Jets/MET with pileup and machine learning

Nhan Tran
Fermilab

Princeton HL-LHC Trigger Workshop
January 16, 2018
Last talk, Giovanni:

**Particle Flow** - efficient combination of detector information to extract best physics performance

*Building of the technology presented by Giovanni…*

**This talk: more advanced algorithms**

- Dealing with pileup
- PUPPI proof-of-concept: jets, MET, jet substructure (?),…
- More sophistication with machine learning and HLS4ML
Multiple pp collisions in the same beam crossing
To increase data rate, squeeze beams as much as possible
Multiple pp collisions in the same beam crossing
To increase data rate, squeeze beams as much as possible

2016: $<\text{PU}> \sim 20-50$
2017 + Run 3: $<\text{PU}> \sim 50-80$
HL-LHC: 140-200

Need sophisticated techniques to preserve the physics!
PUPPI (PileUp Per Particle Id): based on PF paradigm

A general framework that determines, per particle, weight for how likely a particle is from PU

Key insight: using QCD ansatz to infer neutral pileup contribution

1. Define a local discriminant, $\alpha$, between pileup (PU) and leading vertex (LV)

$$\alpha_i^C = \log \left[ \sum_{j \in \text{Ch, LV}} \frac{p_{T,j}}{\Delta R_{ij}} \Theta(R_0 - \Delta R_{ij}) \right]$$

2. Get data-driven $\alpha$ distribution for PU using charged PU tracks
PUPPI (PileUp Per Particle Id): based on PF paradigm
a general framework that determines, per particle, weight for how likely a particle is from PU
key insight: using QCD ansatz to infer neutral pileup contribution

[1] define a local discriminant, $\alpha$, between pileup (PU) and leading vertex (LV)

$$\alpha^C_i = \log \left[ \sum_{j \in \text{Ch}, LV} \frac{p_{T,j}}{\Delta R_{ij}} \Theta(R_0 - \Delta R_{ij}) \right]$$

[2] get data-driven $\alpha$ distribution for PU using charged PU tracks

[3] for the neutrals, ask “how un-PU-like is $\alpha$ for this particle?”, compute a weight

[4] reweight the four-vector of the particle by this weight, then proceed to interpret the event as usual
PUPPI (PileUp Per Particle Id): based on PF paradigm
a general framework that determines, per particle, weight for how likely a particle is from PU

key insight: using QCD ansatz to infer neutral pileup contribution
Large gains from PUPPI, especially at high PU

arXiv:1407.6013 (original), LHCC-P-008 (CMS TP), JME-14-001, CMS analyses,...

many gains at high PU
jet pT resolution
fake jet rate
MET resolution
jet substructure
lepton isolation

...
BEYOND JETS AND MET

Trying to preserve soft, hidden physics
Things hidden in jets and jet substructure
Isolated, soft leptons in high PU environments

* Examples plots from offline studies

Large gains in soft muon backgrounds and jet substructure
IMPLEMENTATION

Implementation of Puppi proof-of-concept using High level synthesis (HLS) as well

[1] define a local discriminant, $\alpha$, between pileup (PU) and leading vertex (LV)

$$\alpha_i^C = \log \left[ \sum_{j \in C_{h,LV}} \frac{p_{T,j}}{\Delta R_{ij}} \Theta(R_0 - \Delta R_{ij}) \right]$$
**IMPLEMENTATION**

Implementation of Puppi proof-of-concept using High level synthesis (HLS) as well

1. Define a local discriminant, $\alpha$, between pileup (PU) and leading vertex (LV)

$$\alpha_i^C = \log \left( \sum_{j \in \text{Ch}, LV} \frac{pT,j}{\Delta R_{ij}} \Theta(R_0 - \Delta R_{ij}) \right)$$

2. Get data-driven $\alpha$ distribution for PU using charged PU tracks

3. For the neutrals, ask “how un-PU-like is $\alpha$ for this particle?”, compute a weight

4. Reweight the four-vector of the particle by this weight, then proceed to interpret the event as usual

**Compute for each neutral**

**Precompute step 2 offline with constants (for given pileup level)**

**Do step 3/4 with a look-up table**

**Resource usage only few % of FPGA and 100s of ns latency with little degradation in performance**
First physics results on HT and MET triggers for CMS phase-2 trigger interim document

Gains in rate reduction, signal efficiency, lower thresholds
Bringing advanced physics algorithms to the hardware trigger!
Proof-of-concept PF+PUPPI running at L1
large physics gains: HT, MET, jet (substructure), lepton isolation

Other advanced algorithms…
how about machine learning?
high level synthesis for machine learning

HLSFML

HLS4ML

Jennifer Ngadiuba, Maurizio Pierini (CERN)
Javier Duarte, Sergio Jindariani, Ben Kreis, Nhan Tran (FNAL)
Phil Harris (MIT)
Zhenbin Wu (UIC)

+ EJ Kreinar (Hawkeye 360) and Song Han (Google/Stanford)
Many parts of the trigger could benefit machine learning clustering, fitting (regression), classification, anomaly detection

Not just LHC physics or triggering

DAQ, neutrino physics, intensity frontier, …

No industry solutions: LHC latency constraints are unheard of

Why HLS?

HLS allows (super)-fast algorithm development

Write a tool for machine learning inference* at low latencies: HLS4ML

*for training, GPUs remain top dog
Simple 2 input example
*(Fisher linear discriminant, linear support vector machine, ...)*

\[ O_j = l_i \times W_{ij} + b_j \]

\[ O_1 = l_1 \times W_{11} + l_2 \times W_{21} + b_1 \]
NN INFEERENCE IN A NUTSHELL

\[ \vec{O}_j = \Phi (\vec{l}_i \times \vec{W}_{ij} + \vec{b}_j) \]

\(\Phi = \text{ACTIVATION FUNCTION} \hspace{1cm} \text{(NON-LINEARITY)}\)

INPUTS

HIDDEN LAYERS

FULLY CONNECTED HIDDEN LAYER

OUTPUTS
**NN INference in a NUTShEEL**

\[ \vec{O}_j = \Phi(l_i \times \vec{W}_{ij} + b_j) \]

\( \Phi = \text{ACTIVATION FUNCTION} \quad \text{(NON-LINEARITY)} \)

**NN inference =**

a bunch of multiplications /additions and LUTs (look up tables) for activation functions

**FULLY CONNECTED HIDDEN LAYER**
Emergent engineering field, efficient implementation of NN architecture

**Compression/Pruning:**

maintain the same performance while removing low weight synapses and neurons (many schemes)

**Quantization/Approximate math:**

32-bit floating point math is overkill

20-bit, 18-bit, …? fixed point, integers? binarized NNs?

![Diagram of pruning process](image)

For further reading, start here: [https://arxiv.org/pdf/1510.00149v5.pdf](https://arxiv.org/pdf/1510.00149v5.pdf)
PROJECT OVERVIEW

- Model
- Compressed model
- HLS conversion
- HLS project
- Tune configuration
  - Precision
  - Reuse/latency
- Co-processors
- RTL design
Usual software workflow

Industry co-processors

L1 application: custom firmware

HLS4ML
HLS4ML - TRANSLATION IN ONE LINE!

```
python keras-to-hls.py -c keras-config.yml
```

```
1  KerasJson: example-keras-model-files/KERAS_1layer.json
2  KerasH5:  example-keras-model-files/KERAS_1layer_weights.h5
3  OutputDir: my-hls-dir-test
4
5  IOType: io_parallel  # options: io_serial/io_parallel
6  ReuseFactor: 1
7  DefaultPrecision: ap_fixed<18,8>
```

**IOType:** parallelize or serialize  
**ReuseFactor:** how much to parallelize  
**DefaultPrecision:** self-explanatory :)
EXAMPLE: JET SUBSTRUCTURE

5 output multi-classifier:
Does a jet originate from a quark, gluon, W/Z boson, top quark?

Network architecture
16 expert inputs
  jet masses, multiplicity
  ECFs (β=0,1,2)

<table>
<thead>
<tr>
<th>16 inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>64 (relu)</td>
</tr>
<tr>
<td>32 (relu)</td>
</tr>
<tr>
<td>32 (relu)</td>
</tr>
<tr>
<td>5 outputs (softmax)</td>
</tr>
</tbody>
</table>

16 inputs
64 (relu)
32 (relu)
32 (relu)
5 outputs (softmax)

HLS4ML Work in Progress
**Example: Network (not jet) Pruning**

Resource usage:
92% DSP usage for Virtex 7
61 clocks (305 ns), Pipeline = 1

**Compression (50%) + reuse = 2:**
29% DSP usage for Virtex 7
60 clocks (300 ns), Pipeline = 2

```
16 inputs
64 (relu)
32 (relu)
32 (relu)
5 outputs (softmax)
```

**HLS4ML Work in Progress**

![Graph showing performance metrics](image-url)
**MINI-SUMMARY**

**HLS4ML**

a tool to translate ML algorithms for FPGAs in minutes
highly parallelizable with user controls for resource usage and latency
tunable precision, resource reuse
very efficient network design with model compression

---

**Work in progress**

Mapping out resource usage and latency as a function of neural network hyper parameters
More network architectures: CNN (in progress), RNN/LSTM (tricky!), TMVA BDT (efficient?)

**Status**

Alpha version - few weeks; Targeting March-April release of Beta version
Please contact us if you are interested! hls4ml.help@gmail.com
one more fun thing to think about for the high level trigger (and beyond?)
Specialized co-processor hardware for machine learning inference

Catapult/Brainwave

Delivering FPGA Partner Solutions on AWS via AWS Marketplace

INTEL® FPGA ACCELERATION HUB

The Intel® Xeon® Acceleration Stack for FPGAs is a robust framework enabling data center applications to leverage an FPGA’s potential to increase

AFI is secured, encrypted, dynamically loaded into the FPGA - can’t be copied or downloaded
Specialized co-processor hardware for machine learning inference

It already exists!
One example: Microsoft catapult

Translation of all of wikipedia in 0.1 seconds!
\(\sim O(100)\) times faster than CPU
Large gains from hardware accelerating co-processors
Industry trending towards specialized computing paradigms

**Option 1**
re-write physics algorithms for new hardware
Language: OpenCL, OpenMP, HLS, ...
Hardware: FPGA, GPU

**Option 2**
re-cast physics problem as a machine learning problem
Language: C++, Python (TensorFlow, PyTorch,...)
Hardware: FPGA, GPU, ASIC

**Why (Deep) Machine Learning?**
a common *language* for solving problems which can universally be expressed on optimized computing hardware and follow industry trends
Summary and Outlook

Recent advances in hardware and compilation/synthesis allow for sophisticated techniques at low latency

Big improvements in performance, preserve soft and hidden signatures

Proof-of-concept holistic pileup mitigation techniques such as PUPPI
Efficient machine learning at Level-1 Trigger
New paradigms for HLT and offline?
BONUS
**EXAMPLE: PARALLELIZATION**

**ReuseFactor**: how much to parallelize operations a hidden layer

**# of multiplications per clock (DSPs usage)**

- **parallel**
  - 8DSPs in 1 clock
  - reuse = 1

- **parallel**
  - 4DSPs in 2 clocks
  - reuse = 2

- **serial**
  - 1 DSP in 8 clocks

(time) (decreasing throughput)
Example: Network (not jet) Pruning

Resource usage:
92% DSP usage for Virtex 7
61 clocks (305 ns), Pipeline = 1

Compression (50%) + reuse = 2:
29% DSP usage for Virtex 7
60 clocks (300 ns), Pipeline = 2

16 inputs
64 (relu)
32 (relu)
32 (relu)
5 (softmax)

Fully connected deep neural network
**Example: Quantization**

Take a simple 1-layer network and scan in input/weight precision. Reduced precision can greatly reduce resource usage. E.g., factor of 4 reduction with 18 instead of 32 bits with minimal loss in performance.
UNDER THE HOOD

<table>
<thead>
<tr>
<th></th>
<th>BRAM</th>
<th>DSP</th>
<th>FF</th>
<th>LUT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>3</td>
<td>3329</td>
<td>95924</td>
<td>8127</td>
</tr>
<tr>
<td>% Usage</td>
<td>~0</td>
<td>92</td>
<td>11</td>
<td>18</td>
</tr>
</tbody>
</table>

61 clocks (305 ns)

16 x 64

64 x 32

32 x 32

32 x 5

softmax (5)
THE COMPUTING CHALLENGE

**Current:**

~5 minutes per HL-LHC event

100 times the data… exabytes!

Major HLT and computing challenges going forward!
MOORE’S LAW AND DENVARD SCALING

Moore’s Law continues

Dennard Scaling fails

Moore’s Law continues

Dennard Scaling fails

Single threaded performance not improving

Circa ~2005: “The Era of Multicore”

→ Today: Transition to the “Era of Specialization”? (c.f. Doug Burger)
* GPUs still best option for training
* FPGAs generally much more power efficient