cloud and on-premises
Work-in-progress: how to coordinate and orchestrate distributed heterogeneous resources

We have a wide network of resources, and perform at-scale tests with many different client-servers configurations, with servers both remote and on-site
Triton Inference Server

Client sends request over network

Server receives request

Server queues and schedules request

The number of connected GPUs/FPGAs is scaleable; each has an instance of each model

Models are stored in local repository

Many model formats (TensorFlow, Pytorch, TensorRT, …)

Output monitoring information
Scalability

A client-server schematic

How many clients can a GPU service? What is the throughput?

What is the network limit? How reliable is it?

How does the throughput scale with server size?
Network Limit

- Server-on-site: no bandwidth limit found
- Remote server: egress limit at 70 Gb/s for MIT T2
- Exceeds needs for use cases considered