Integration of ML Frameworks in CMSSW

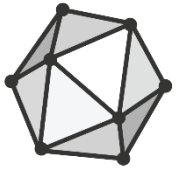
Kevin Pedro (FNAL)

December 4, 2020

CMSSW Basics

- Statistics:
  - Core codebase: ~6 million lines of code
    - Mostly C++ and Python
    - 101 subsystems, 1276 packages
  - O(500) externals (outside software distributed with our releases)
  - 60–100 active contributors on GitHub (open source)
- Dedicated build system SCRAM
  - Recently migrated from Perl to Python!
- CMSSW uses task-based multithreading via Intel TBB
  - All code must be thread-safe!
  - In order of preference:
    1. Use framework-supported thread-safe module (global, stream, one)
    2. Use tbb or atomic operations
    3. Use locks (least efficient)

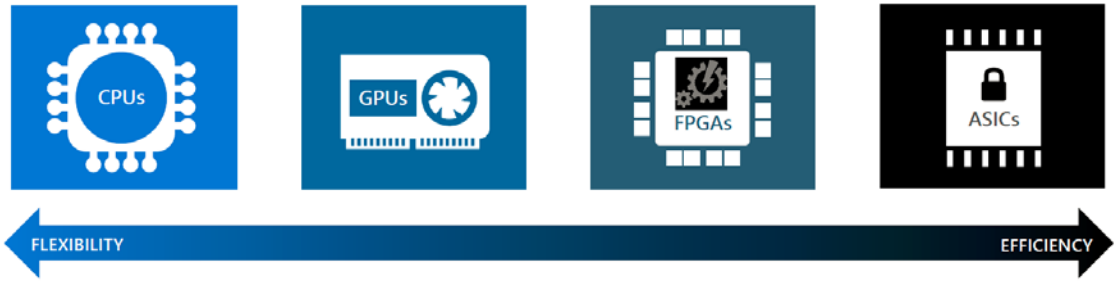
Specific ML Frameworks



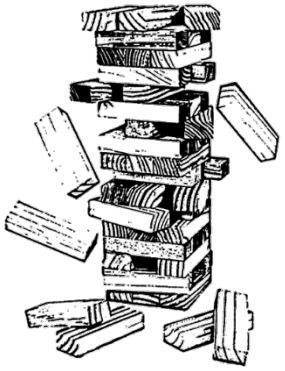
- CMSSW currently includes C++ APIs (as externals) for:
  - TensorFlow, ONNX, MXNet, LWTNN, TMVA (in ROOT)
  - Ongoing work to add PyTorch
- Maintenance burden for these externals is high:
  - Difficult build tools (especially Bazel)
  - Complex and inflexible build configurations
    - Takes significant work to use existing versions of external dependencies instead of reinstalling everything from scratch
  - Thread-safety, etc. problems are common
    - Even if problems are fixed (e.g. after a week of repeated helgrind runs), external maintainers may not accept contributed solution

Specific Frameworks + Coprocessors

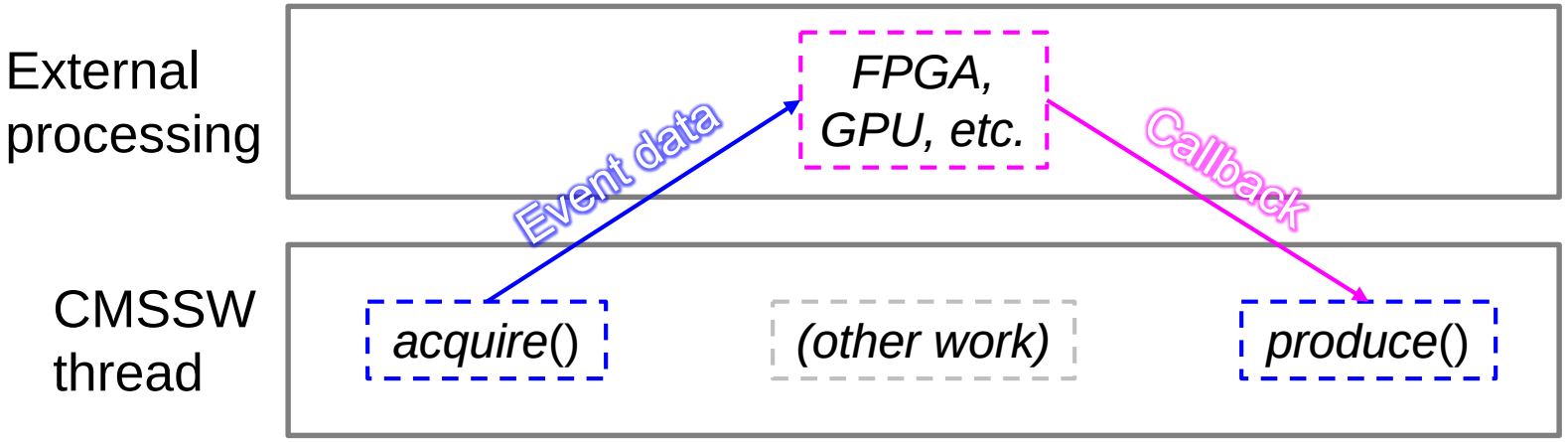
- The C++ APIs can all be used to run inference on CPU
  - Significant work in performance optimization, e.g.:
    - Use OpenBLAS rather than GSL CBLAS
    - Enable multi-vectorization to support multiple instruction sets
      - AVX still not available on all grid CPUs
    - Reduce memory usage by sharing some objects
  - Some APIs are still slow (TMVA)
    - CMSSW contains a custom BDT reader, ~5× faster
- Currently no support direct inference via any ML framework on GPU etc.
  - Very hard to keep consistent and compatible CUDA version (for example)
    - Need to consider C++ compiler versions used for CMSSW
    - Different ML frameworks (and other CUDA-dependent code) may require different CUDA versions



General Approach: Direct Connect

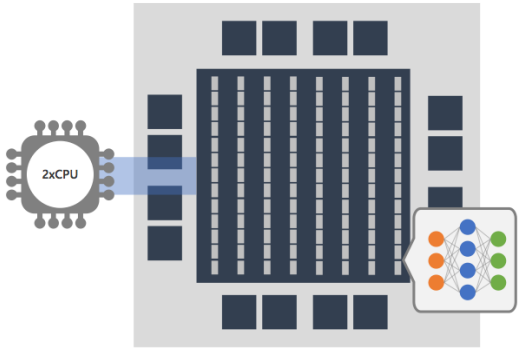


- CMS can offload to directly-connected GPUs
  - Uses ExternalWork feature (below): asynchronous, non-blocking task-based processing
- This approach will be used for Run 3 HLT
  - Classical reco algorithms rewritten in CUDA (Patatrack)
- In principle, standalone CUDA kernels could be generated for specific ML algorithms to utilize Nvidia GPUs
  - Has not been attempted in practice; maintenance burden seems high
- Exploring compatibility libraries (e.g. Kokkos) to write code that can be compiled for multiple devices
  - Not clear if useful for ML



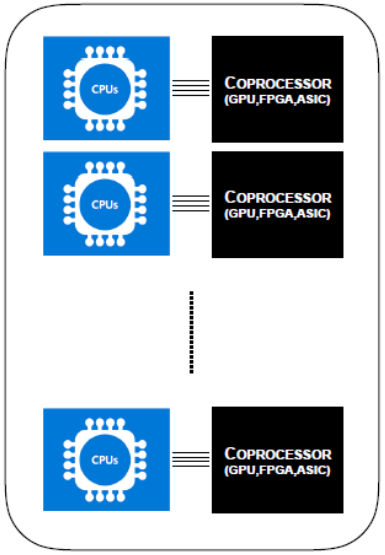
Interlude: FPGAs

- hls4ml can translate BDTs, DNNs, etc. into FPGA instructions
  - Planned for use in CMS L1 trigger for Run 3
- Other avenues have also been explored:
  - Xilinx ML Suite, Microsoft Brainwave, Intel OneAPI, …
- These tools are currently treated separately from CMSSW
- Possible to use similar direct connect + ExternalWork setup to access FPGAs from CMSSW, e.g. via OpenCL
  - Has not been pursued so far

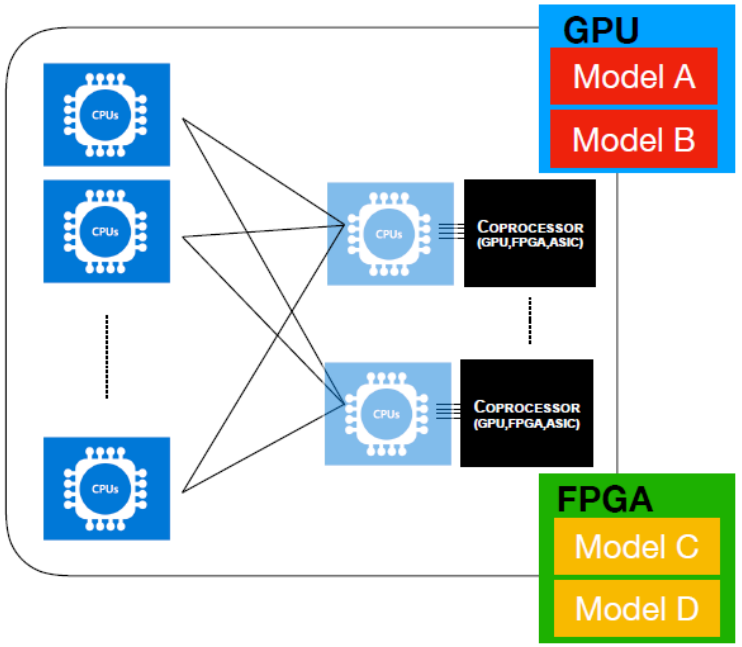


General Approach: Inference as a Service

Direct



as a Service



- Multiple CPUs send inference requests to coprocessor server
- Ensures optimal utilization of GPUs/FPGAs, along with flexibility
- One coprocessor could serve ~100 CPUs
  - Depending on conditions and requirements
  - Much more cost effective than buying 1 GPU for every grid CPU…

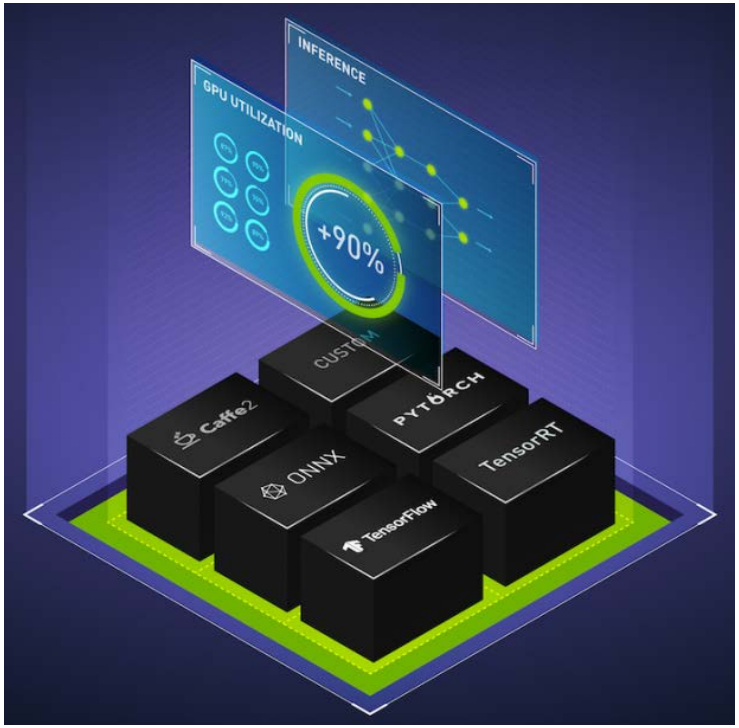
SONIC Approach

- SONIC (Services for Optimized Network Inference on Coprocessors): inference as a service in experiment software frameworks

- ✓ Use industry tools:
  - gRPC, inference servers, Kubernetes
    - Interact with cloud services: Azure, AWS, GCP
- ✓ Minimize rewriting and reimplementing in C++
  - CMSSW modules just convert input and output data into desired formats
  - Supports producers and filters, as well as analyzers (for trees or histos)
- ✓ Available in CMSSW
  - Utilizes ExternalWork for asynchronous, non-blocking calls
    - Minimize impact of latency
  - Development aimed at production-readiness for Run 3
- Currently focused on Nvidia Triton inference server
  - Previously explored tensorflow/serving

Triton Inference Server

- Supports all frameworks as backends (even non-ML)
  - Reduce maintenance concerns by keeping these separate from CMSSW
- Many killer features:
  - Dynamic batching (GPU throughput ↑)
    - Enables multi-event processing w/o any CMSSW framework changes
  - Load balancing (for multi-GPU server)
  - TensorRT optimization: reduced precision, layer fusion, etc.
  - Shared memory (for local GPU), CPU fallback
- Developed FPGA as a Service Toolkit (FaaST)
  - Open source, interoperable w/ Triton gRPC calls



- Upcoming features:
  - I/O compression
  - “Model analyzer” to optimize server deployment

Conclusions

- CMS currently devotes non-trivial effort to integrate and maintain C++ APIs for O(6) different ML frameworks
- Even more daunting: integrating coprocessor-specific features of ML fwks
- Direct connect tooling available for Nvidia GPUs via CUDA
  - Not utilized for ML so far

- Inference as a service offers best path forward for ML inference
  - Simplicity: lower maintenance, separate ML frameworks from CMSSW
  - Flexibility: use multiple types of coprocessors, server configurations, etc.
  - Cost: 1 coprocessor can serve many CPUs
  - Optimizations: easy to deploy custom instructions, reduced precision, dynamic batching, etc.

Backup