Fast inference of jet substructure classifiers with FPGAs

Zhenbin Wu
(University of Illinois at Chicago)

Machine Learning for Jet Physics
Nov. 15th, 2018

Personal

Jennifer Ngadiuba, Vladimir Loncar, Maurizio Pierini

Javier Duarte, Burt Holzman, Sergo Jindariani, Ben Kreis, Mia Liu, Kevin Pedro, Ryan Rivera, Nhan Tran, Aris Tsaris

HawkEye 360 Edward Kreinar

Imperial College London Sioni Summer

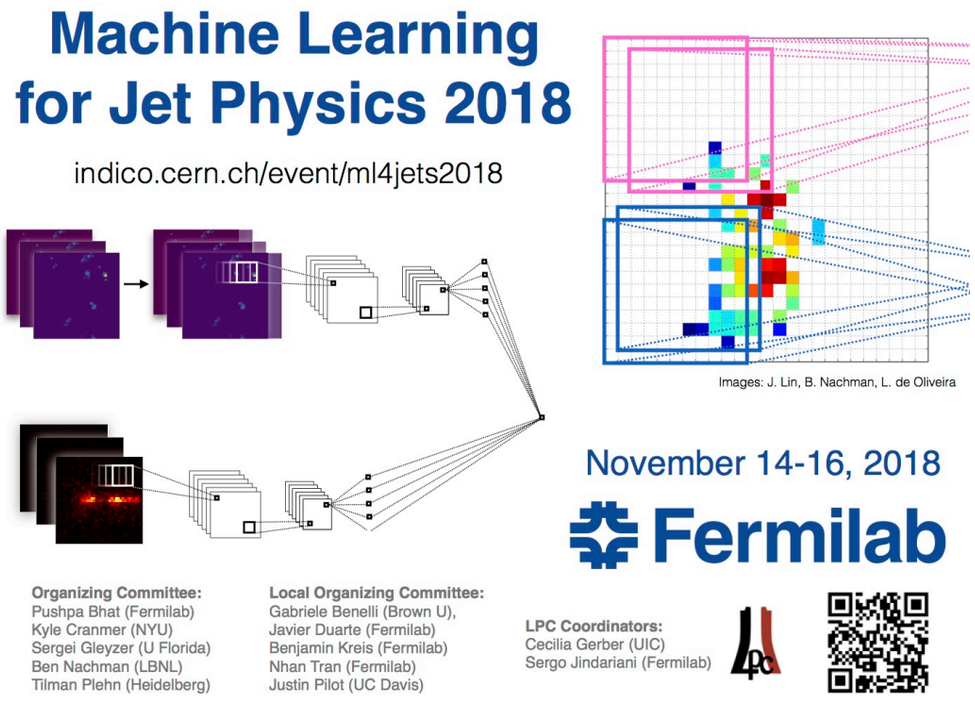
Song Han, Phil Harris, Dylan Rankin

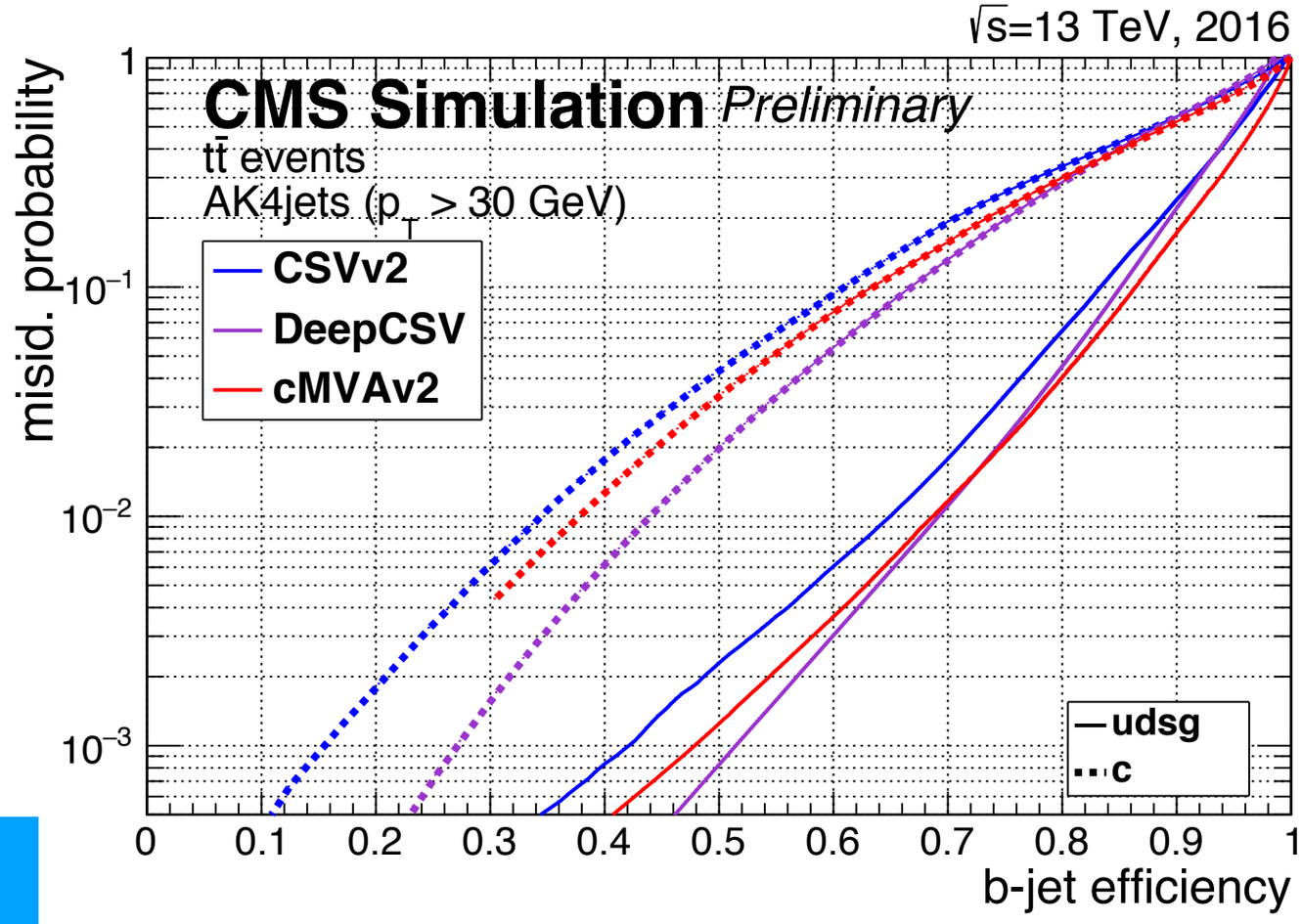
Zhenbin Wu

Mark Neubauer, Markus Atkinso

Machine Learning in Jets

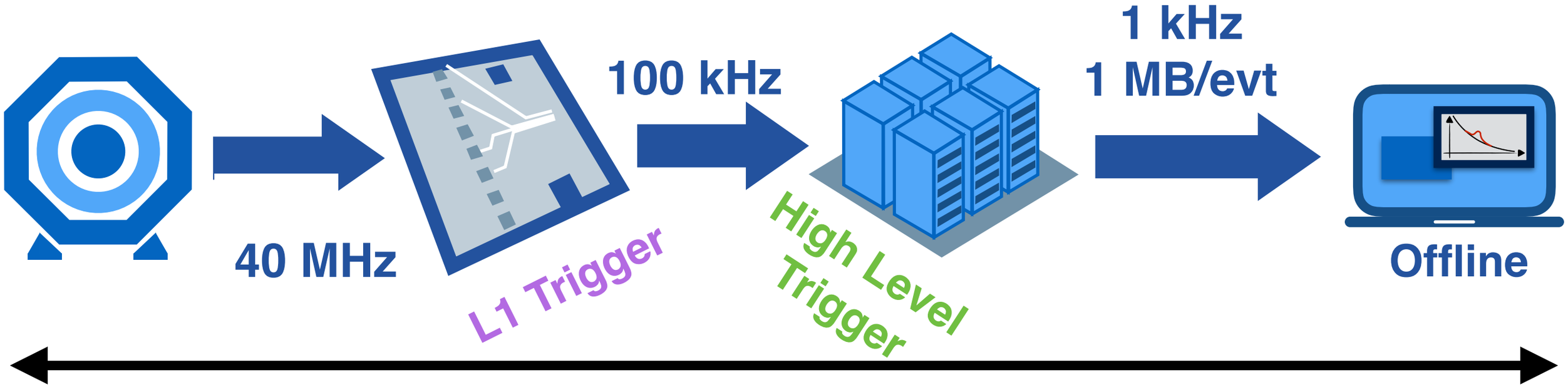
- Learning optimized nonlinear functions of many inputs for performing difficult tasks from (real or simulated) data
- Many successes in Jets: **identification of b-quark jets**, Higgs candidates, W/Z/top taggers …





Typically applied offline, not online (in hardware trigger)

Data Processing at CMS



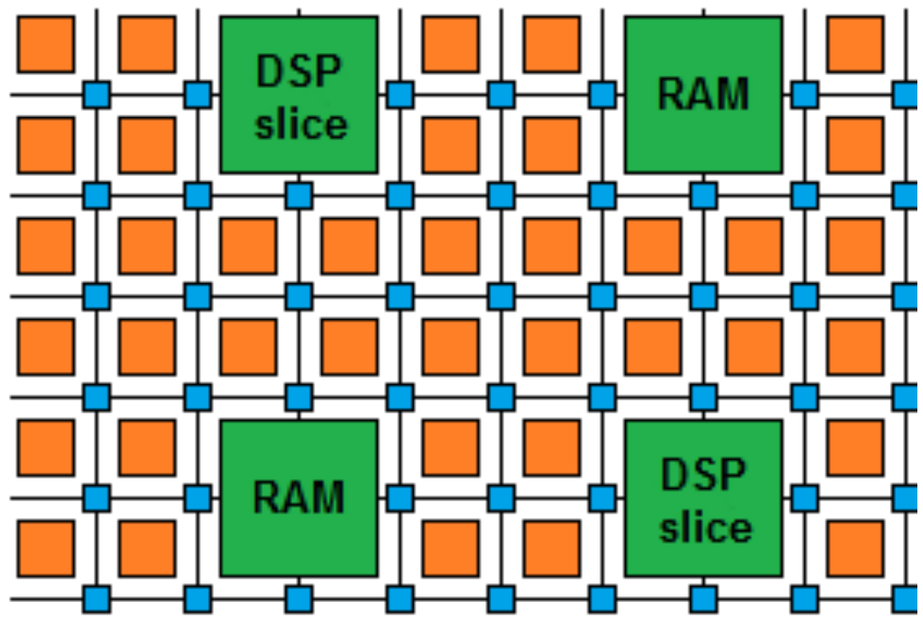
Level 1 Trigger (hardware, FPGAs based)
Decision in ~4 us

High Level Trigger (software, CPU based)
Decision in ~100 ms

Can we inference ML fast enough for trigger?

FPGAs and High Level Synthesis

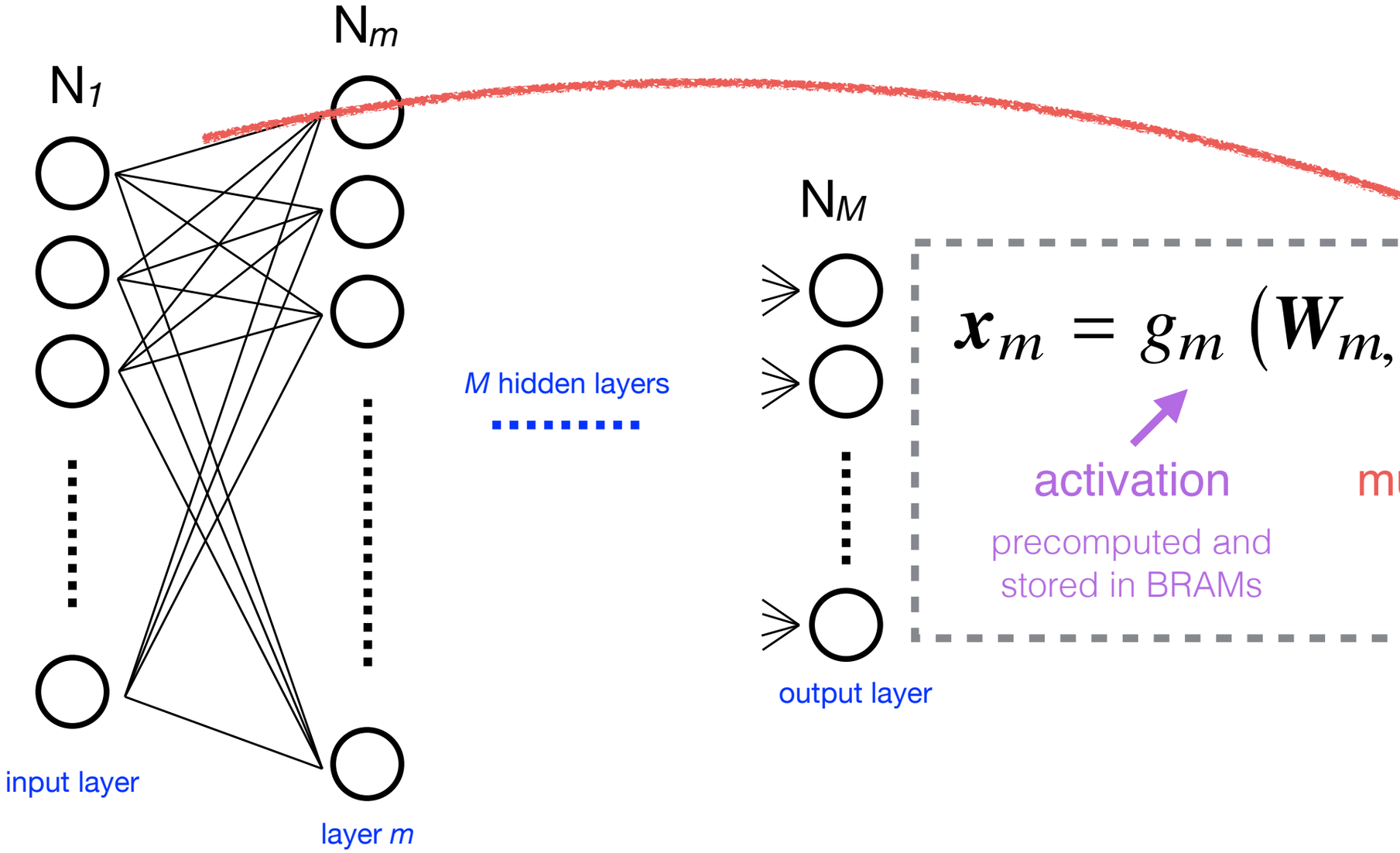
- Field Programmable Gate Arrays
  - Reprogrammable fabric of logic cells embedded with DSPs, BRAMs; high speed IO, etc.
    - *logic cells (O(M)): circuit block for logic operation*
    - *Digital Signal Processors (DSPs) used for multiplication ~(O(K))*
    - *BRAMs : on chip memory*
  - Massively parallel
  - Low power consumption (relative to CPU/ GPU)
- High Level Synthesis firmware
  - C-style code that generates traditional RTL code for FPGAs
    - C code with additional **directives**
  - Faster development for physicists




```
module dut(rst, clk, q);
  input rst;
  input clk;
  output q;
  reg [7:0] c;

  always @ (posedge clk)
  begin
    if (rst == 1b'1) begin
      c <= 8'b00000000;
  end
  else begin
    c <= c + 1;
  end

  assign q = c;
endmodule
```

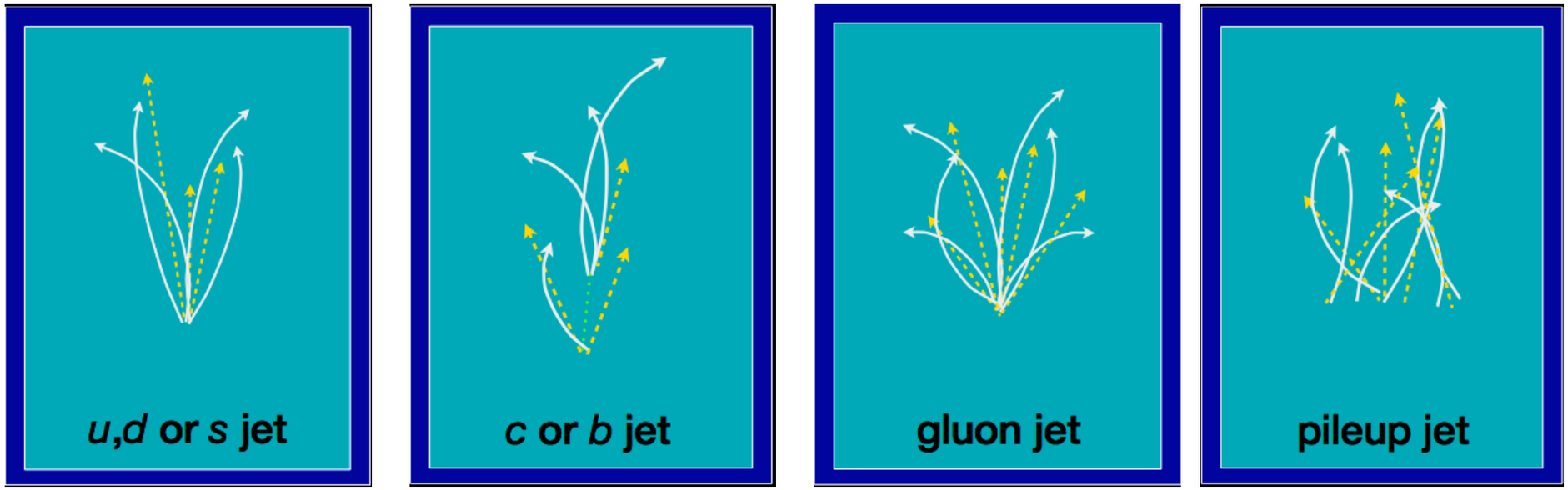
RTL Verilog

NN Inference



- NN inference = multiplication/addition and precomputed activation functions (look up table)
- The flexibility of FPGA suits the need of NN inference
- Leave the training of NN for GPU+CPU

Case Study: Jet Tagging



Jet Substructure Inputs

arXiv:1305.0007

Observables

m_{mMDT}

$N_2^{\beta=1,2}$

$M_2^{\beta=1,2}$

$C_1^{\beta=0,1,2}$

$C_2^{\beta=1,2}$

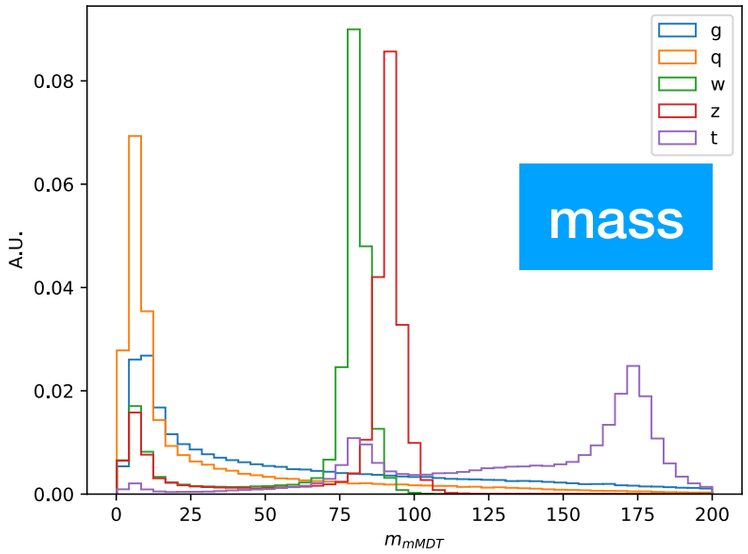
$D_2^{\beta=1,2}$

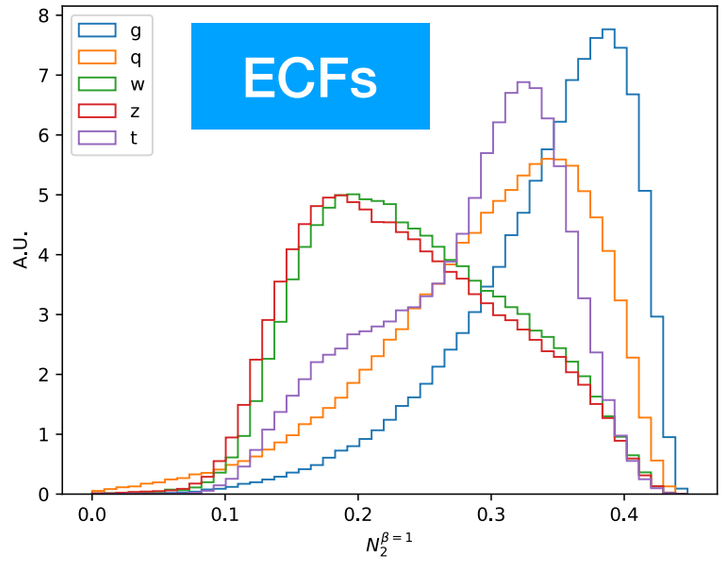
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$

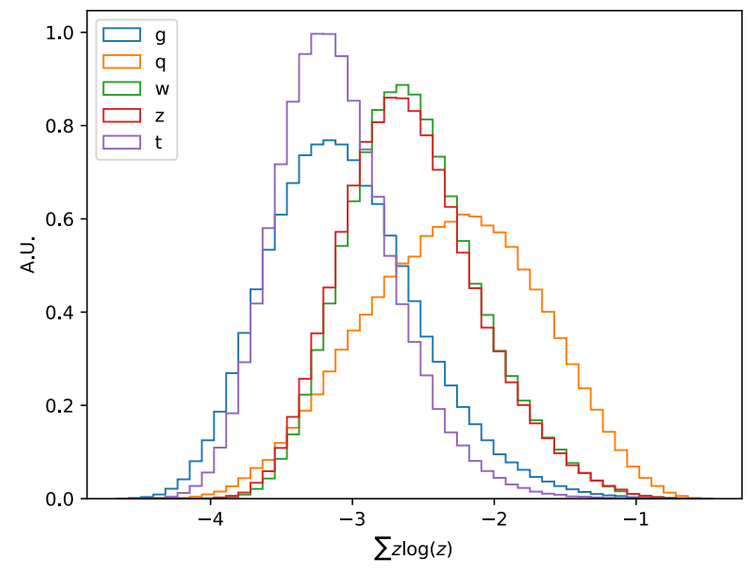
$\sum z \log z$

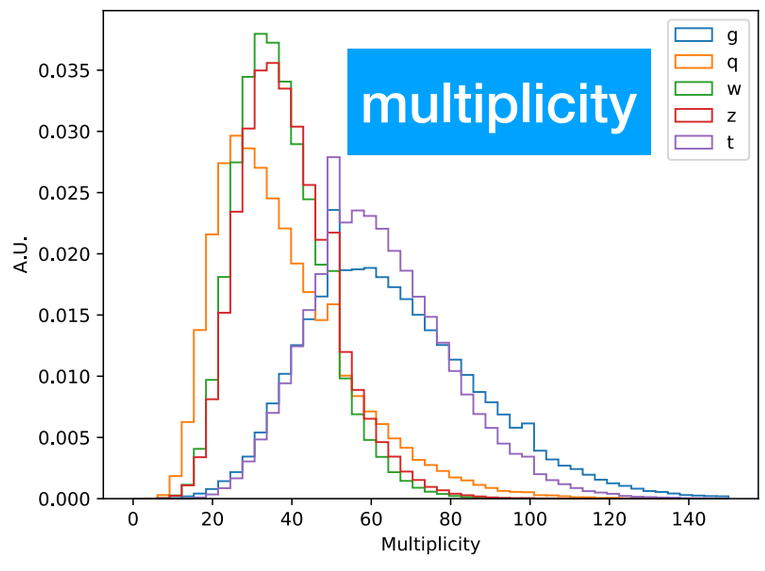
Multiplicity

arXiv:1307.0007









- Illustrative example, using high level-feature, not realistic for FPGA

Jet Substructure Inputs

arXiv:1305.0007

Observables

m_{mMDT}

$N_2^{\beta=1,2}$

$M_2^{\beta=1,2}$

$C_1^{\beta=0,1,2}$

$C_2^{\beta=1,2}$

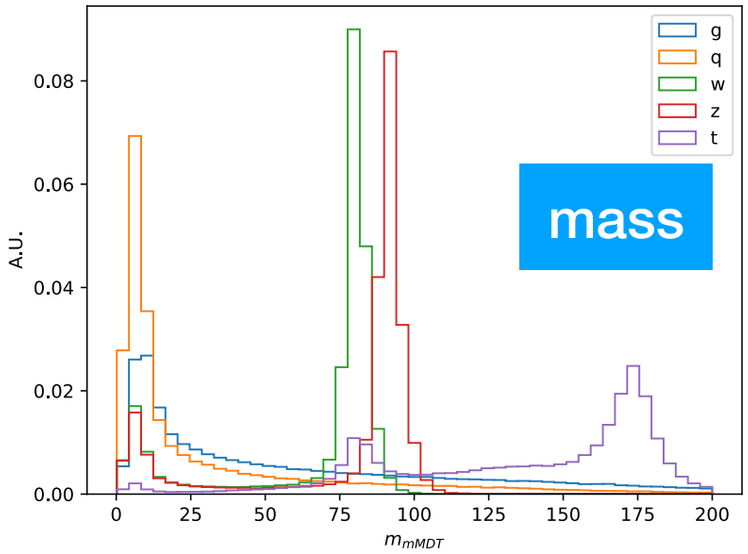
$D_2^{\beta=1,2}$

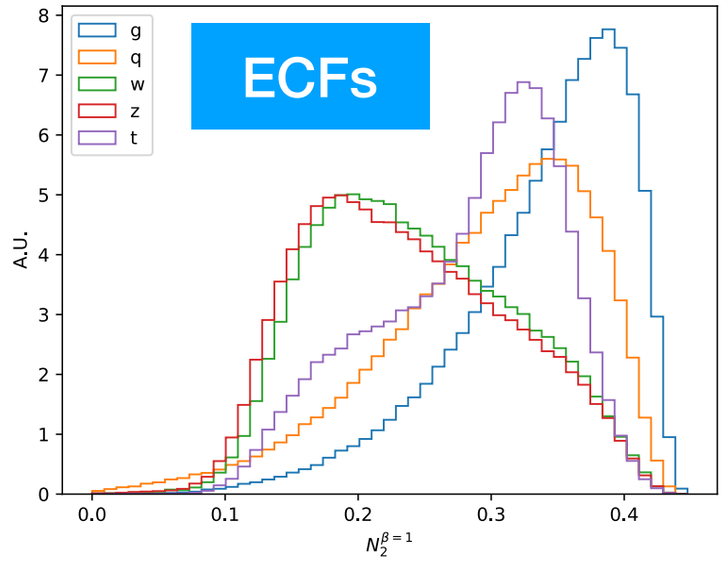
$D_2^{(\alpha,\beta)=(1,1),(1,2)}$

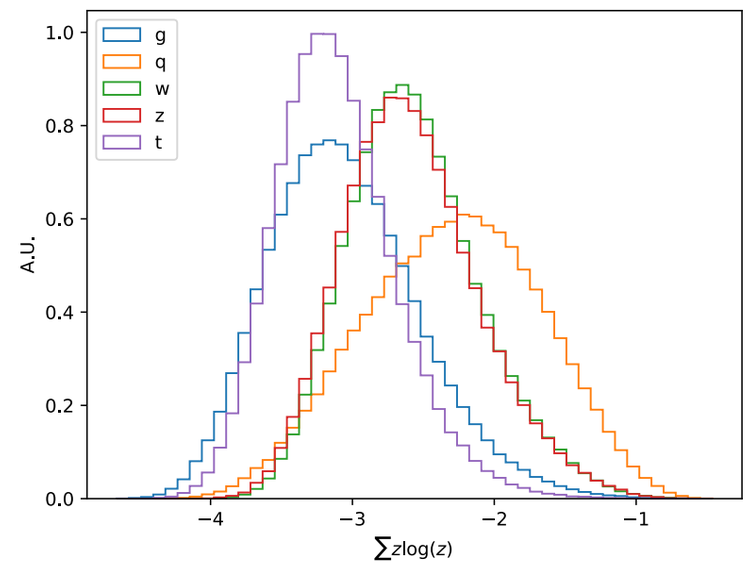
$\sum z \log z$

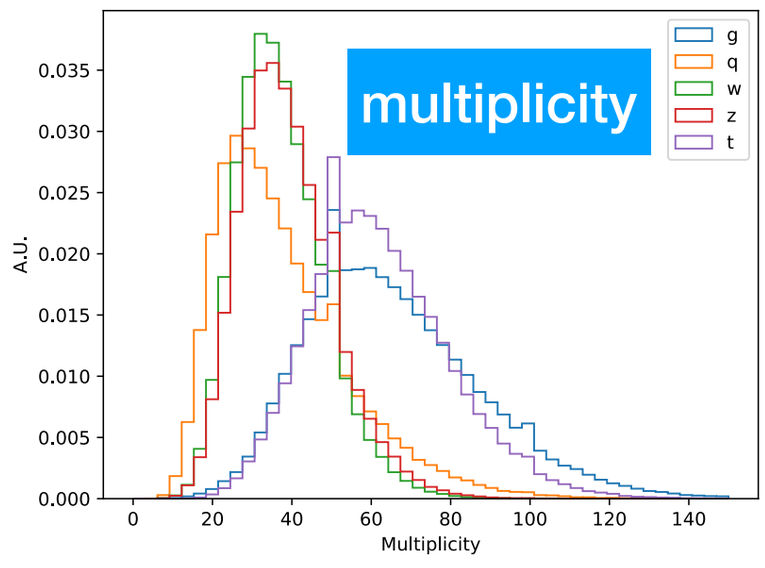
Multiplicity

arXiv:1307.0007





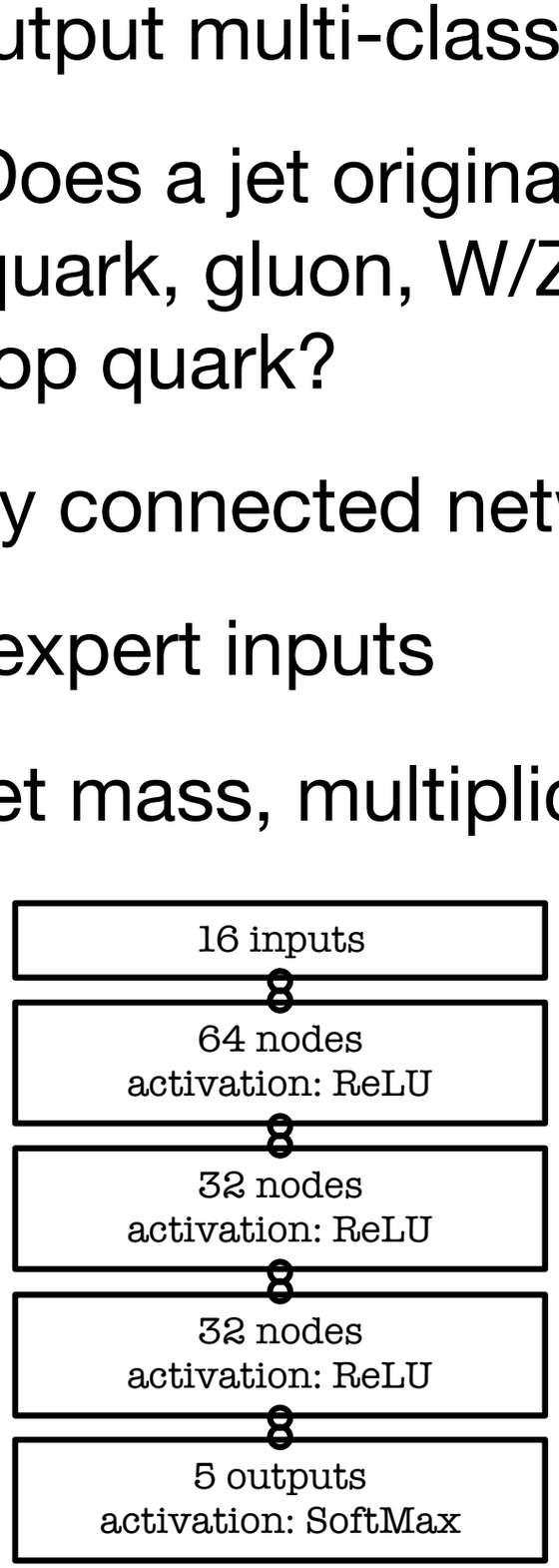


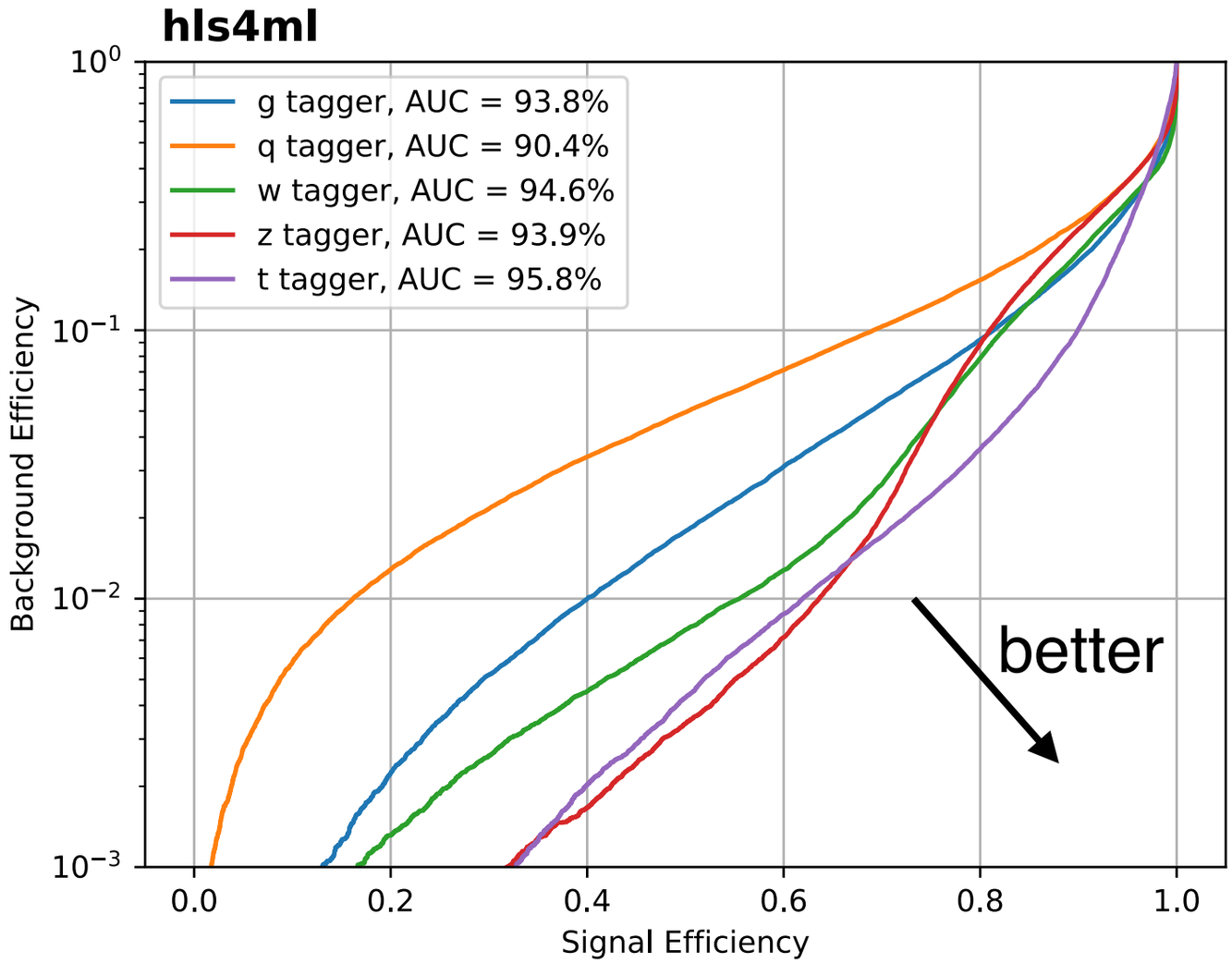


- Excited to see development from this workshop
- Looking for efficient and performant NN for substructure

Case Study: Jet Substructure

- 5 output multi-classifier
  - Does a jet originate from a quark, gluon, W/Z boson, top quark?
- Fully connected network
- 16 expert inputs
  - jet mass, multiplicity, ECFs

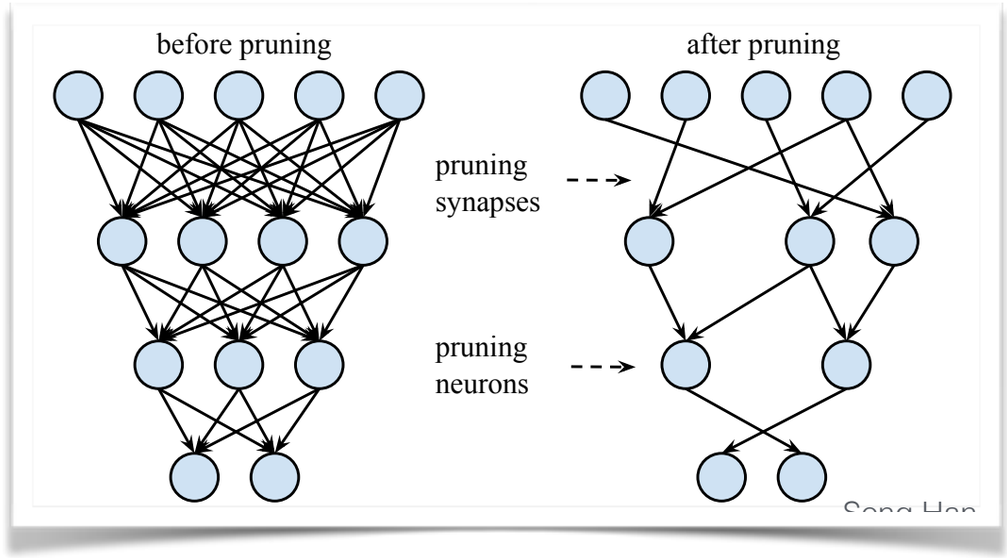


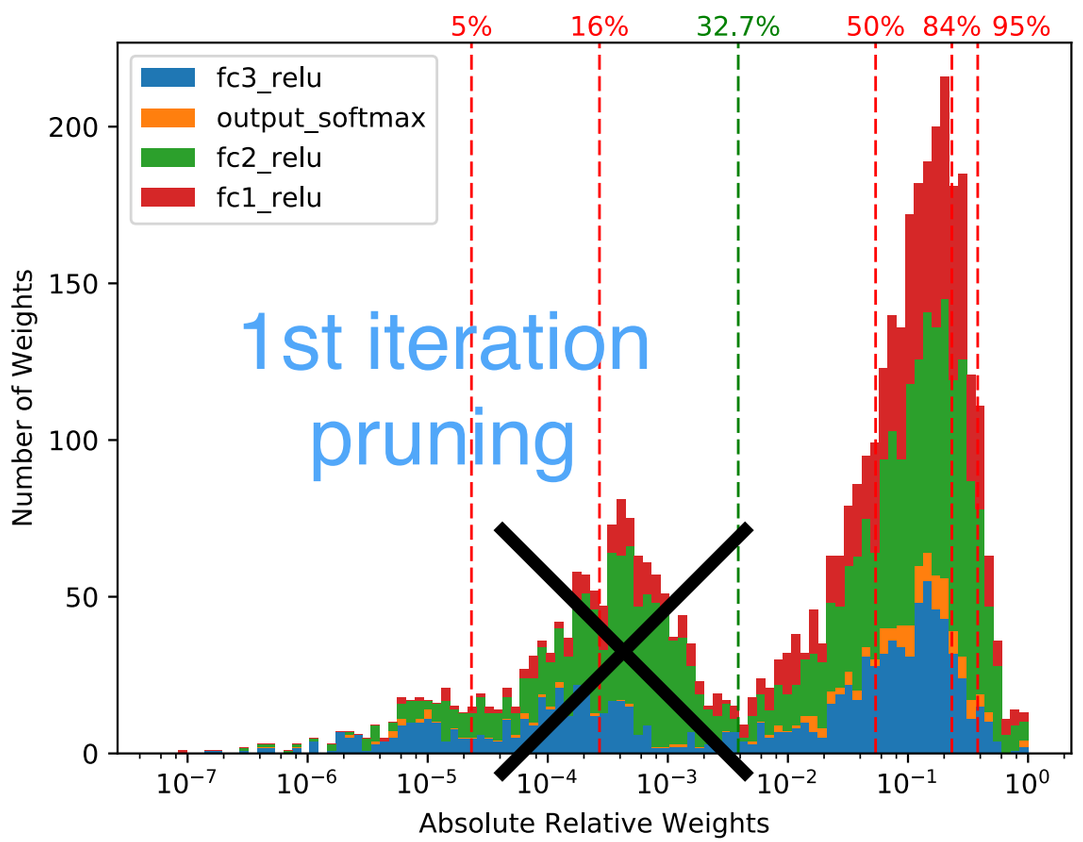


4,256 synapses

Compression

- FPGAs provide huge flexibility, but constrained by **input bandwidth**, limited **resources** on chip, **latency** requirement
- Compression techniques remove redundancy in model
  - Train with L_1 regularization
    - $L_\lambda(\mathbf{w}) = L(\mathbf{w}) + \lambda \sum_i |w_i|$
    - Downweights unimportant synapses
    - Histograms on right:
      |weight| / (max |weight|)
  - Remove / fix to zero lowest magnitude weights (per layer)
    - Removing synapses
- After 7 iteration, 70% reduction with no loss in performance





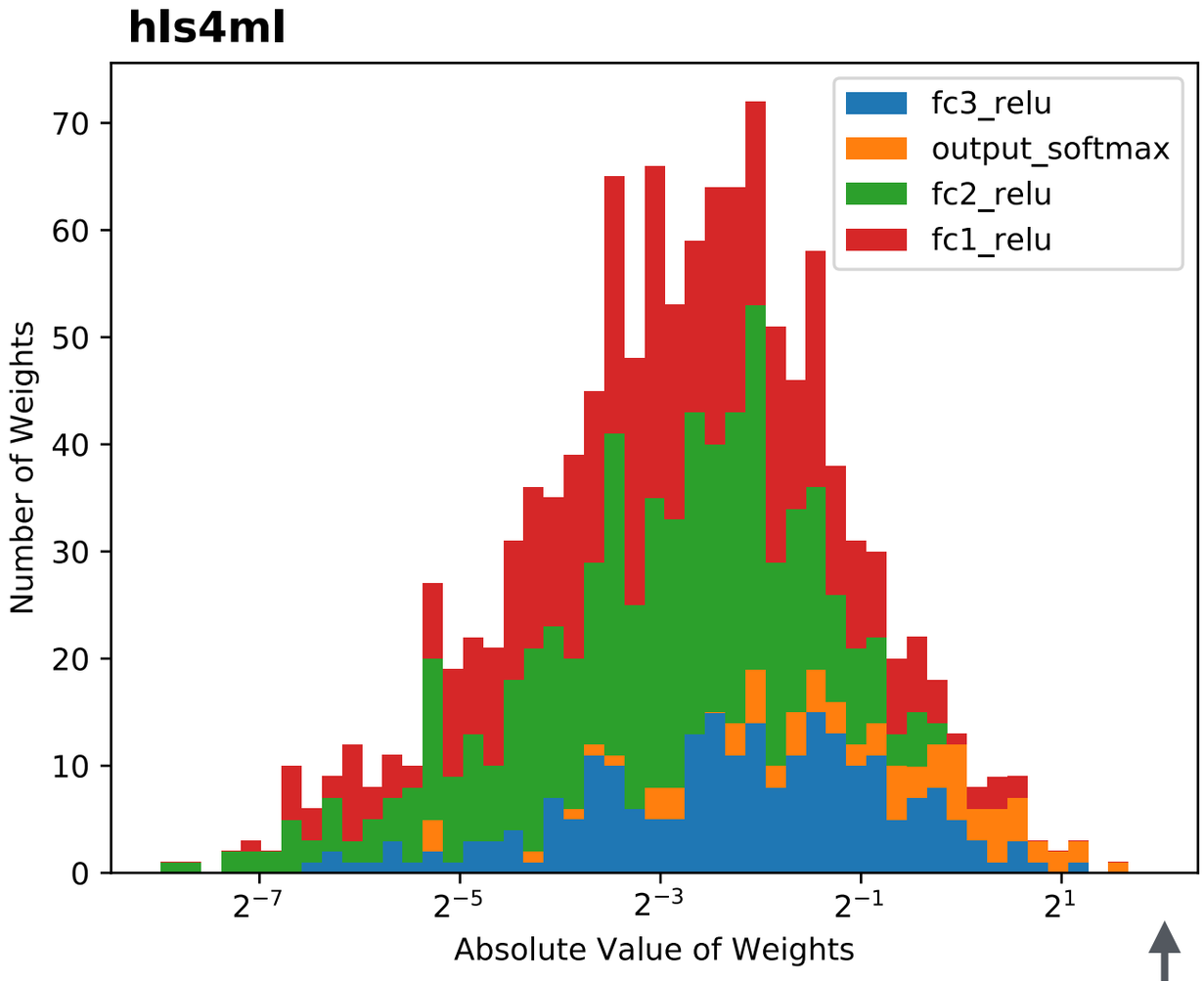
For further reading: [arXiv:1510.00149]

Quantization

- Fixed point data types
  - Faster and lower in FPGA-resource use than floating point
  - ap_fixed<width,integer>

ap_fixed<14,4>

- Recipe for minimizing number of bits:
  - Choose number of integer bits to avoid underflows/overflows that lead to drastic performance loss
  - Choose number of fractional bits to reach desired performance

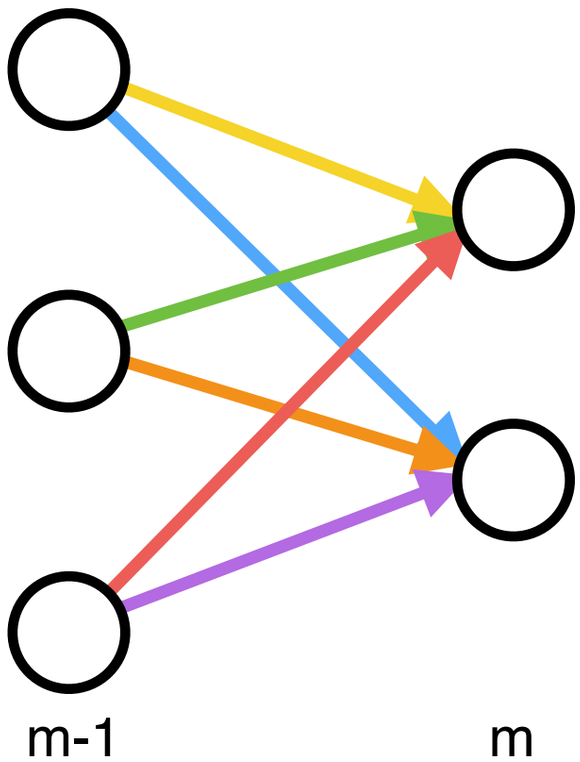


integer bits = 2 + 1 for sign
(need more for neurons)

Work in progress:
Binary/Ternary Network

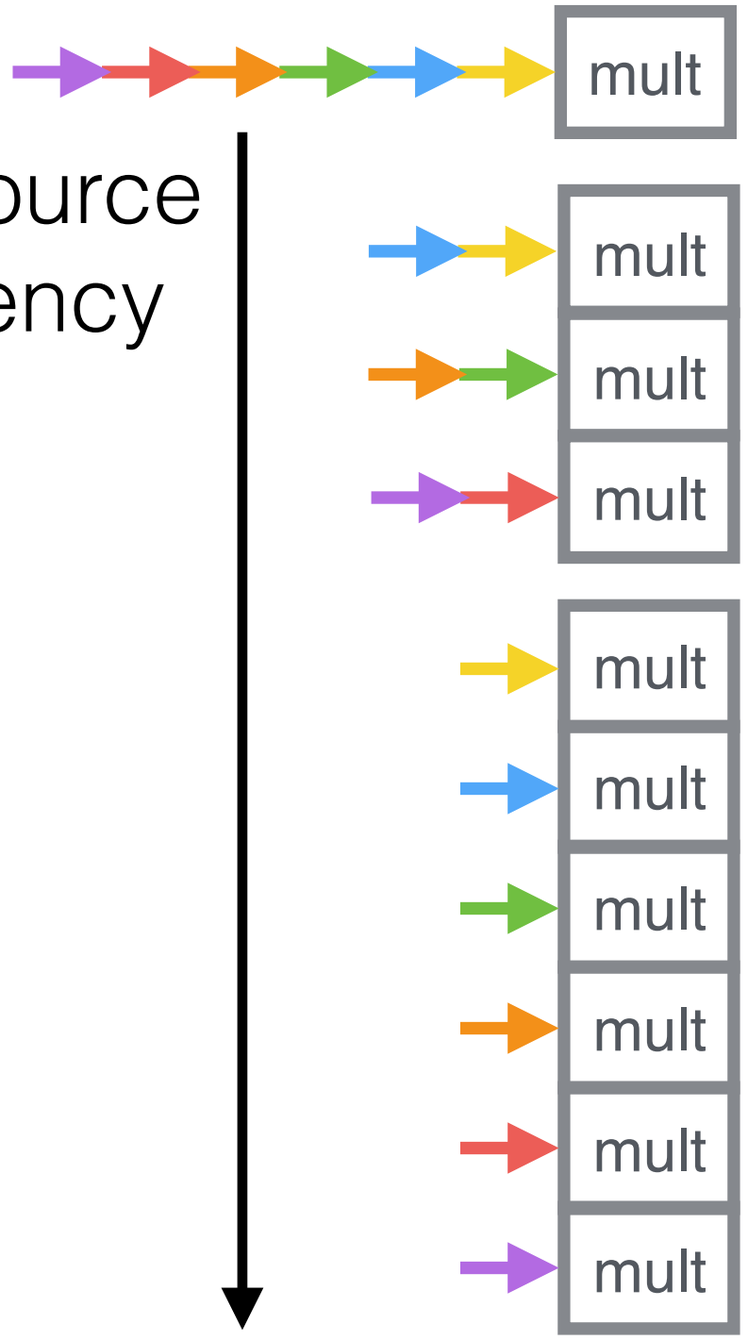
Parallelization

- Configurable “reuse factor” = number of times a multiplier is used to do a computation
- Trade-off between latency and resource usage



6 multiplications

More resource
Less latency



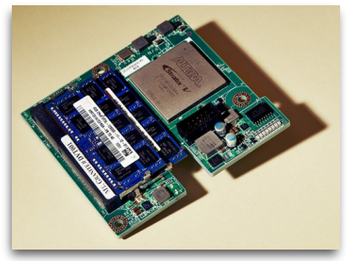
use 1 multiplier 6 times
accepts new set of inputs every 6 clocks

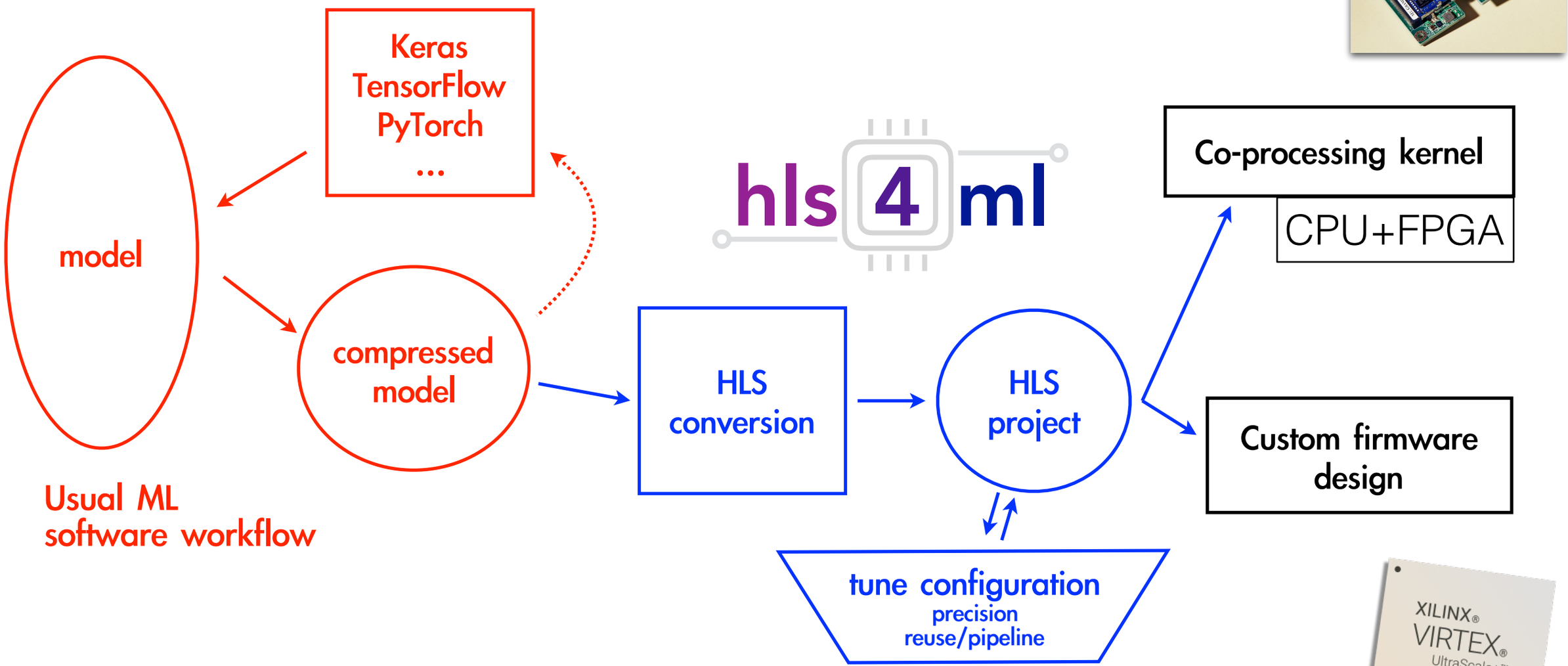
use 3 multipliers 2 times each
accepts new set of inputs every 2 clocks

use 6 multipliers 1 time each
accepts new set of inputs every clock

Compression, Quantization, and Parallelization made easy in

high level synthesis for machine learning





https://arxiv.org/abs/1804.06913
https://hls-fpga-machine-learning.github.io/hls4ml/

hls4ml case study

Examine compression, quantization, and parallelization in jet substructure case study

- Firmware block from **hls4ml** *ready in minutes* along with preliminary FPGA resource usage estimates
  - Final “implementation” gives exact resource usage (discussed later)

- Setup
  - Xilinx Vivado 2017.2
  - HLS target clock frequency: 200 MHz (5 clocks/BX)
  - Kintex Ultrascale, xcku115-flvb2104-2-i
    - 1.4M logic cells, 5,520 DSPs

Quantization & Compression

ap_fixed<width,integer>

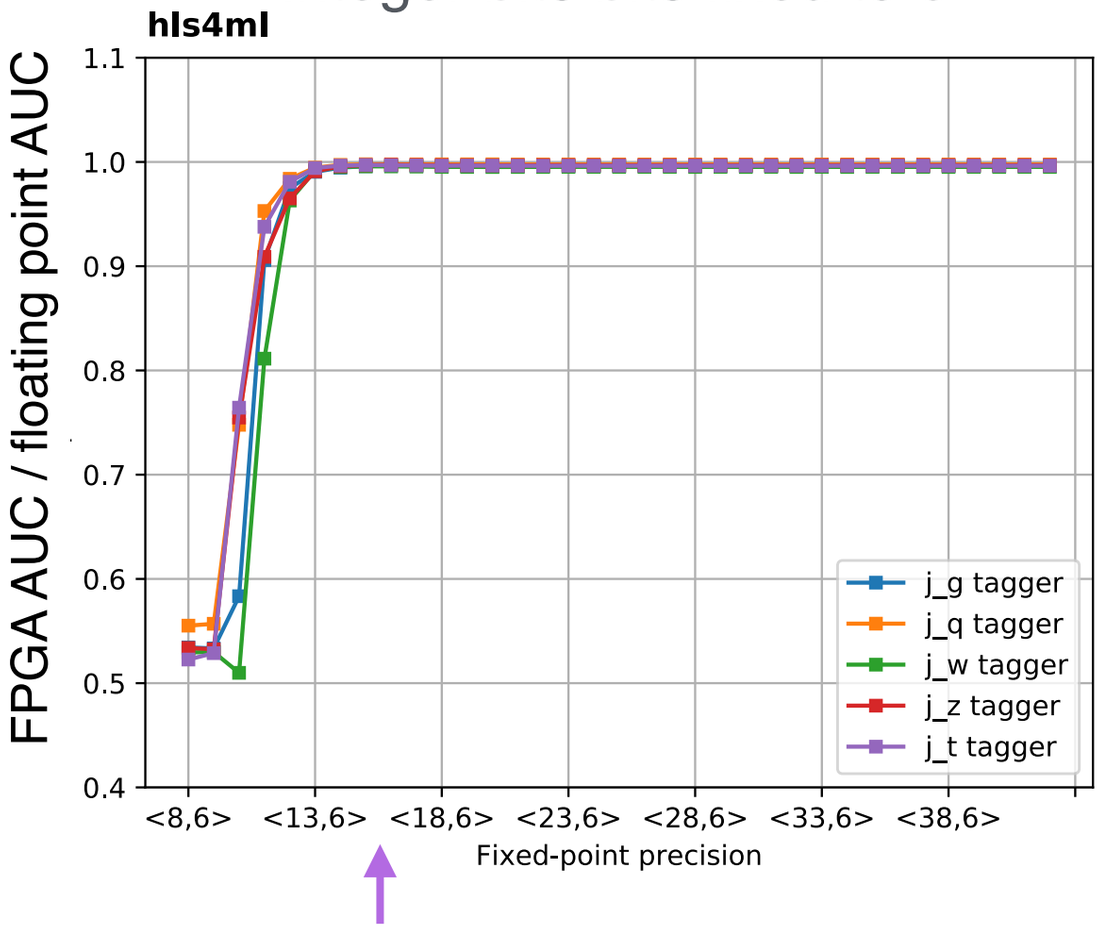
0101.1011101010

integer | fractional | width

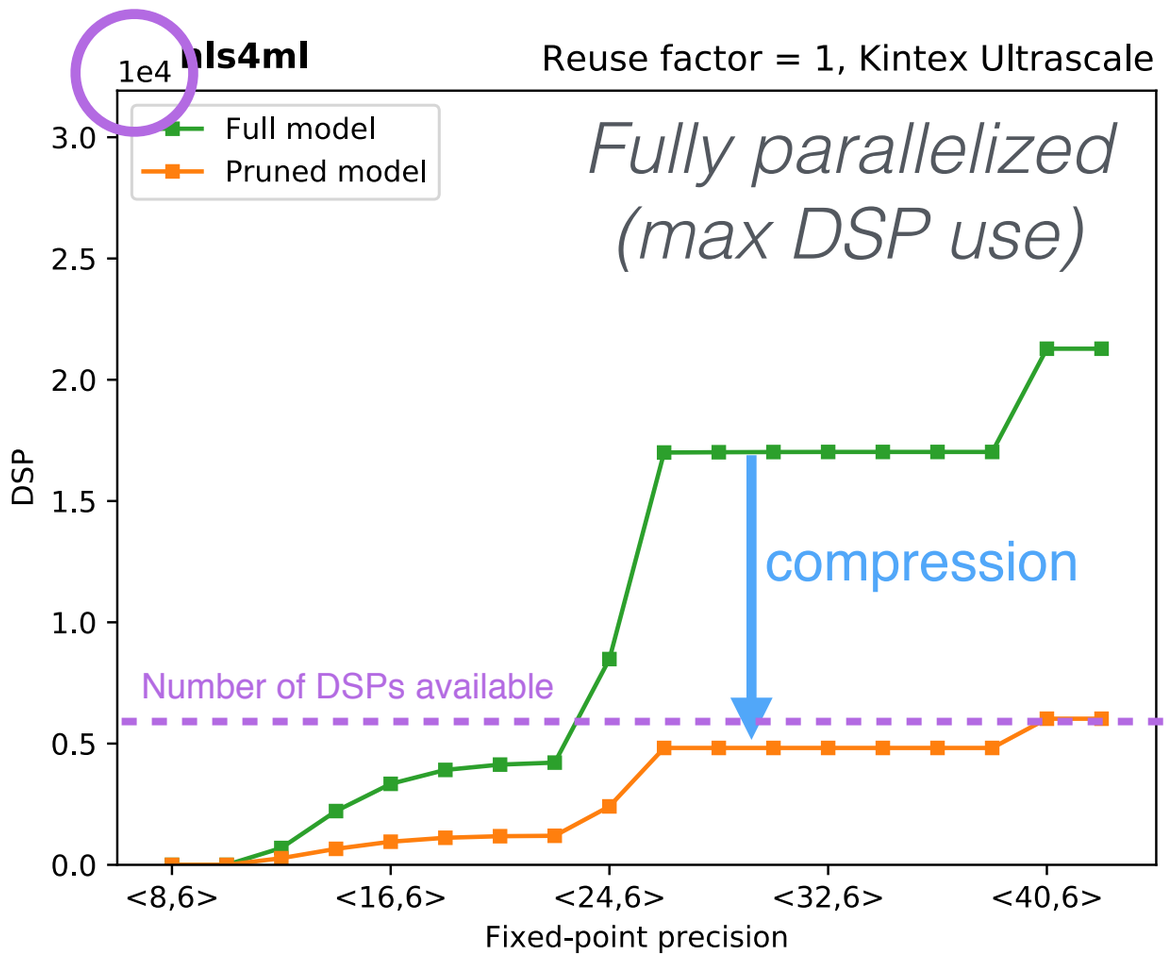
- DSPs (used for multiplication) will often be limiting resource
  - DSPs have a max size for input (e.g. 25x18 bits), so number of DSPs per multiplication changes with precision

Scan fractional bits

Integer bits bits fixed to 6

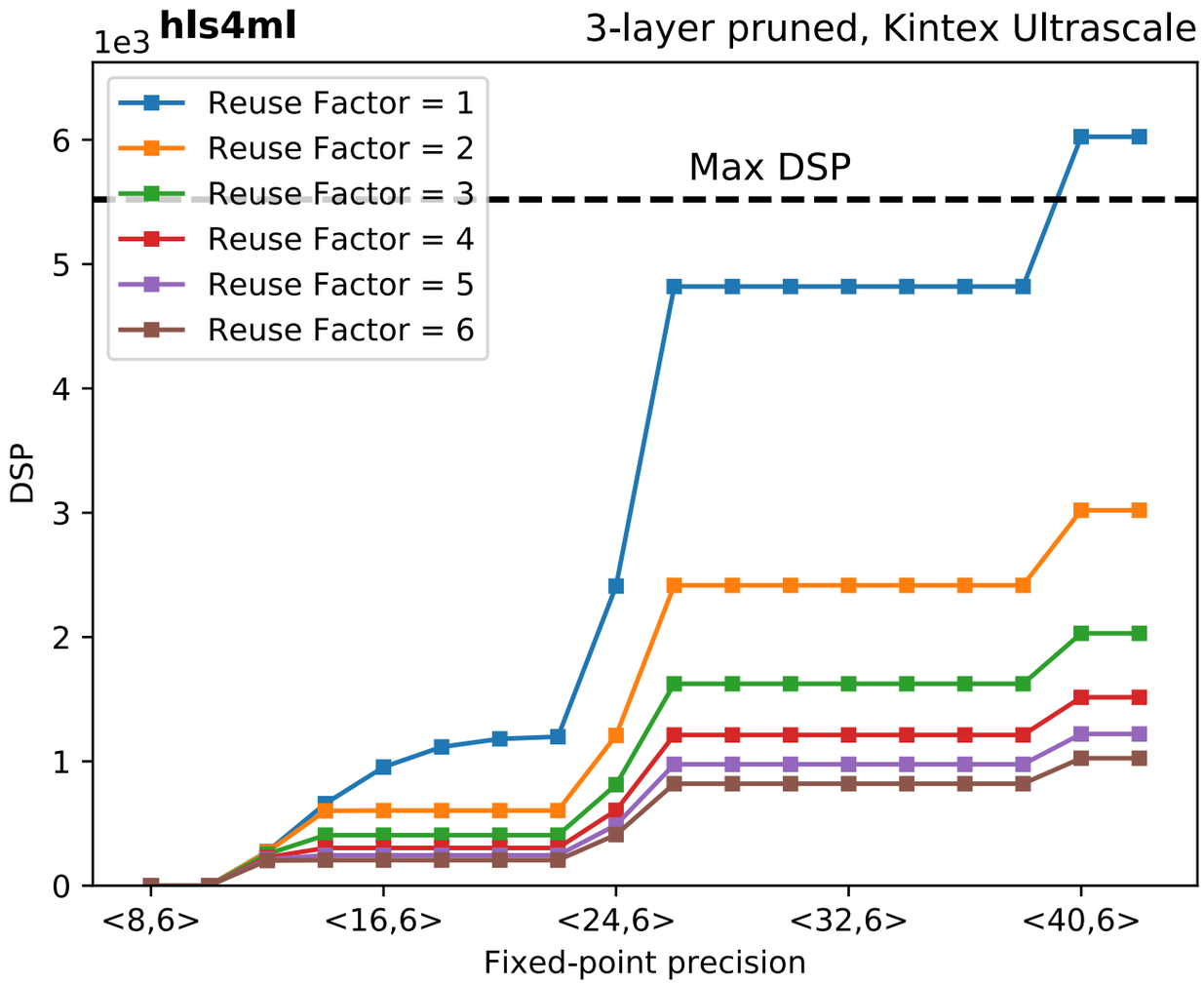


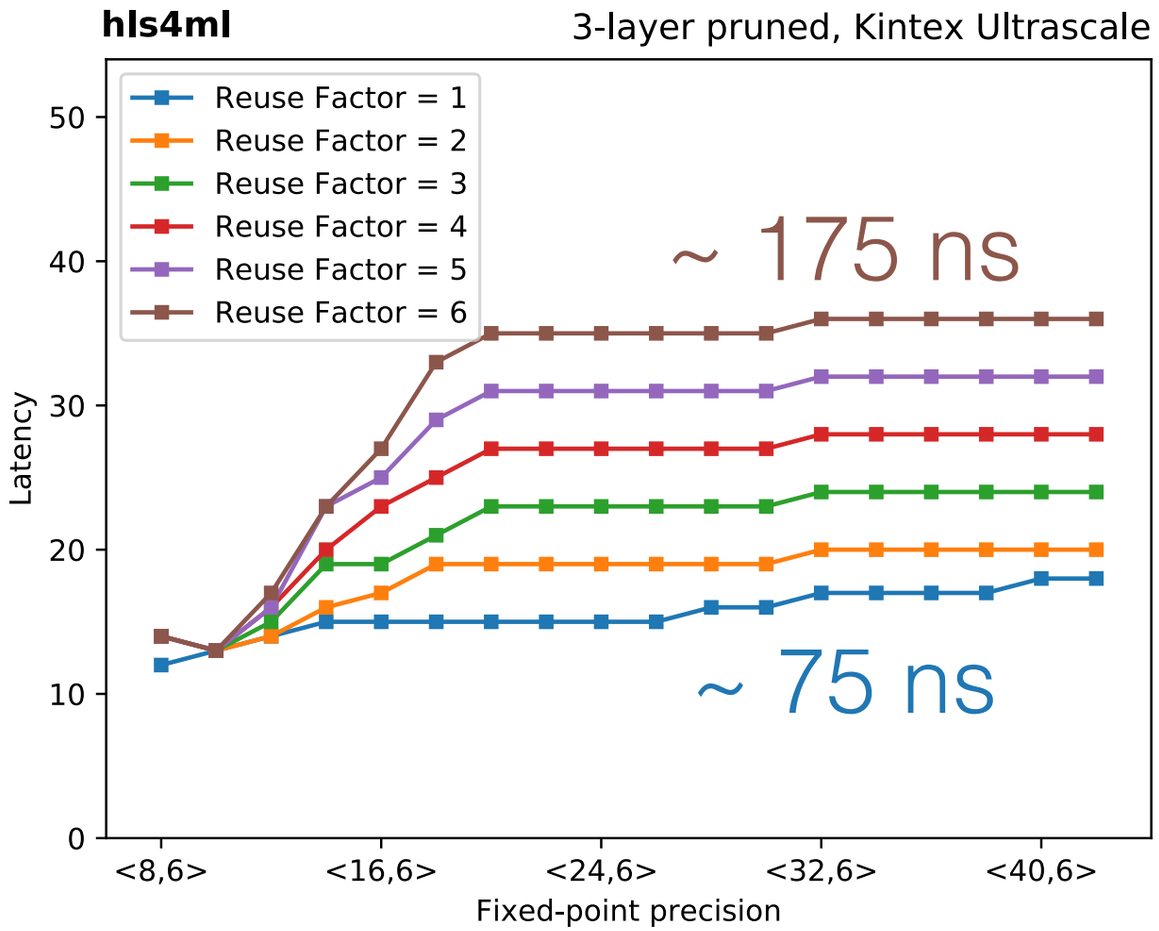
Full performance reached at 8 fractional bits

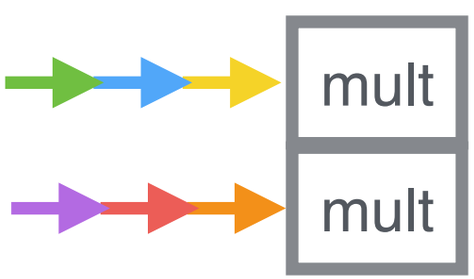


70% compression ~ 70% fewer DSPs

Reuse Factor







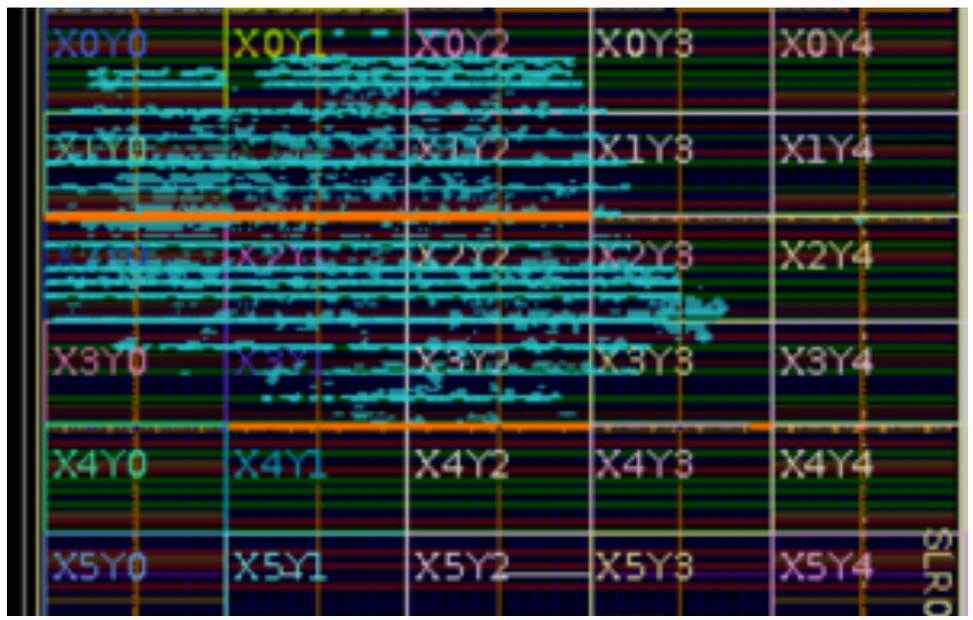
- Trade off between resource and latency
- NN inference within ~O(100)ns

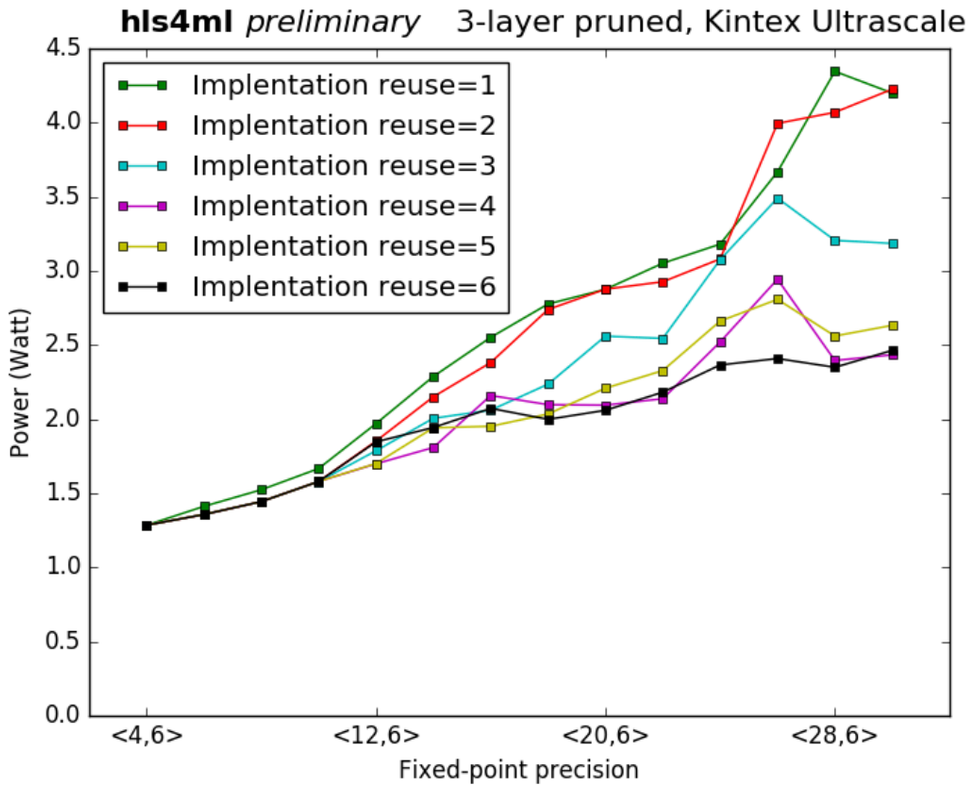
Firmware implementation

- Final implementation gives actual resource usage and timing estimate

- Implement in a minimal design, simply routing all firmware block's inputs and outputs to FPGA available pins

- Power usage increases with precision, it goes down for less throughput (higher reuse factor)





hls4ml New Developments

- Beta version is live! arXiv:1804.06913
- Work in progress:
  - LHC/DUNE applications
  - More network architectures:
    - Boosted Decision Tree (testing)
    - Binary Dense NN (testing)
    - Conv1D, 2D (testing)
    - BatchNormalization (prototyping)
    - LSTM, GRU (prototyping)
    - Graph-based NN (prototyping)

Fast inference of deep neural networks in FPGAs for particle physics

Javier Duarte[a] , Song Han[b] , Philip Harris[b] , Sergo Jindariani[a] , Edward Kreinar[c] , Benjamin Kreis[a] , Jennifer Ngadiuba[d] , Maurizio Pierini[d] , Ryan Rivera[a] , Nhan Tran[a] , Zhenbin Wu[e]

[a] *Fermi National Accelerator Laboratory, Batavia, IL 60510, USA*
[b] *Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
[c] *HawkEye360, Herndon, VA 20170, USA*
[d] *CERN, CH-1211 Geneva 23, Switzerland*
[e] *University of Illinois at Chicago, Chicago, IL 60607, USA*

E-mail: hls4ml.help@gmail.com

hls-fpga-machine-learning.github.io/hls4ml

More time means
➡ More resource reuse
➡ Bigger networks

“L1” Trigger

“High Level” Trigger

Offline

1 ns

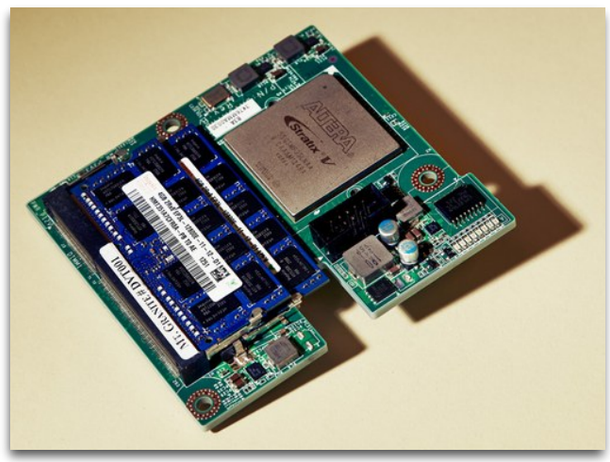
1 us

1 ms

1 s

Pure FPGA

Acceleration with FPGAs



heterogenous computing

Already at Cloud Scale

>100k FPGA network

Microsoft Catapult
Microsoft Brainwave

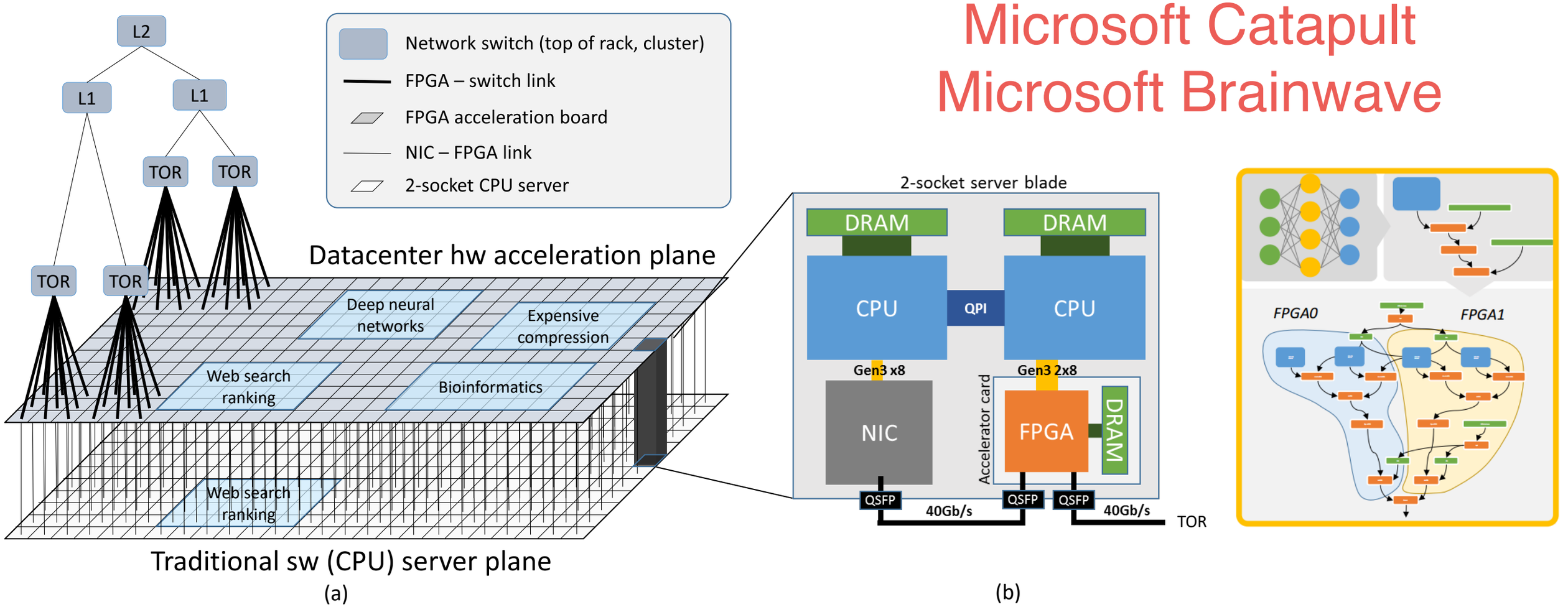
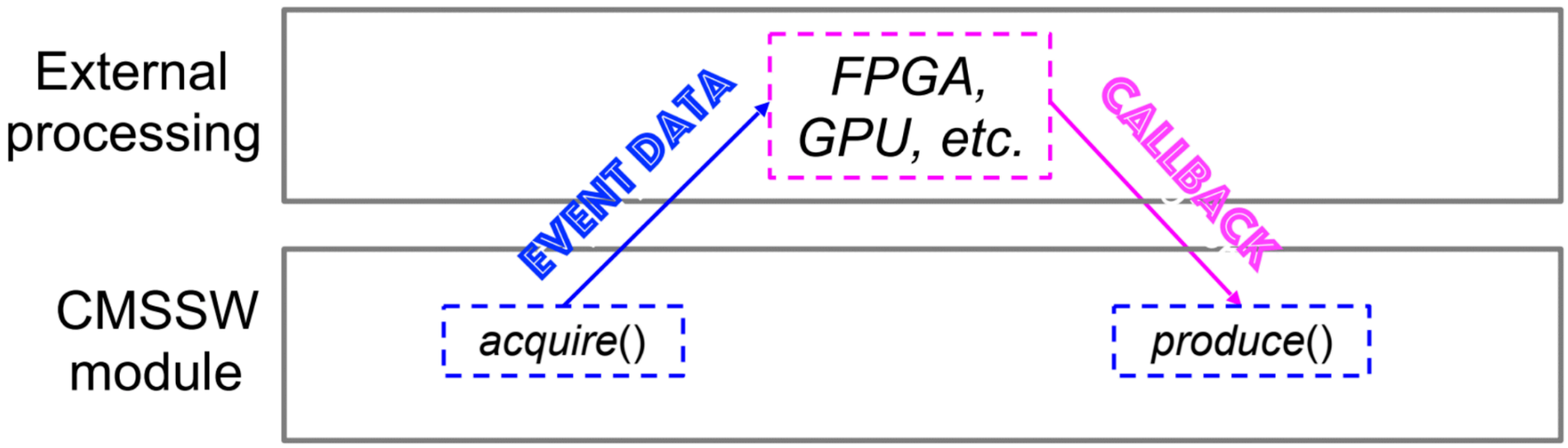


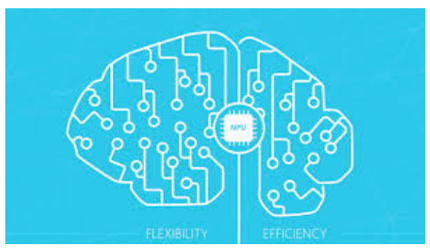
Fig. 1. (a) Decoupled Programmable Hardware Plane, (b) Server + FPGA schematic.

- Brainwave provides a full services at scale, multi-FPGA/CPU fabric
- Demonstrated large improvement in processing time for Bing searches
- Caveat: only selected DNN models currently available (ResNet50)

SONIC in CMS

- **S**ervices for **O**ptimized **N**etwork **I**nference on **C**oprocessors
  - a framework to exploit cloud resources for on-demand inference
- CPU runs "locally" and sends data to the cloud system, using FPGAs for inference
- Good performance in initial tests with ResNet50 on Microsoft Azure
  - "remote": cmslpc @ FNAL to Azure (VA), ‹time› = 56 ms
  - "onprem": run CMSSW on Azure VM, ‹time› = 10 ms
    (~2 ms on FPGA, rest is classifying and I/O)
  - CPU (cmslpc): 1.75 sec
    (6 min to load ResNet50 session)



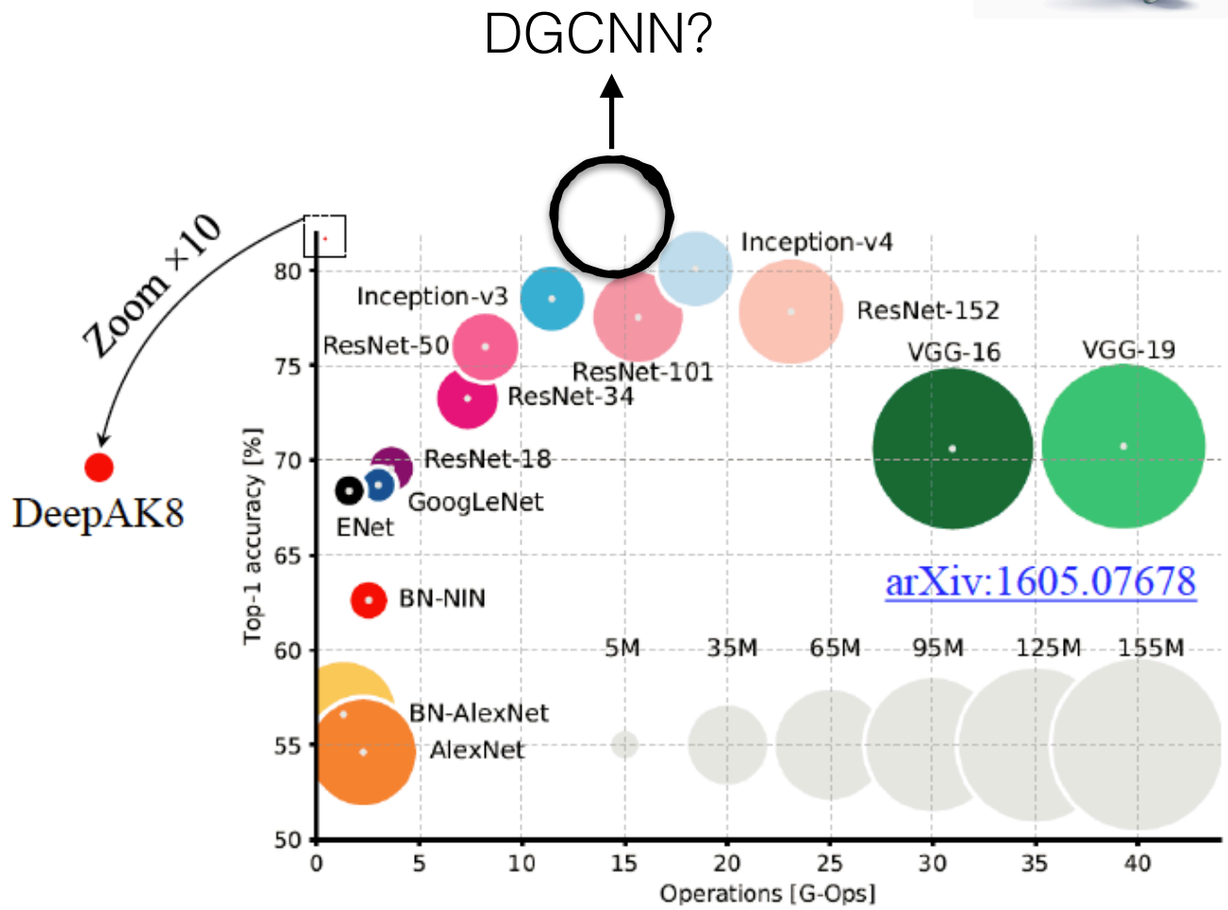


aws

Summary

- **hls4ml**, translates machine learning inference into firmware
  - aims to be a flexible tool to implement NN with low latency
  - paper: arxiv.1804.06913

- SONIC: exploring applications for acceleration with CPU-FPGA co-processors
  - Tested with Microsoft Azure
  - Testing with Amazon AWS FPGA (F1) instance



- ResNet50: 25M parameters, 7B operations
- Examples of large networks used in CMS:
  - DeepAK8, 500K parameters, 15M operations
  - DeepDoubleB, 40K parameters, 700K operations

- ~~While HEP NN is relative small compared to commercial NN~~, we should explore the most efficient way to get good jet substructure performance

& economical

Backup

hls4ml program flow

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

Inputs →

Config

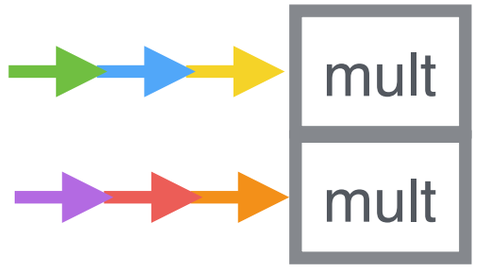
```
KerasJson: example-keras-model-files/KERAS_1layer.json
KerasH5:   example-keras-model-files/KERAS_1layer_weights.h5
OutputDir: my-hls-test
ProjectName: myproject
XilinxPart: xc7vx690tffg1927-2
ClockPeriod: 5

IOType: io_parallel # options: io_serial/io_parallel
ReuseFactor: 1
DefaultPrecision: ap_fixed<18,8>
```

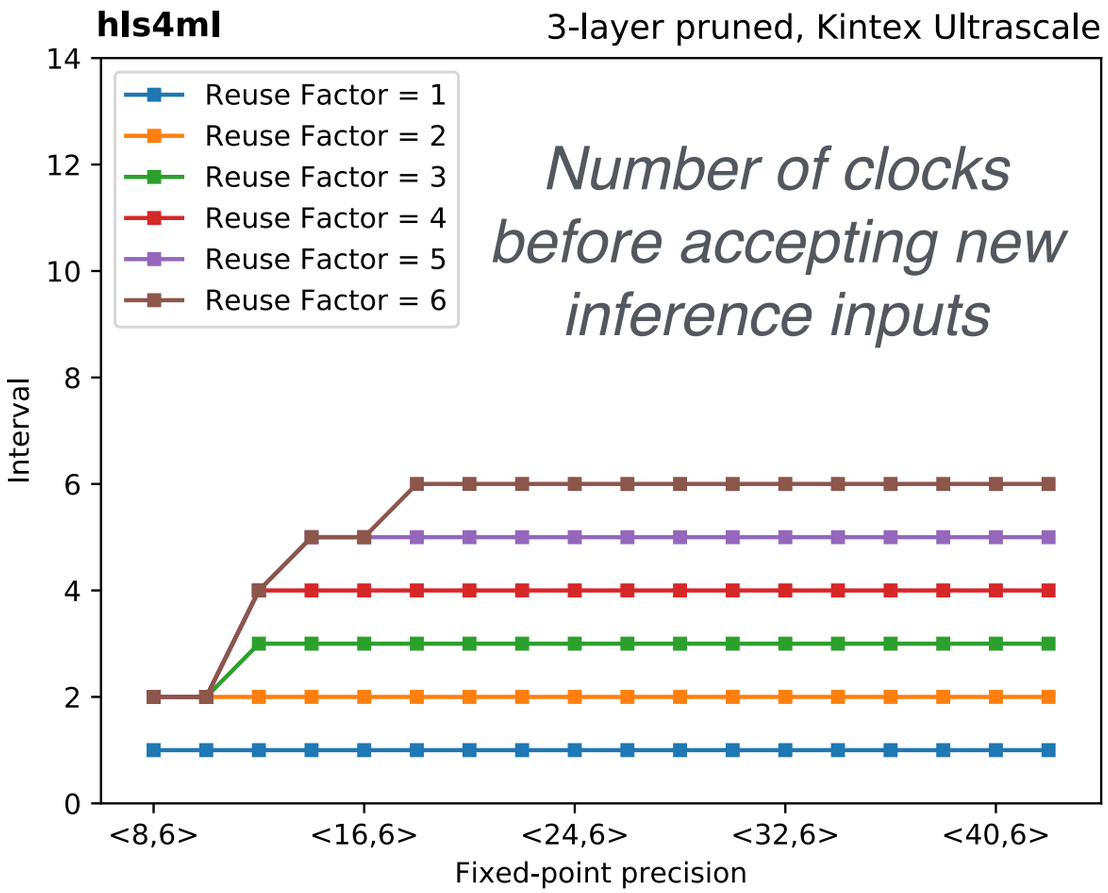
- `IOType`: parallelize or serialize
- `ReuseFactor`: how much to parallelize
- `DefaultPrecision`: inputs, weights, biases

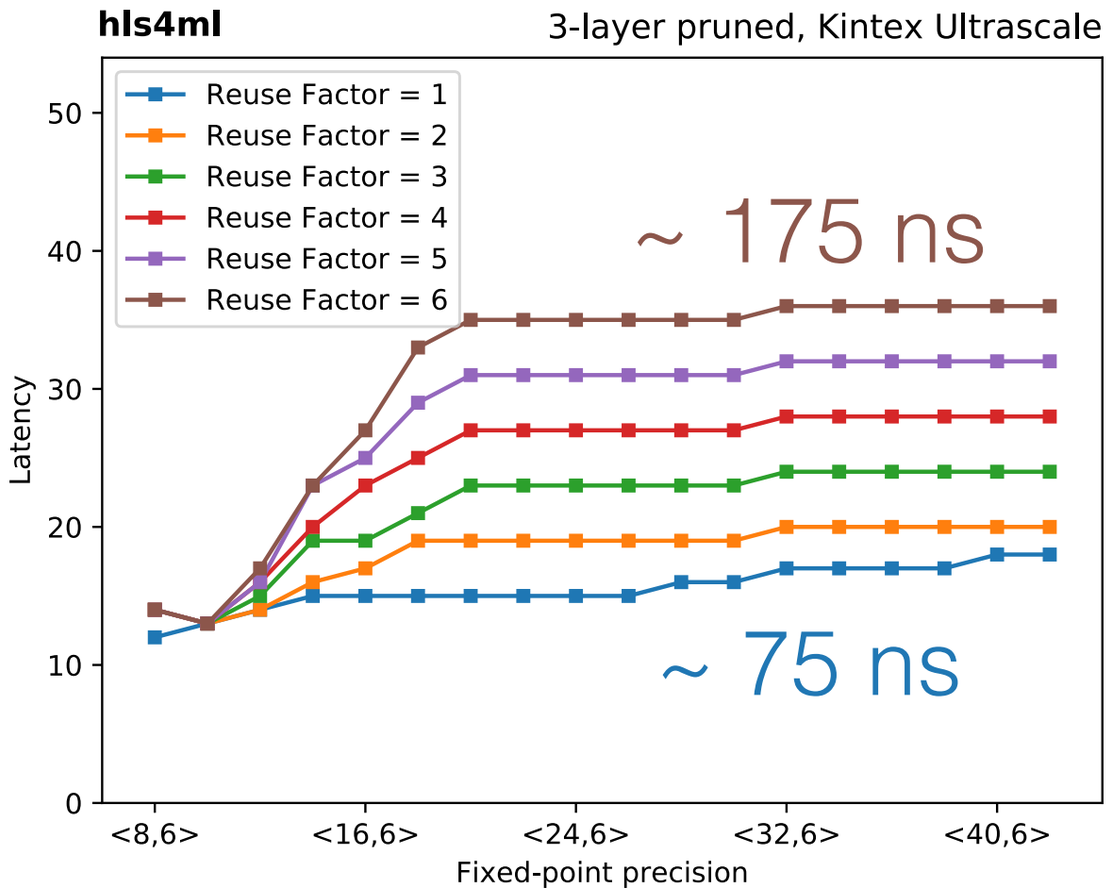
```
my-hls-test/:
build_prj.tcl
firmware
myproject_test.cpp
```

vivado_hls -f build_prj.tcl, produce a firmware block in ~m

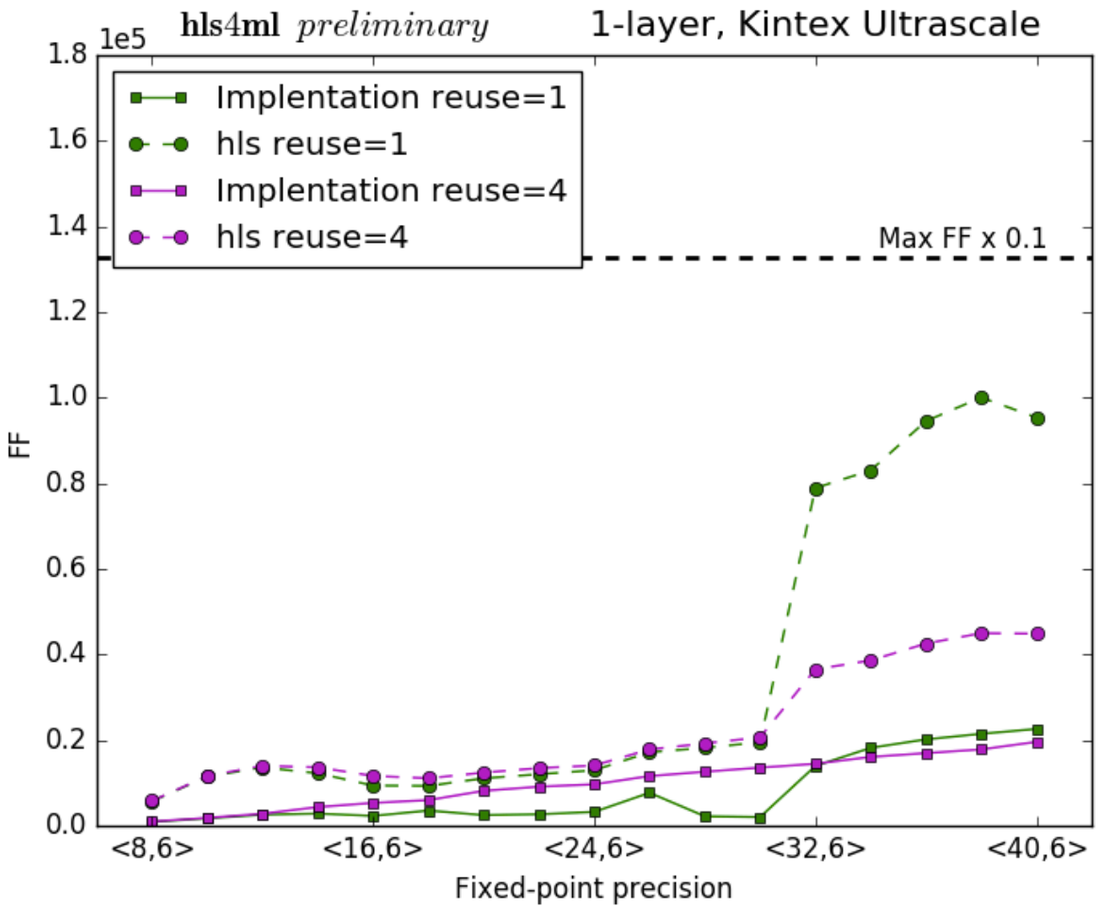
Reuse comes at a cost



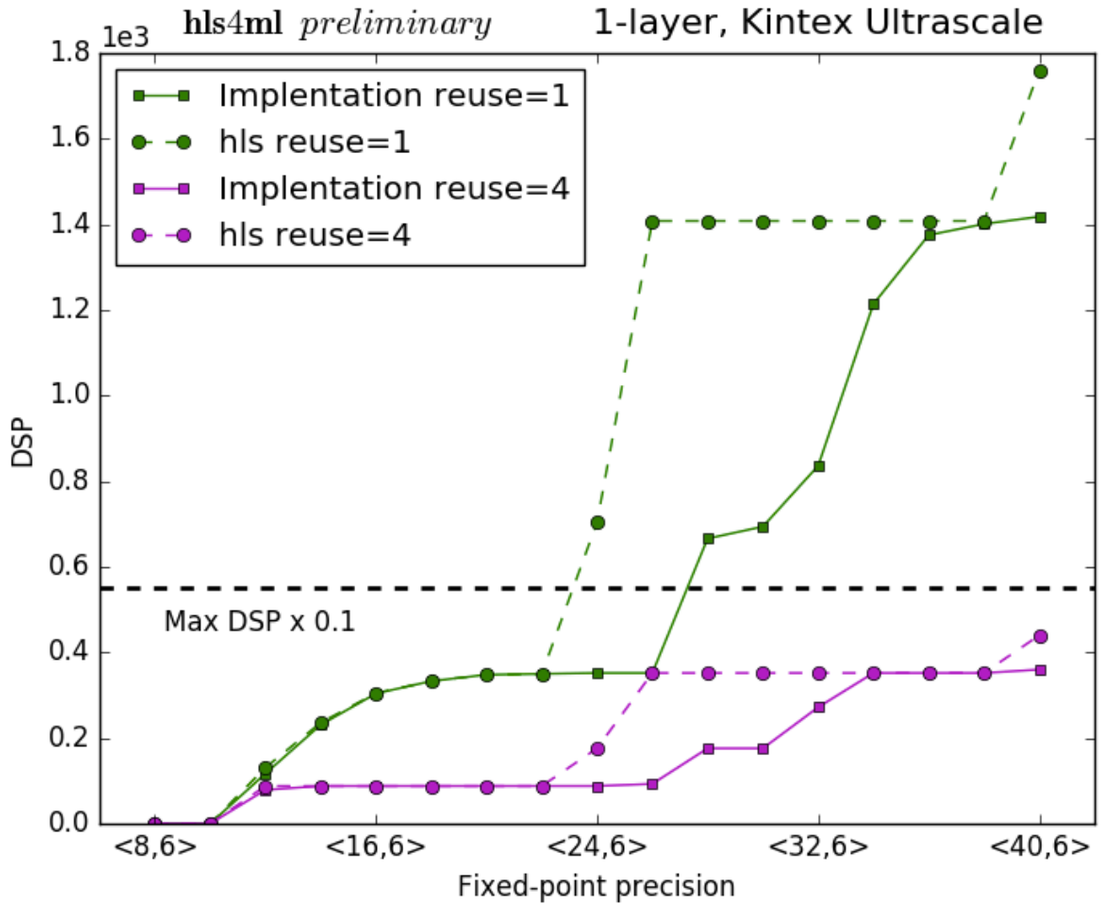




Firmware implementation

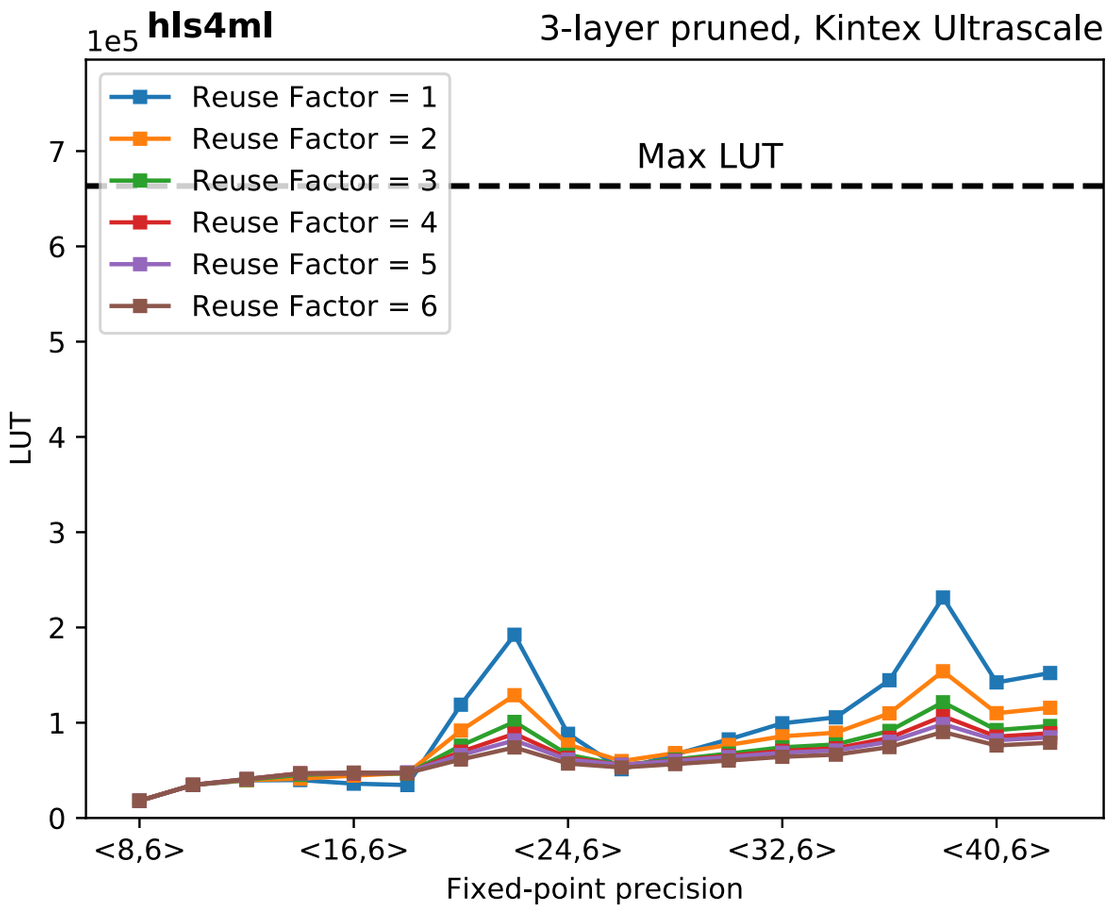


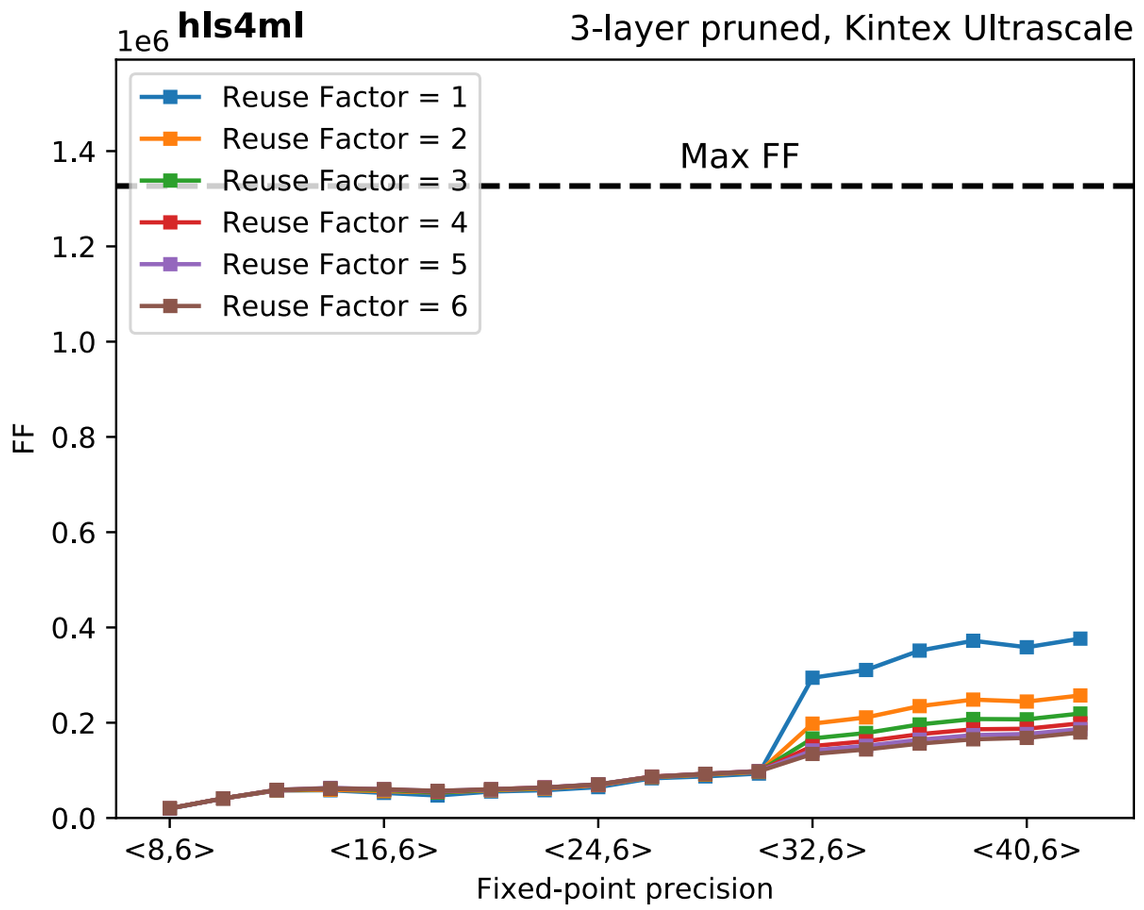
DSPs are estimated fairly well around DSP precision transition, Vivado does further optimizations



FFs are overestimated by a factor of 2-4
LUTs are overestimated by a factor of ~2

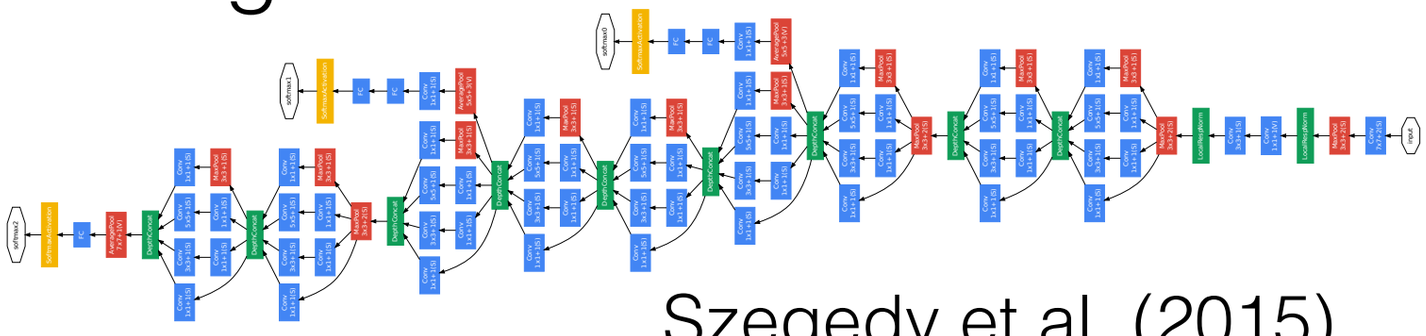
Other Resources





- Fairly linear increase with precision
- Small percentage of total available
  - *But could matter depending on what else is on FPGA*
- Spikes present at steep transitions in DSP usage not observed in implementation

GoogLeNet



Szegedy et al. (2015)

TABLE IV: Layer- and module-wise performance on the GoogLeNet model.

Layer #	Ops (M-ops)	Theor. Time (ms)	Actual Time (ms)	Perf. (G-ops/s)	Eff. %
Layer 1	236	1.84	2.50	94.4	73.7%
Layer 2	756	5.49	5.64	134.0	97.3%
Inception 3a	256	2.25	2.59	98.9	86.9%
Inception 3b	609	4.98	5.22	116.6	95.4%
Inception 4a	147	1.28	1.45	101.5	88.3%
Inception 4b	176	1.49	1.69	104.0	88.2%
inception 4c	214	1.66	1.87	114.4	88.8%
inception 4d	237	1.92	2.03	116.8	94.6%
Ineption 4e	340	2.68	2.84	119.7	94.4%
Inception 5a	112	0.78	0.83	134.9	94.0%
Inception 5b	141	1.04	1.09	129.7	95.4%
Total	3224	25.41	27.75	116.2	91.6%

TABLE VI: A comparison of throughput and efficiency across recent works in literature.

	Eyeriss[26]		**Zhang[27]**	**Caffeine[18]**	**Qiu[19]**	**HWCE[28]**	**Snowflake**		
	AlexNet	VGG	AlexNet	VGG	VGG	AlexNet	AlexNet	GoogLeNet	ResNet-50
Platform	65nm CMOS	65nm CMOS	VX485T	KU060	Zynq 7045	Zynq 7045	Zynq 7045	Zynq 7045	Zynq 7045
Clock (MHz)	200	200	100	200	150	100	250	250	250
Precision	16-bit fixed	16-bit fixed	32-bit float	16-bit fixed	16-bit fixed	16-bit fixed	16-bit fixed	16-bit fixed	16-bit fixed
MAC Units	168	168	448	1058	780	800	256	256	256
Actual Perf. (G-ops/s)	46.1	24.5	61.6	310.0	187.8	140.8	120.3	116.2	122.3
Peak Perf. (G-ops/s)	67.2	67.2	89.6	423.2	234	160	128	128	128
Frame Rate (fps)	38.4	0.8	51.3	258.3	6.3	117.3	100.3	36.3	17.7
Power (W)	0.28	0.24	18.61	25	9.63	-	9.48	9.53	9.61
Energy Eff. (G-ops/J)	164.6	102.1	3.3	12.4	19.5	-	12.7	12.2	12.7
Computational Eff.	69%	36%	69%	73%	80%	88%	**94%**	**91%**	**95%**

27.75 ms latency

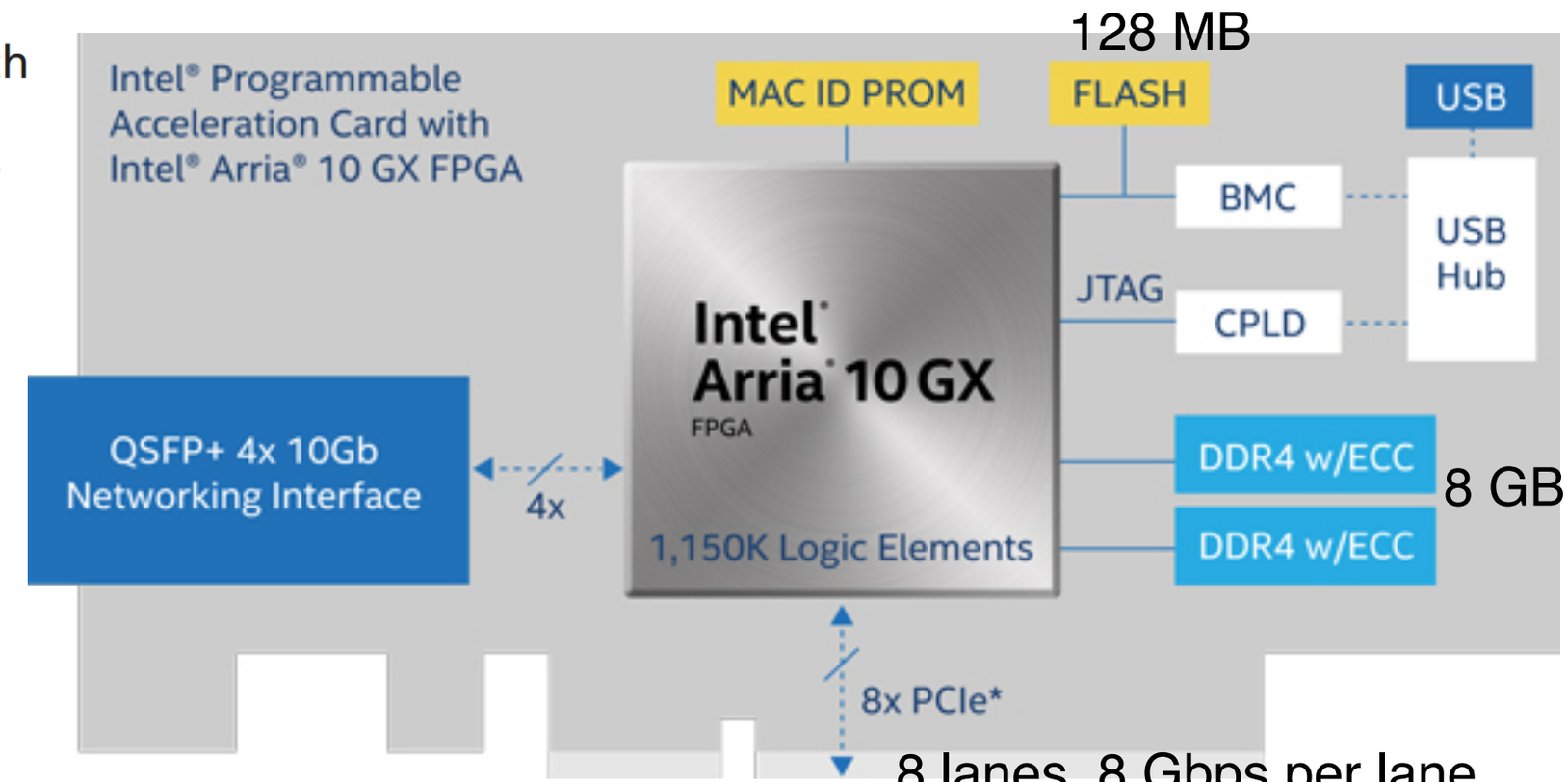
Gokhale et al. arXiv:1708.02579 (2017)

Intel, commercially available today

Targeted Workloads

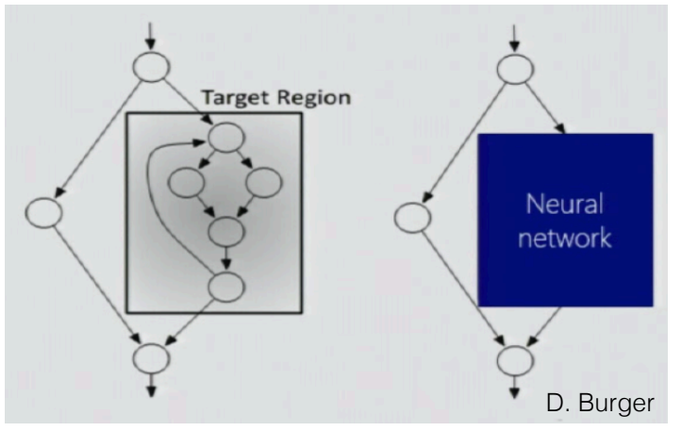
- Big data analytics
- Artificial intelligence
- Video transcoding
- Cyber security
- High-performance computing (HPC), such as genomics and oil and gas
- Financial technology, or FinTech

libraries becoming available



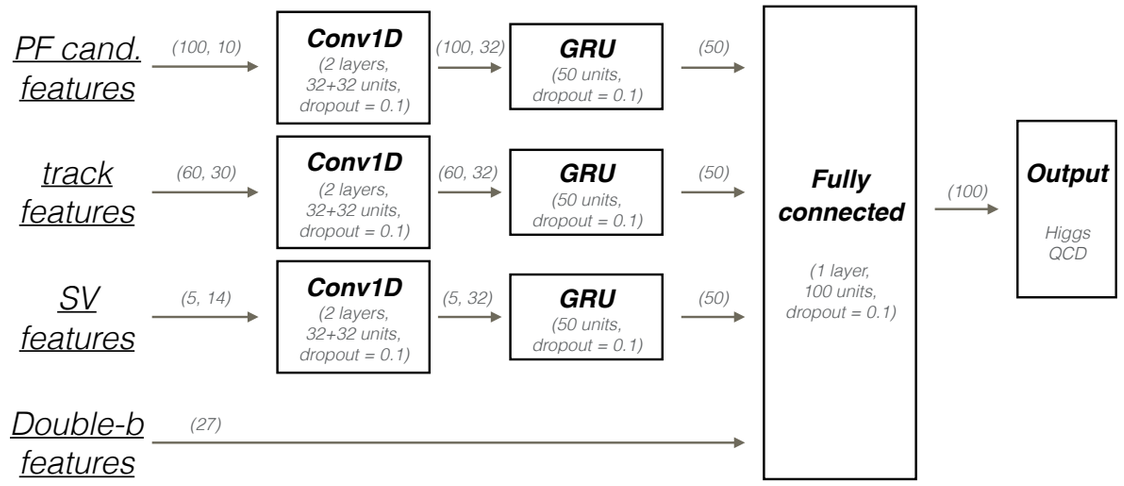
New Possibilities for LHC

- Hardware:
  - FPGA accelerators on-site for **HLT**
  - FPGA accelerators in **offline** computing resources
    - *Cloud: Microsoft, Amazon, etc.*
- New possibilities:
  1. Much larger networks possible
  2. Migrate upstream
     - *E.g. offline to HLT*
  3. Recast bottlenecks into ML problems



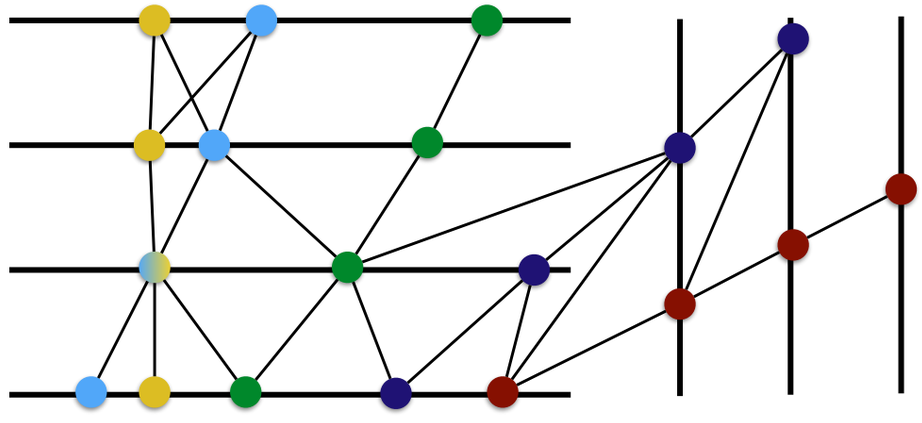
- *E.g. tracking, imagine algorithms in talks by Jean-Roch Vlimant and Steven Farrell done in FPGAs*

E.g. offline H(bb) tagger

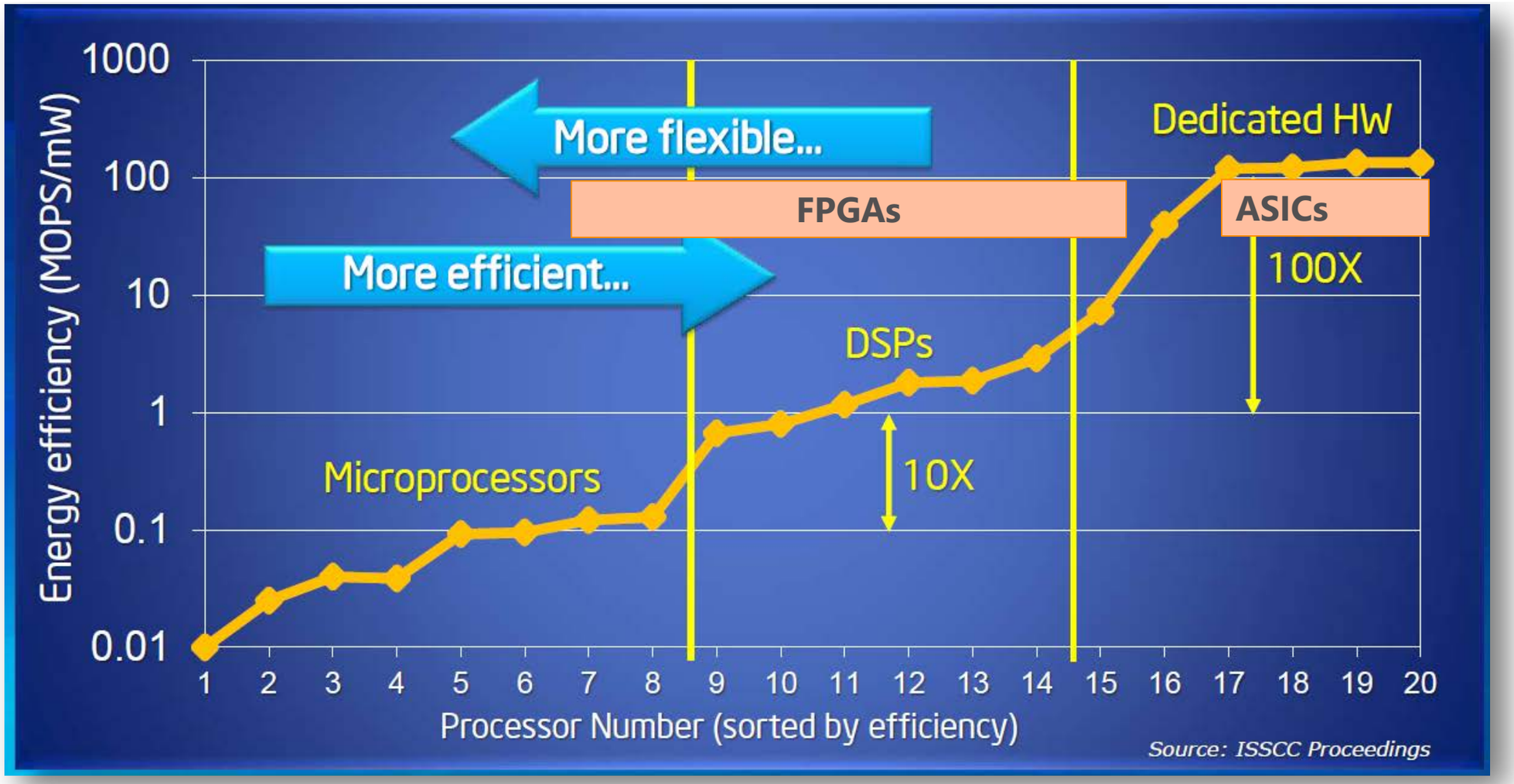


2.9k inputs, 60k parameters

E.g. graph NN for tracking

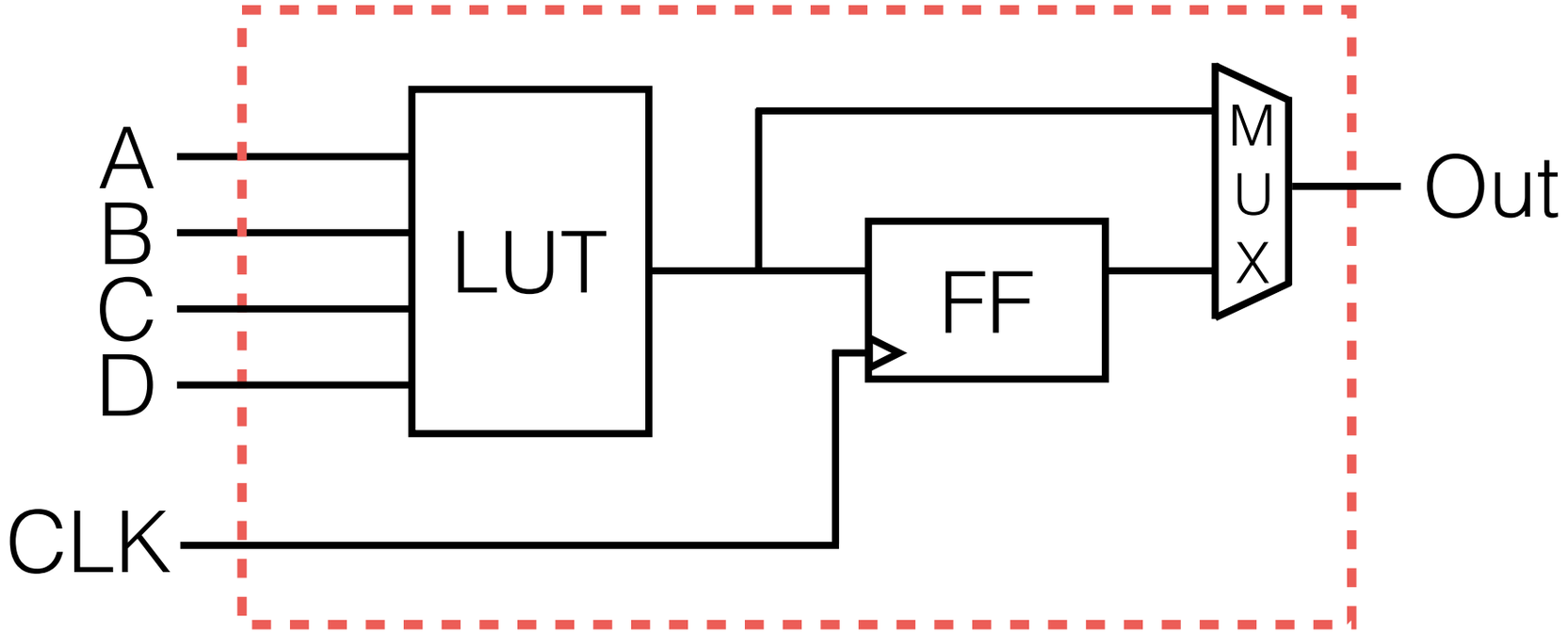


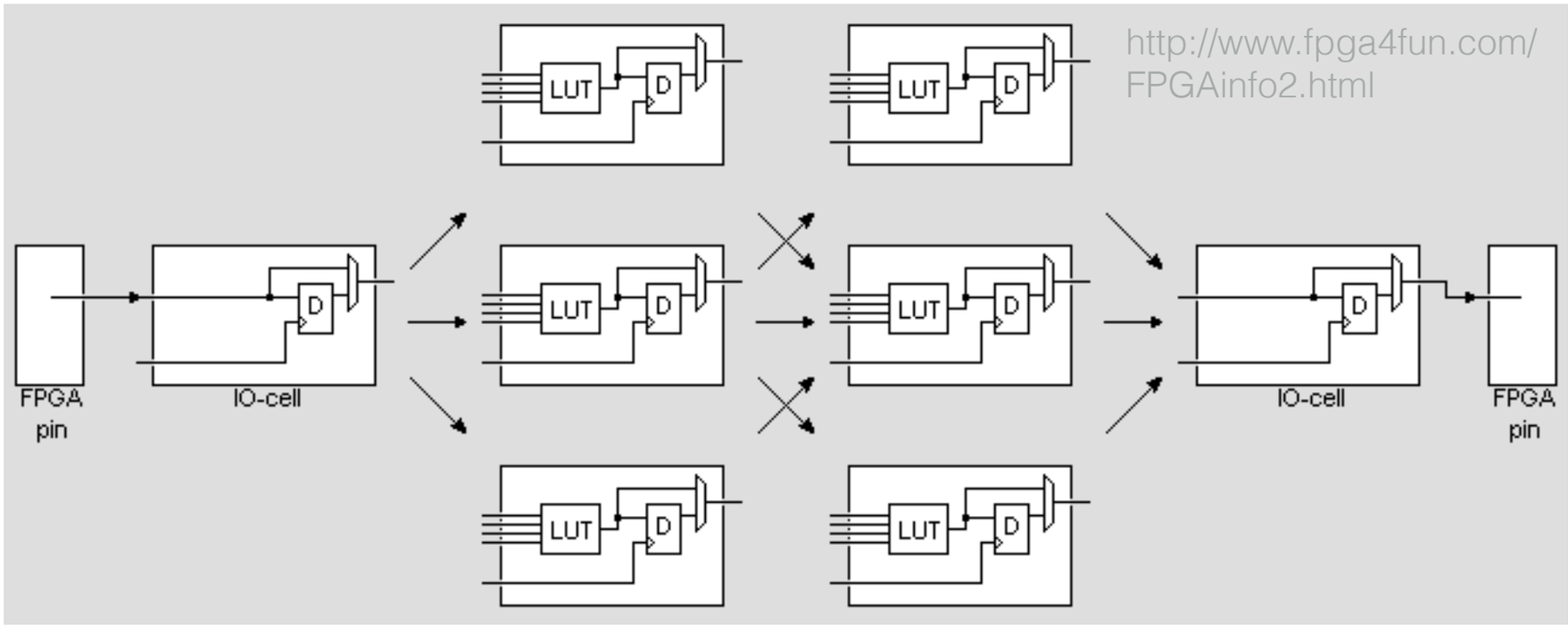
HEP.TrkX, https://indico.cern.ch/event/658267/contributions/2881175/attachments/1621912/2581064/Farrell_heptrkx_ctd2018.pdf



Source: Bob Broderson, Berkeley Wireless group (via Andrew Putnam)

Logic Cell


A
B
C
D
CLK
LUT
FF
MUX
Out


http://www.fpga4fun.com/
FPGAinfo2.html
LUT
D
FPGA
pin
IO-cell
IO-cell
FPGA
pin

Kintex® UltraScale™ FPGAs

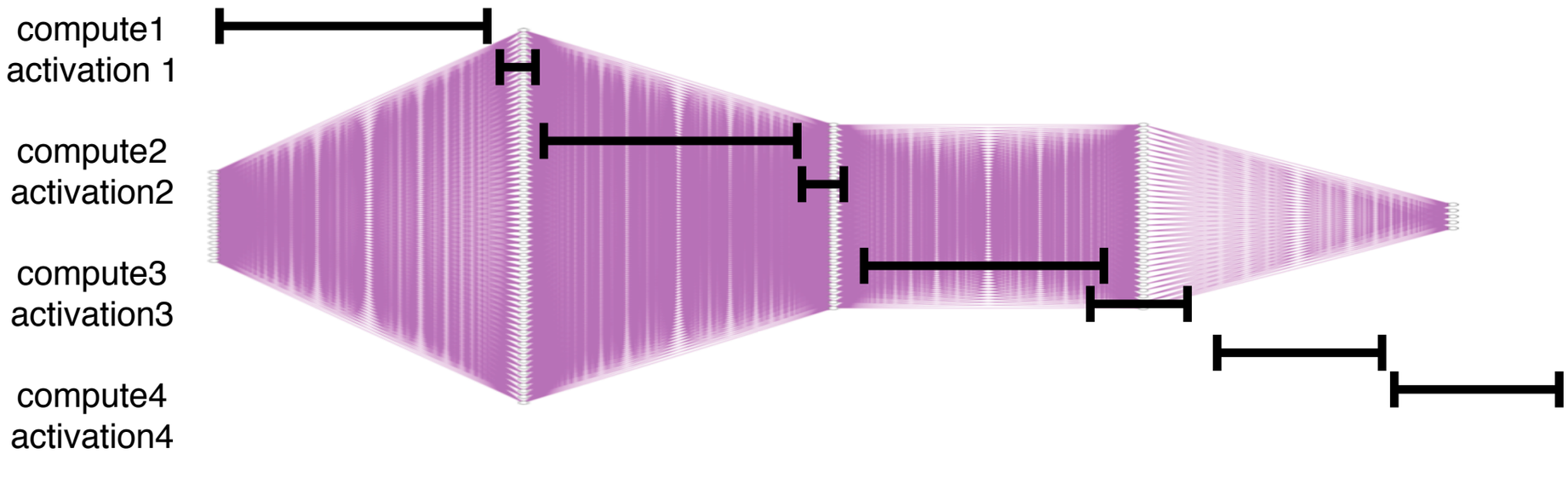
	Device Name	KU025[1]	KU035	KU040	KU060	KU085	KU095	KU115
Logic Resources	System Logic Cells (K)	318	444	530	726	1,088	1,176	1,451
	CLB Flip-Flops	290,880	406,256	484,800	663,360	995,040	1,075,200	1,326,720
	CLB LUTs	145,440	203,128	242,400	331,680	497,520	537,600	663,360
Memory Resources	Maximum Distributed RAM (Kb)	4,230	5,908	7,050	9,180	13,770	4,800	18,360
	Block RAM/FIFO w/ECC (36Kb each)	360	540	600	1,080	1,620	1,680	2,160
	Block RAM/FIFO (18Kb each)	720	1,080	1,200	2,160	3,240	3,360	4,320
	Total Block RAM (Mb)	12.7	19.0	21.1	38.0	56.9	59.1	75.9
Clock Resources	CMT (1 MMCM, 2 PLLs)	6	10	10	12	22	16	24
	I/O DLL	24	40	40	48	56	64	64
I/O Resources	Maximum Single-Ended HP I/Os	208	416	416	520	572	650	676
	Maximum Differential HP I/O Pairs	96	192	192	240	264	288	312
	Maximum Single-Ended HR I/Os	104	104	104	104	104	52	156
	Maximum Differential HR I/O Pairs	48	48	48	48	56	24	72
Integrated IP Resources	DSP Slices	1,152	1,700	1,920	2,760	4,100	768	5,520
	System Monitor	1	1	1	1	2	1	2
	PCIe® Gen1/2/3	1	2	3	3	4	4	6
	Interlaken	0	0	0	0	0	2	0
	100G Ethernet	0	0	0	0	0	2	0
	16.3Gb/s Transceivers (GTH/GTY)	12	16	20	32	56	64[2]	64
Speed Grades	Commercial	-1	-1	-1	-1	-1	-1	-1
	Extended	-2	-2 -3	-2 -3	-2 -3	-2 -3	-2	-2 -3
	Industrial	-1 -2	-1 -1L -2	-1 -1L -2	-1 -1L -2	-1 -1L -2	-1 -2	-1 -1L -2

	Package Footprint[3, 4, 5, 6]	Package Dimensions (mm)	HR I/O, HP I/O, GTH/GTY					
Footprint Compatible with Virtex® UltraScale Devices	A784[7]	23x23[8]		104, 364, 8	104, 364, 8			
	A676[7]	27x27		104, 208, 16	104, 208, 16			
	A900[7]	31x31		104, 364, 16	104, 364, 16			
	A1156	35x35	104, 208, 12	104, 416, 16	104, 416, 20	104, 416, 28		52, 468, 28	
	A1517	40x40				104, 520, 32	104, 520, 48		104, 520, 48
	C1517	40x40						52, 468, 40	
	D1517	40x40							104, 234, 64
	B1760	42.5x42.5					104, 572, 44	52, 650, 48	104, 598, 52
	A2104	47.5x47.5							156, 676, 52
	B2104	47.5x47.5						52, 650, 64	104, 598, 64
	D1924	45x45							156, 676, 52
	F1924	45x45					104, 520, 56		104, 624, 64

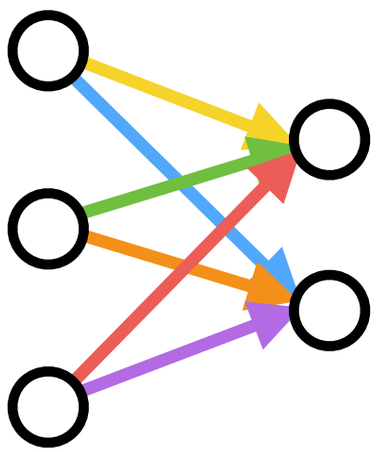
Notes:
1. Certain advanced configuration features are not supported in the KU025. Refer to the Configuring FPGAs section in DS890, *UltraScale Architecture and Product* Overview.
2. GTY transceivers in KU095 devices support data rates up to 16.3Gb/s.
3. Packages with the same package footprint designator, e.g., A2104, are footprint compatible with all other UltraScale devices with the same sequence. See the migration table for details on inter-family migration.
4. Maximum achievable performance is device and package dependent; consult the associated data sheet for details.
5. For full part number details, see the Ordering Information section in DS890, *UltraScale Architecture and Product* Overview.
6. See UG575, *UltraScale Architecture Packaging and Pinouts User Guide* for more information.
7. GTH transceivers in A784, A676, and A900 packages support data rates up to 12.5Gb/s.
8. 0.8mm ball pitch. All other packages listed 1mm ball pitch.

Latency and Pipelining in **hls4ml**

layers are sequential



computations within layer are parallelizeable



everything is pipelined

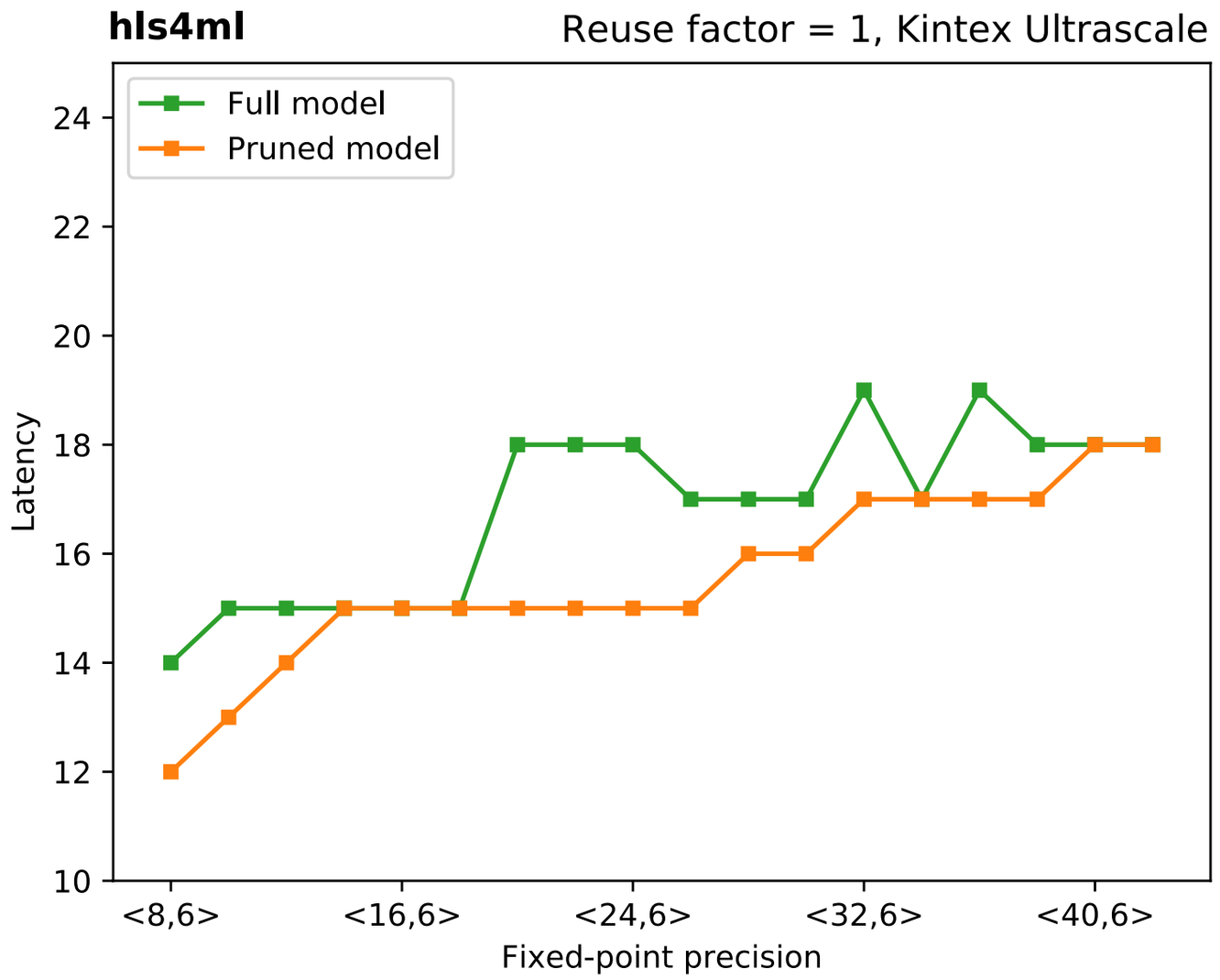
new inputs after “initial interval”

Compression

Network	Substructure (uncompressed)	Substructure (compressed)
AUC / Expected AUC	99.68%	99.55%
Parameters	4389	1338
Compression factor	-	3.3×
DSP48E	3329	954
Logic (LUT + FF)	263,234	88,797
Latency	75 ns	75 ns

Table 2: A summary of the vital statistics and HLS resource estimates of the uncompressed and compressed jet substructure tagging model with a network precision of fixed-point `<16,6>` and fully pipelined with clock frequency of 200 MHz synthesized on a Xilinx Kintex Ultrascale FPGA.

Latency vs Compression



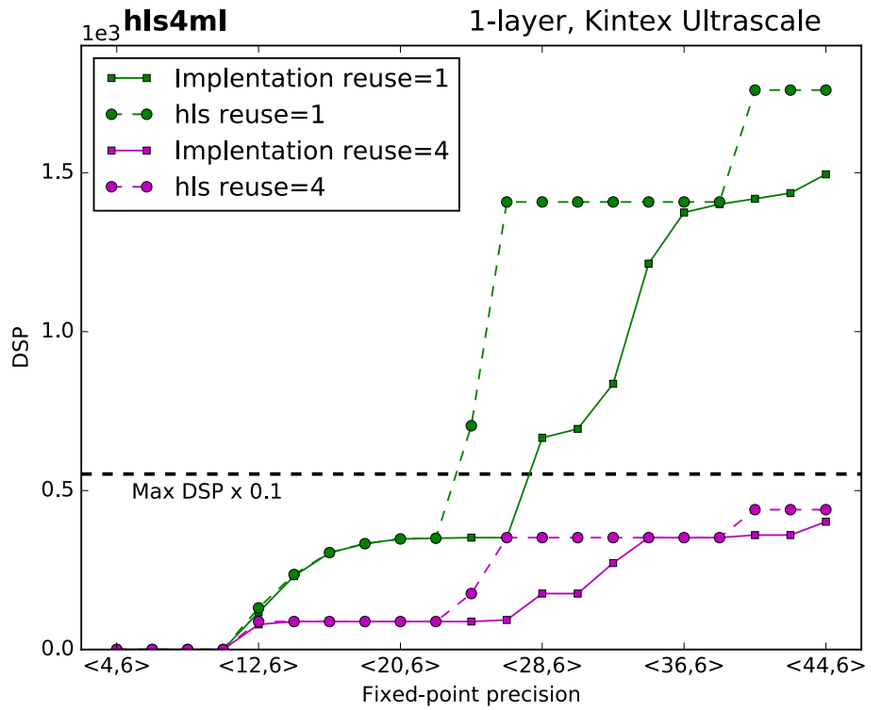


Figure 15: Comparsion of the DSP usage for the one-hidden-layer implemenation for the Xilinx Kintex Ultrascale FPGA as a function of the precision for various reuse factors.

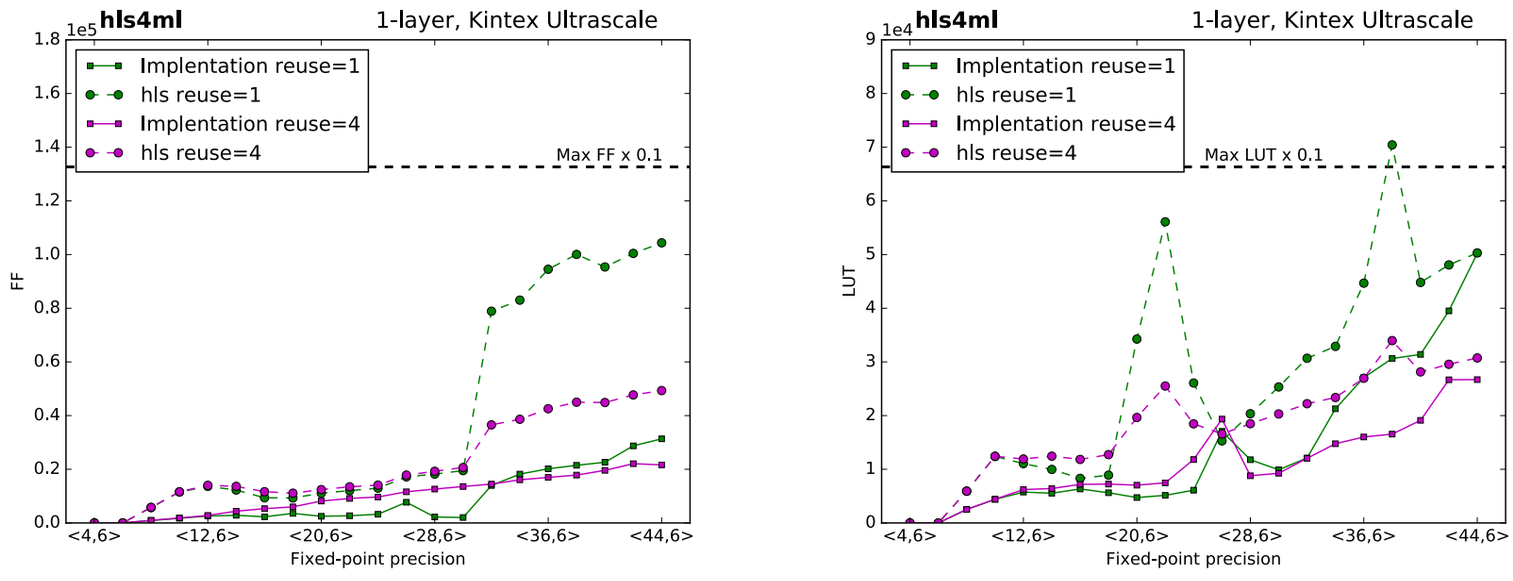


Figure 16: Comprsion of the FF performance (Left) and the LUT performance (Right) for the Kintex Ultrascale processor as a function of the precision for 1 and 4 reuse factors.