FPGA-based Implementations of Machine Learning Algorithms and the EPIC Approach

Philip Leong (梁恆惠) | Computer Engineering Laboratory
School of Electrical and Information Engineering,
The University of Sydney

THE UNIVERSITY OF SYDNEY

Computer Engineering Laboratory

- Focuses on how to use parallelism to solve demanding problems
  - Novel architectures, applications and design techniques using VLSI, FPGA and parallel computing technology
- Research
  - Reconfigurable computing
  - Machine learning
  - Nanoscale interfaces

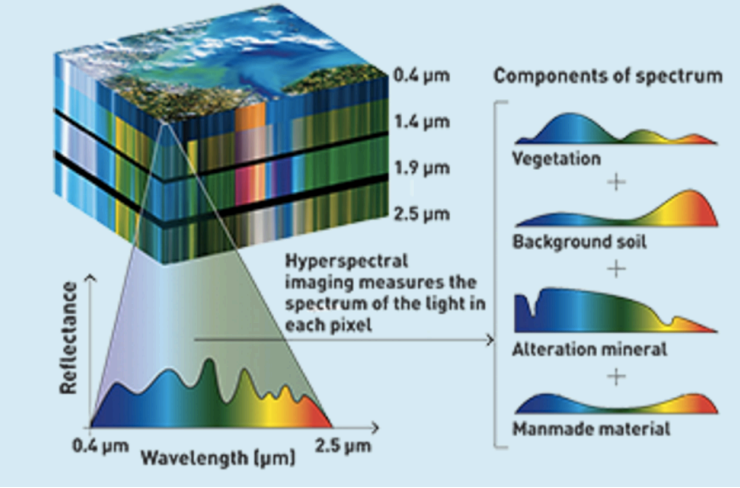
Motivation – Embedded ML

- **Systems of the future** can use FPGA-based ML to interpret and process data **adaptively in real-time**
  - Offer new capabilities
  - e.g. trigger oscilloscope or spectrum analyzer on an anomaly
  - e.g. battery power, high data rates, supercomputer performance in small form factor
- There are many applications that could benefit
  - test equipment
  - network monitors
  - prognostics and health management

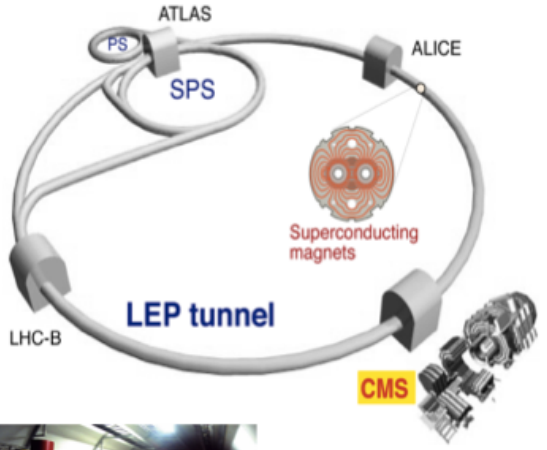
Throughput and Latency

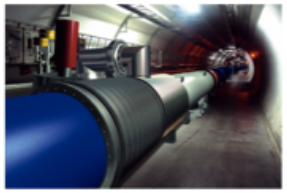
Challenges in measurement and control are becoming feasible

- Significant improvements in ML algorithms but cannot keep up with sources e.g. hyperspectral imager or wireless transceiver
- Need extremely high throughput



- In control applications we need low latency e.g. triggering data collection in Large Hadron Collider
- Need very low latency





Improvements in throughput and latency enable new applications!

Motivation for using FPGAs

- FPGAs offer an opportunity to provide ML algorithms with higher throughput and lower latency through
  - **E**xploration– easily try different ideas to arrive at a good solution
  - **P**arallelism – so we can arrive at an answer faster
  - **I**ntegration – so interfaces are not a bottleneck
  - **C**ustomisation – problem-specific designs to improve efficiency
- **Describe our work on implementations of ML that use these ideas**

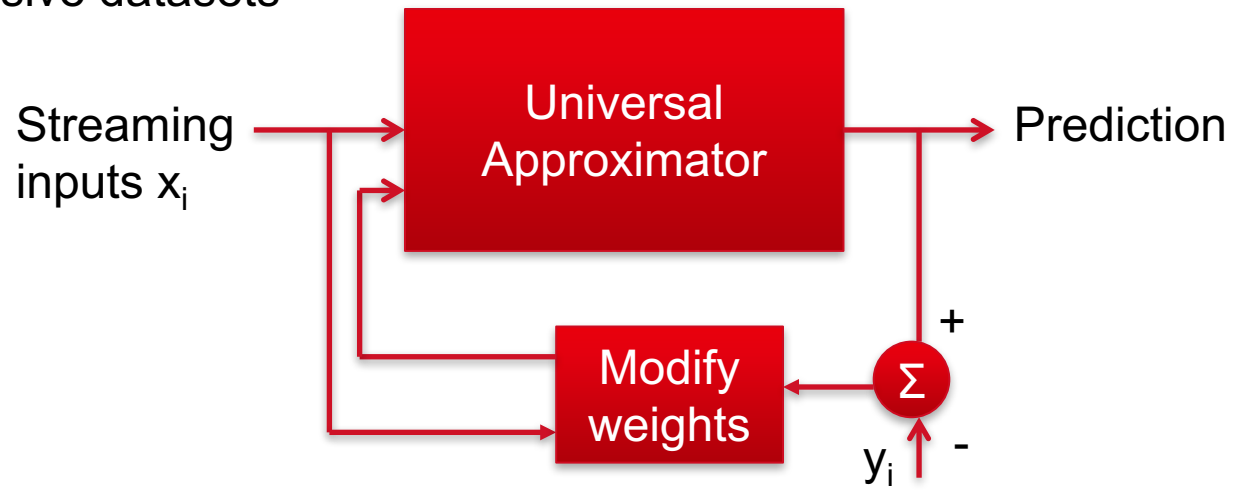
EPIC

- Exploration (Online kernel methods)
- Parallelisation
- Integration
- Customisation

Online Kernel Methods

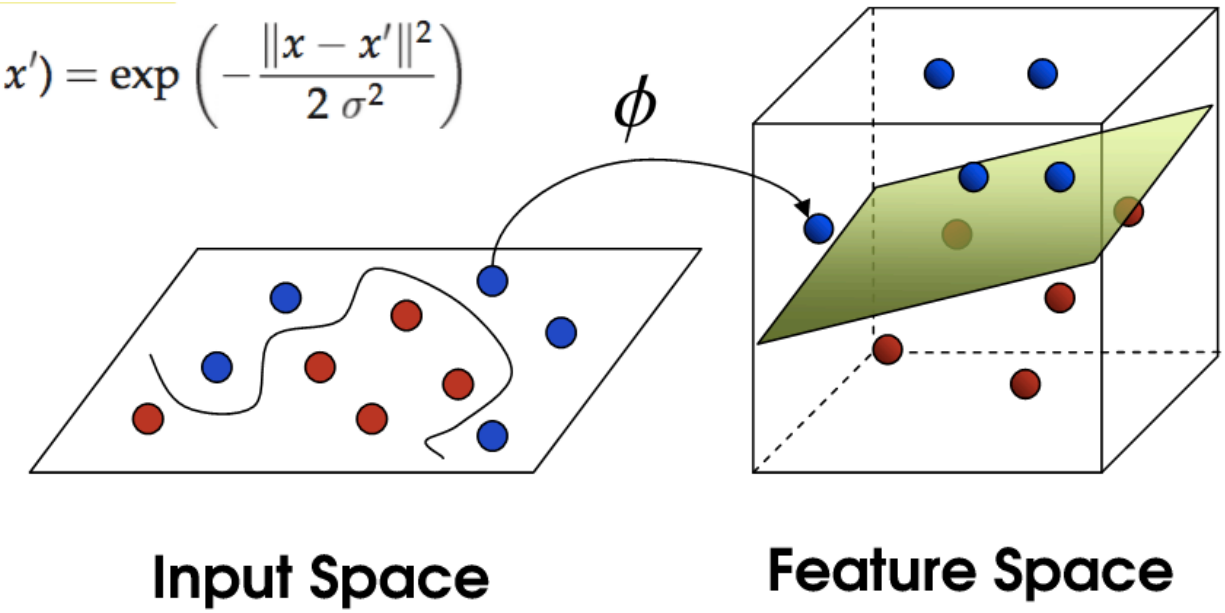
Examples are KLMS and KRLS

- Traditional ML algorithms are batch based
  - Make several passes through data
  - Requires storage of the input data
  - Not all data may be available initially
  - Not suitable for massive datasets
- Our approach: online algorithms
  - Incremental, inexpensive state update based on new data
  - Single pass through the data
  - Can be high throughput, low latency



Mapping to a Feature Space

$$k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2}\right)$$



- Choose high dimensional feature space (so easily separable)
- Use kernel trick to avoid computing the mapping (fast)
- Do regression/classification using $f(x_i) = \sum_{j=1}^{N} \alpha_j \kappa(x_i, v_j)$

THE UNIVERSITY OF SYDNEY

Kernel Methods Scaling

- In conventional kernel methods, computation and memory O(Nd)
  - N is dictionary size, d is input vector dimension
- BUT… N scales linearly with the dataset size

Random Kernel Expansions

Exact Kernel Methods

$$f(x) = \sum_{i=1}^{N} \alpha_i \kappa(x, d_i)$$

Random Kitchen Sinks

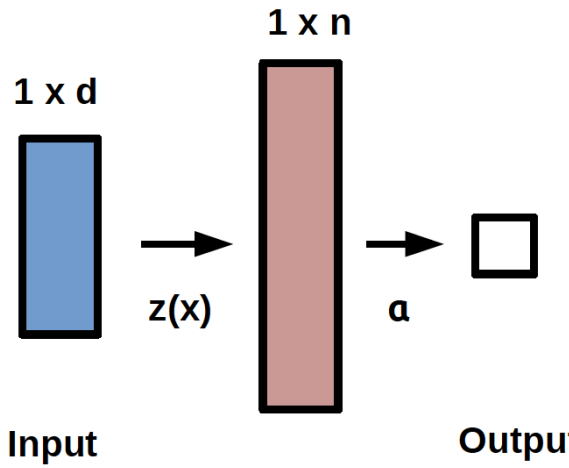
$$f(x) = \sum_{i=1}^{n} \alpha_i z(x)$$

$$z(x) = \frac{1}{\sqrt{n}} \cos(W x)$$

** Only for k(x,x') = k(x-x',0)

Define z(x):

- Approximates $\kappa(x, x')$
- Matrix-Vector + Non-Linear Activation (just like Multilayer Perceptron)
- W is **fixed** and **random**



Fastfood

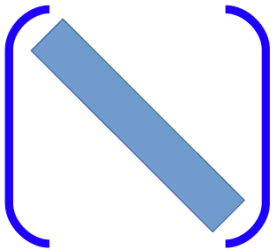
- Computes **z(x)** really quickly by replacing **Wx** with combinations of Diagonal and Hadamard matrices

$$z(x) = \frac{1}{\sqrt{n}} \cos(Vx), \quad where \quad Vx = [V_0 x, V_1 x, \ldots, V_j x]$$

$$V_j x = SHGPHBx$$

** Each $V_j x$ is an independent dxd transform

S, G, B



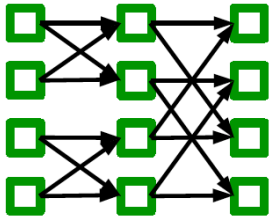
P

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

H

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$$

Fast Hadamard Transform (dxd)

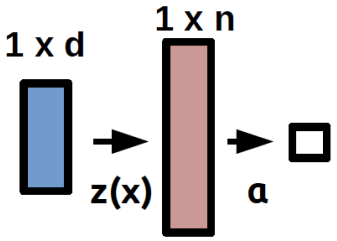


Memory = O(3n)

Time = O(n log d)

Architecture

- Hierarchical Systolic Array
- Linear combination of **n** basis functions, distributed across **p** PEs



$$f(x) = \sum_{i=1}^{n} \alpha_i z(x) = \sum^{h} \sum^{b} \sum_{i=1}^{k} \alpha_i z(x)$$

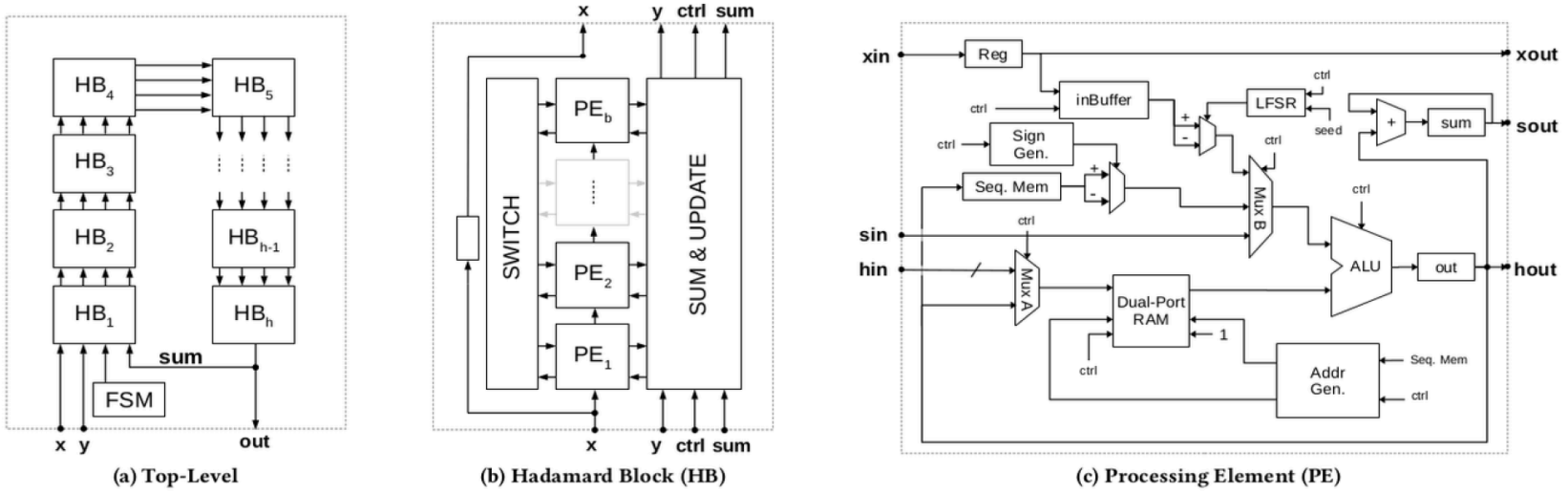
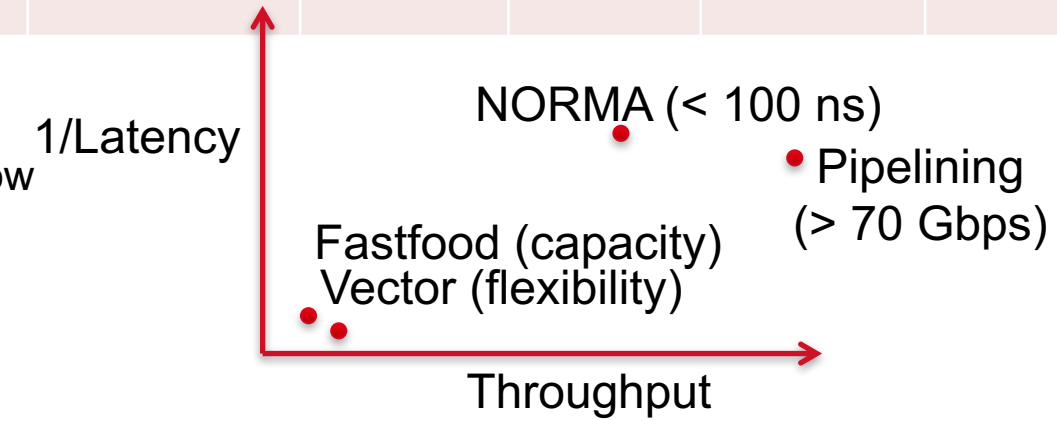


Figure 1: A hierarchical block diagram of the Fastfood processor

Kernel Methods

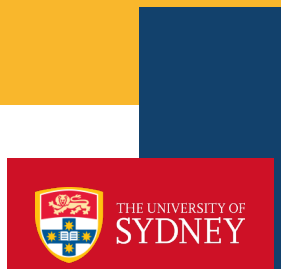
Impl.	d	N	Latency (cycles)	Fmax (MHz)	Time (ns)	CPU (ns)	Speed Up
Vector - Flexible (Stratix 5)	8	127	4396	157	28000	141000	5.1x
Pipeline - Throughput (Virtex 7)	8	16	207	314	**3.18**	940	296x
Braided - Latency (Virtex 7)	8	200	**13**	127	7.87	4025.8	511x
FASTFOOD - Capacity (Kintex Ultrascale)	8K	90.1K	16930	**508**	3388	17200	245x

- M – Input dimension
- N – Dictionary Size (or Sliding Window Size for KRLS)



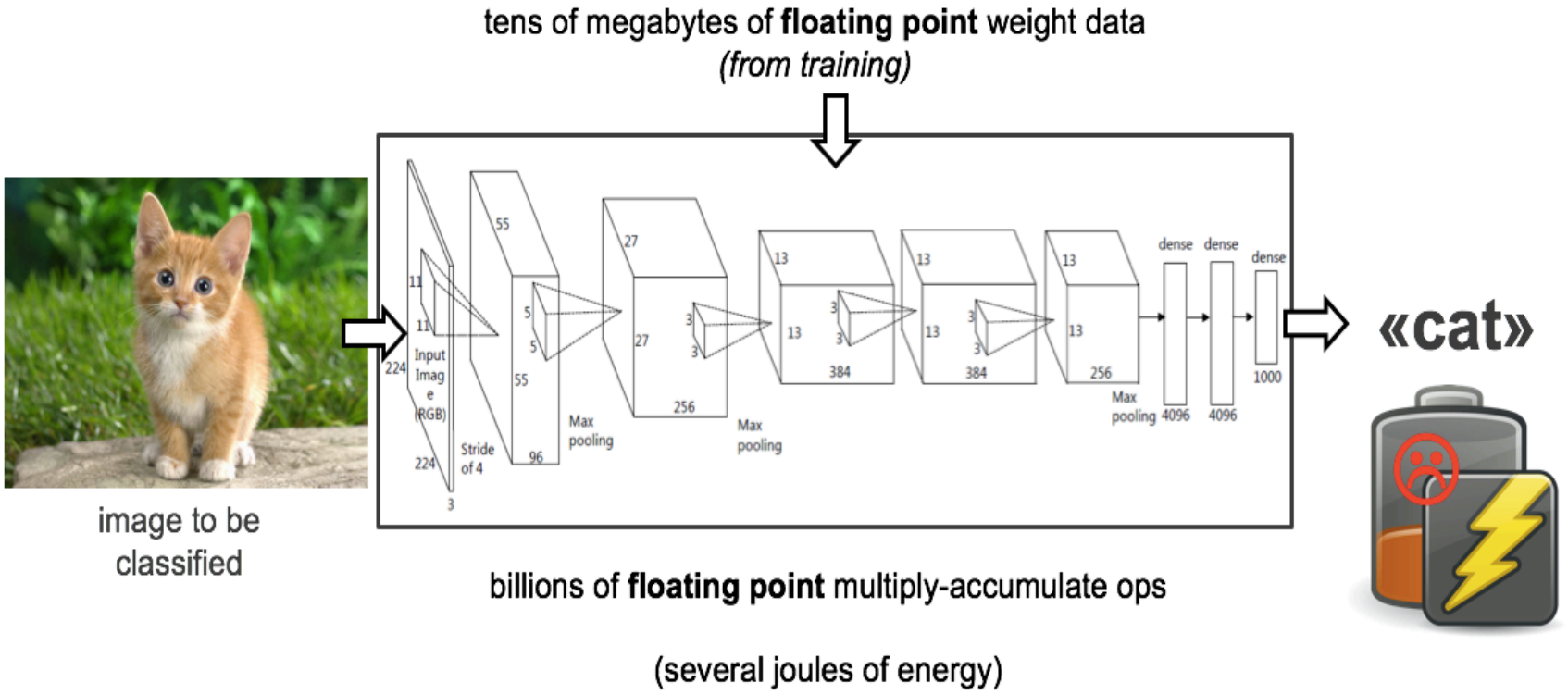
EPIC

- Exploration
- Parallelisation (Binarised Neural Network)
- Integration
- Customisation



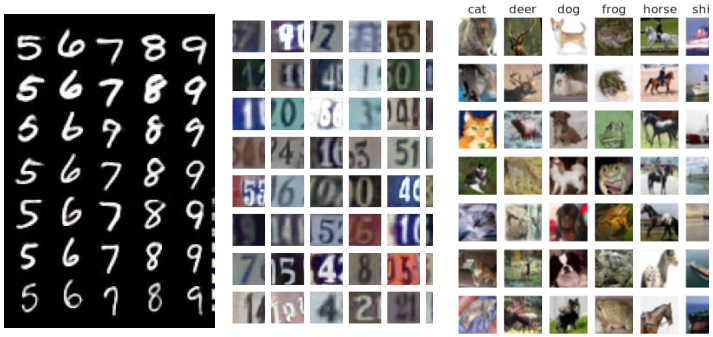
Inference with Convolutional Neural Networks

Slides from Yaman Umuroglu et. al., “FINN: A framework for fast, scalable binarized neural network inference,” FPGA’17



Binarized Neural Networks

- The extreme case of quantization
  - Permit only two values: +1 and -1
  - Binary weights, binary activations
  - Trained from scratch, not truncated FP



- Courbariaux and Hubara et al. (NIPS 2016)
  - Competitive results on three smaller benchmarks
  - Open source training flow
  - Standard “deep learning” layers
    - Convolutions, max pooling, batch norm, fully connected…

	MNIST	SVHN	CIFAR-10
Binary weights & activations	0.96%	2.53%	10.15%
FP weights & activations	0.94%	1.69%	7.62%
BNN accuracy loss	-0.2%	-0.84%	-2.53%

% classification error (lower is better)

Advantages of BNNs

Vivado HLS estimates on Xilinx UltraScale+ MPSoC ZU19EG

- *Much* smaller datapaths
  - Multiply becomes XNOR, addition becomes popcount
  - No DSPs needed, everything in LUTs
  - Lower cost per op = more ops every cycle
- *Much* smaller weights
  - Large networks can fit entirely into on-chip memory (OCM)
  - More bandwidth, less energy compared to off-chip

Precision	Peak TOPS	On-chip weights
1b	~66	~70 M
8b	~4	~10 M
16b	~1	~5 M
32b	~0.3	~2 M

200x

30x

- **fast** inference with **large BNNs**

Comparison

		Accuracy	FPS	Power (chip)	Power (wall)	kFPS / Watt (chip)	kFPS / Watt (wall)	Precision
FINN	MNIST, SFC-max	95.8%	12.3 M	7.3 W	21.2 W	1693	583	1
	MNIST, LFC-max	98.4%	1.5 M	8.8 W	22.6 W	177	269	1
	CIFAR-10, CNV-max	80.1%	21.9 k	3.6 W	11.7 W	6	2	1
	SVHN, CNV-max	94.9%	21.9 k	3.6 W	11.7 W	6	2	1
Prior Work	MNIST, Alemdar et al.	97.8%	255.1 k	0.3 W	-	806	-	2
	CIFAR-10, TrueNorth	83.4%	1.2 k	0.2 W	-	6	-	1
	SVHN, TrueNorth	96.7%	2.5 k	0.3 W	-	10	-	1

Max accuracy loss: ~3%

10 – 100x better performance

CIFAR-10/SVHN energy efficiency comparable to TrueNorth ASIC

EPIC

- Exploration
- Parallelisation
- Integration (radio frequency machine learning)
- Customisation

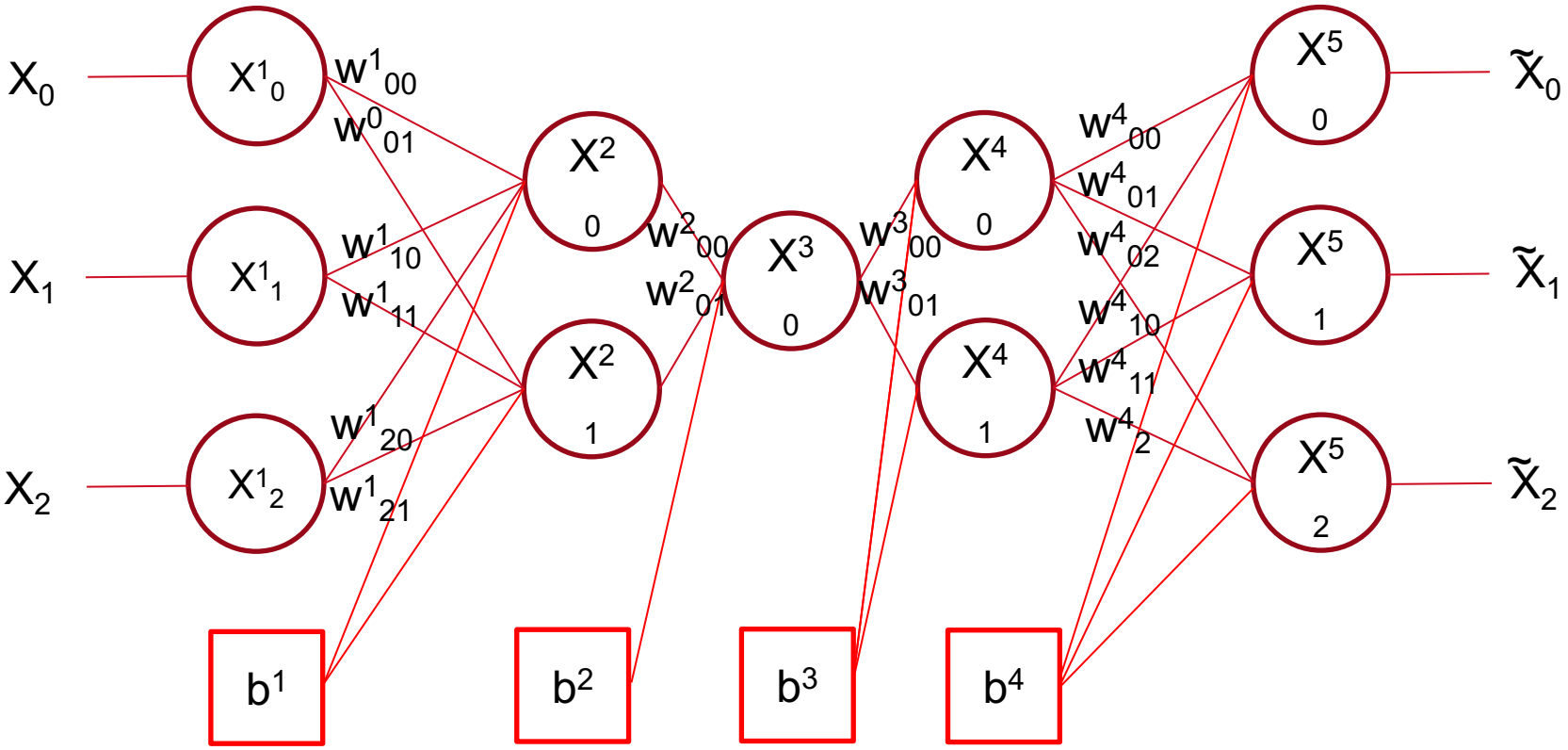
THE UNIVERSITY OF SYDNEY

Radio Frequency Machine Learning

- Processing radio frequency signals remains a challenge
  - high bandwidth and low latency difficult to achieve
- Autoencoder to do anomaly detection

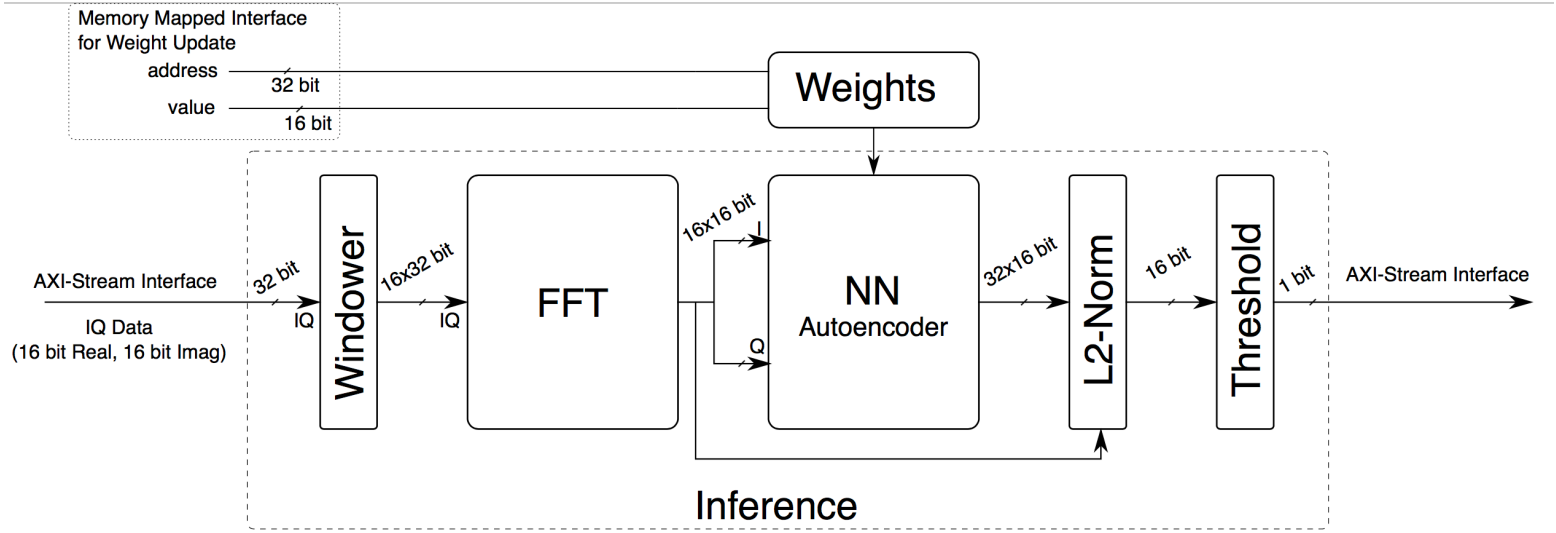
Autoencoder

Train so $\tilde{x}$ x (done in an unsupervised manner)



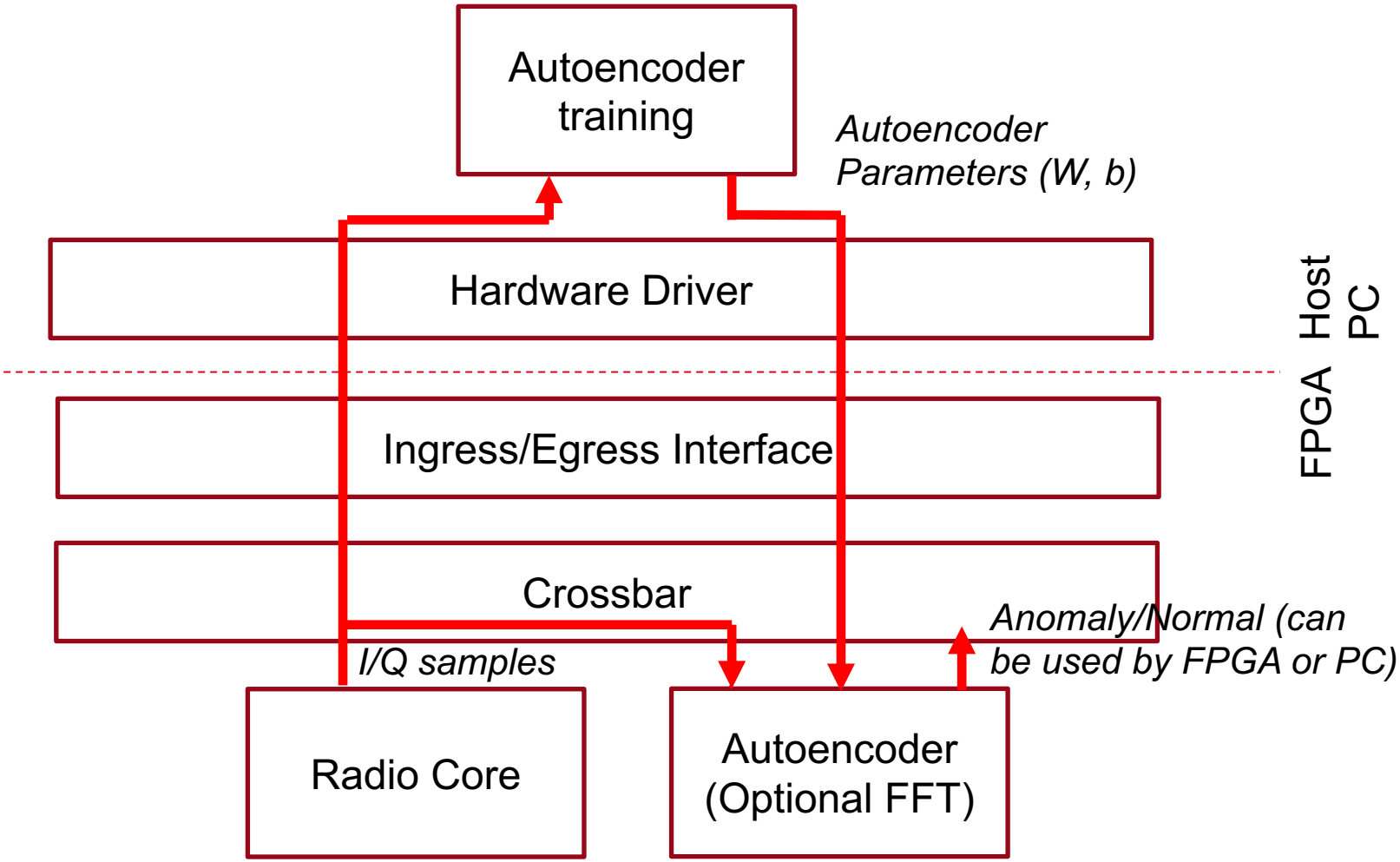
Autoencoder learns “normal” representation

- Anomaly if distance between autoencoder output and input large
- FPGA has sufficiently high performance to process each sample of waveform at 200 MHz!
  - This minimises latency and maximises throughput
  - Weights trained on uP and updated on FPGA without affecting inference

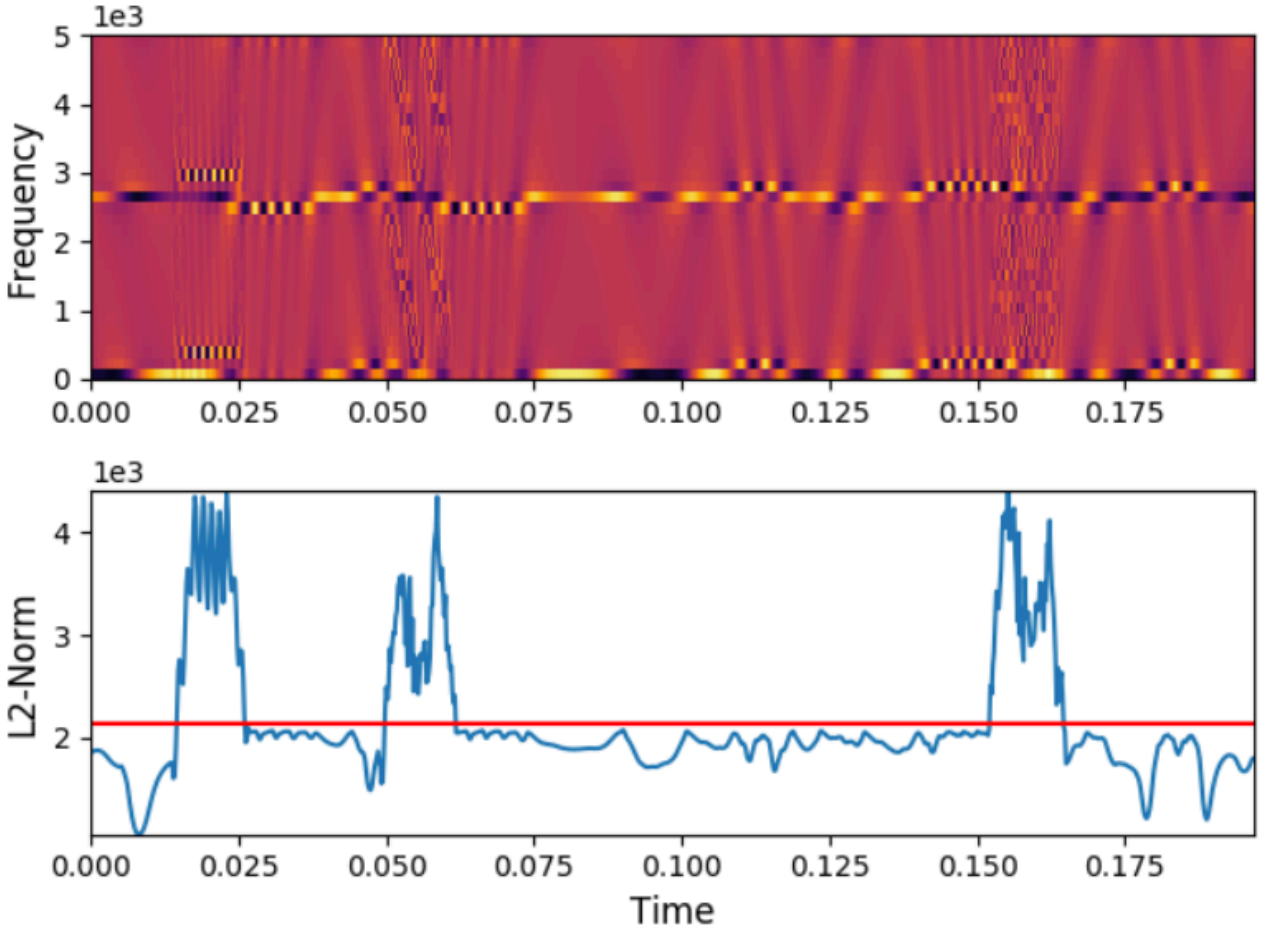


Software Defined Radio Architecture

Implemented on Ettus X310 platform



Example


1e3
5
4
3
2
1
0
Frequency
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
1e3
4
3
2
L2-Norm
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Time

Performance (XC7K410T)

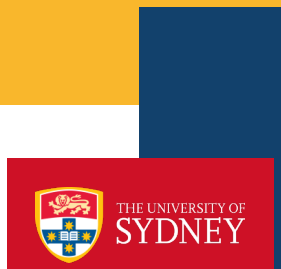
Typical SDR latency >> 1 ms

Module	II	Latency (cycles)	BRAM	DSP	FF	LUT
Windower	1	0	0	0	1511	996
FFT	1	8	0	48	4698	2796
NN	1	17	4	1280	213436	13044
L_2-Norm	1	4	0	32	1482	873
Thres	1	0	0	0	3	21
Weight Update	258	257	0	0	21955	4528
Inference (FFT+NN)	1	37	1068	1360	241522	45448
Inference (NN)	1	29	1068	1312	236824	42652
Total	N/A	N/A	1068	1360	263477	49976
Total Util.	N/A	N/A	67%	88%	51%	19%

Operation	Throughput	Latency
Inference(FFT+NN)	5ns	185ns
Inference(NN)	5ns	105ns
Weight Update	1290ns	1285ns

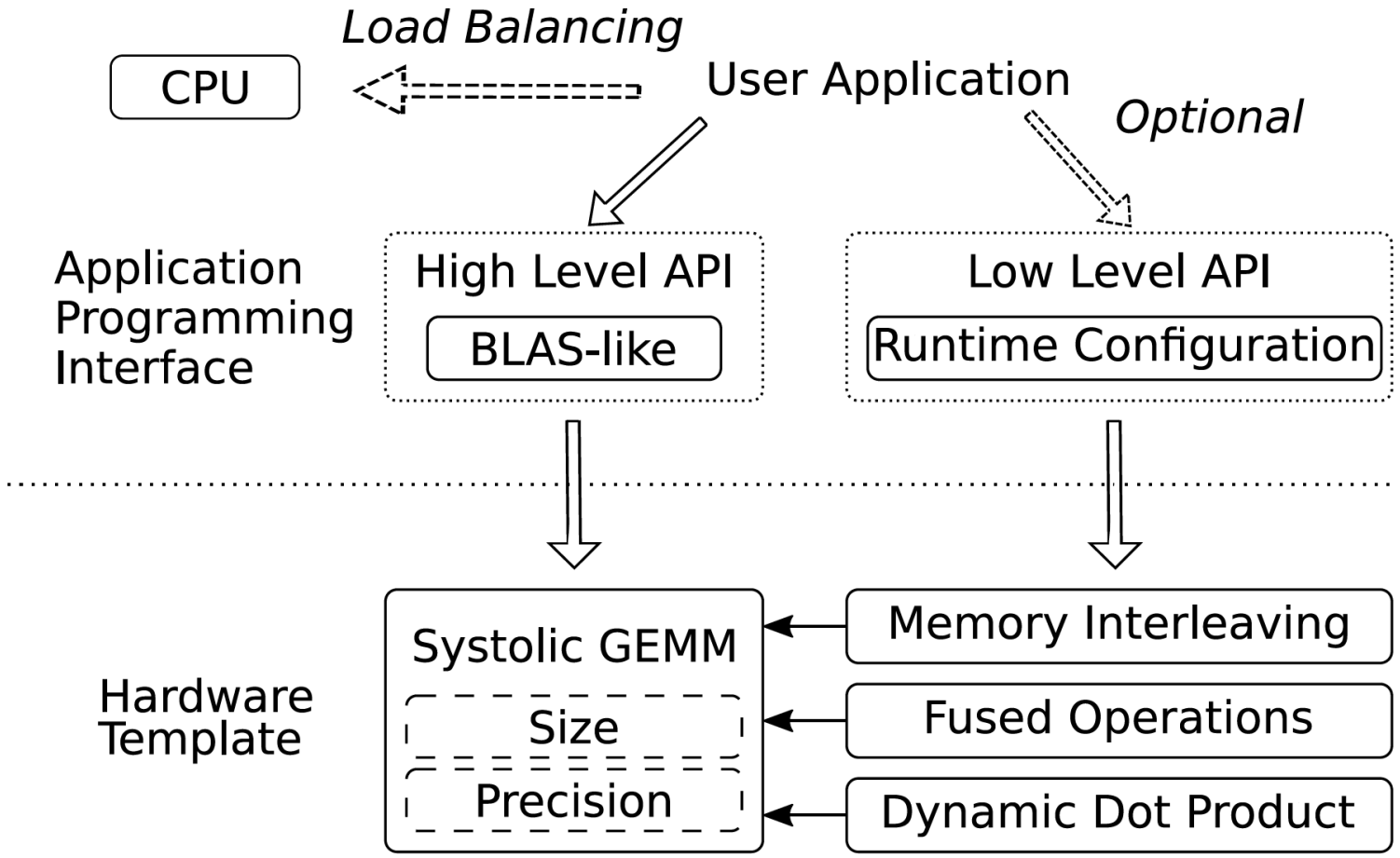
EPIC

- Exploration
- Parallelisation
- Integration
- Customisation (Matrix Multiplication on Intel Harp v2)



Customizable Matrix Multiplication Framework

A C++ API and hardware template GEMM on (Xeon+FPGA) Harp v2



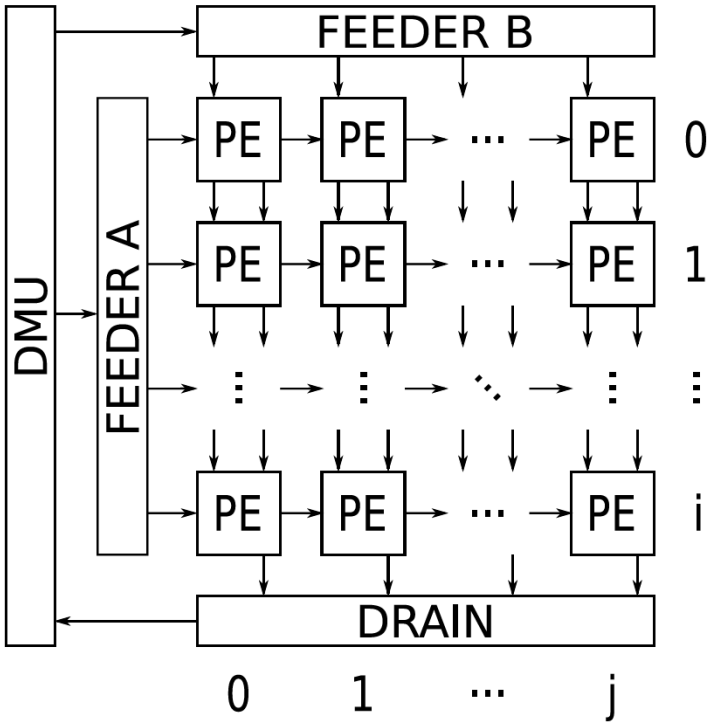
API and Tunable Options

```
void gemm(trans a, trans b, int m, int n, int k, float
    alpha, float* a, int lda, float* b, int ldb, float
    beta, float* c, int ldc);
```

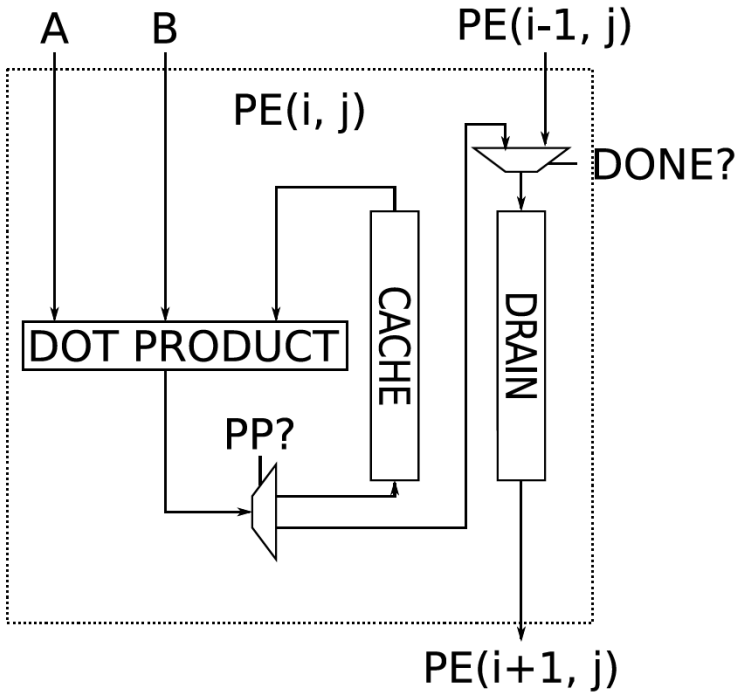
```
template<typename T1, typename T2>
fpga_gemm<T1,T2>::fpga_gemm(
uint32_t a_rows, unit32_t b_cols, unit32_t common,
uint32_t i_a_lead_interleave,
uint32_t i_b_lead_interleave,
uint32_t i_feeder_interleave,
float i_alpha, float i_beta, GEMM_MODE i_mode);
```

Parameters	Type	Options
Systolic Array Size (Sec. 4)	C	*Logic & Memory Limited
Precision (Sec. 4.1)	C	FP32, INT16, INT8, INT4, Ternary, Binary
Accumulator Width (Sec. 4.1)	C	*Logic & Memory Limited
Interleaving (Sec. 4.2)	C&R	*Memory Limited
Fused Ops (Sec. 5.1)	C&R	Scaling, Batch Norm, Clip, Rounding, ReLU

Hardware Template

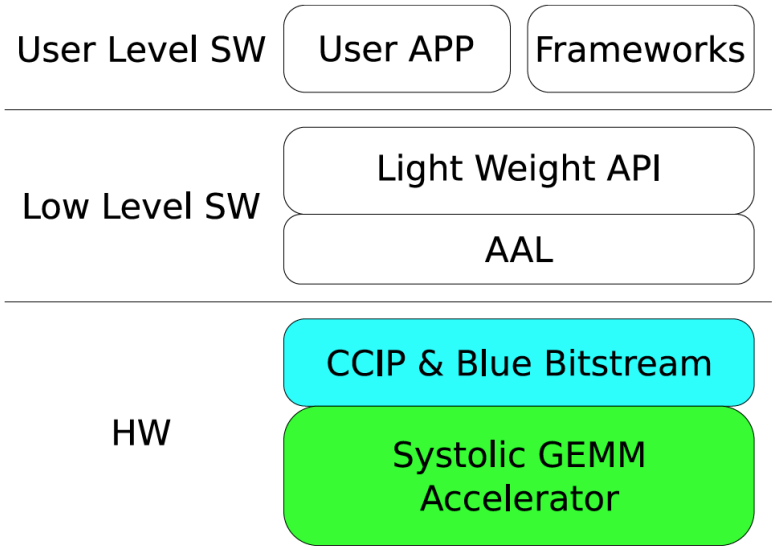


(a) Systolic Array

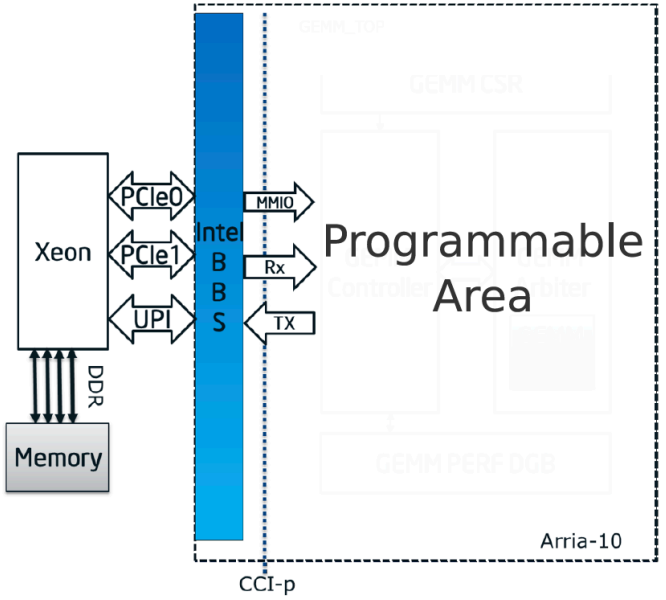


(b) Processing Element

System Interface



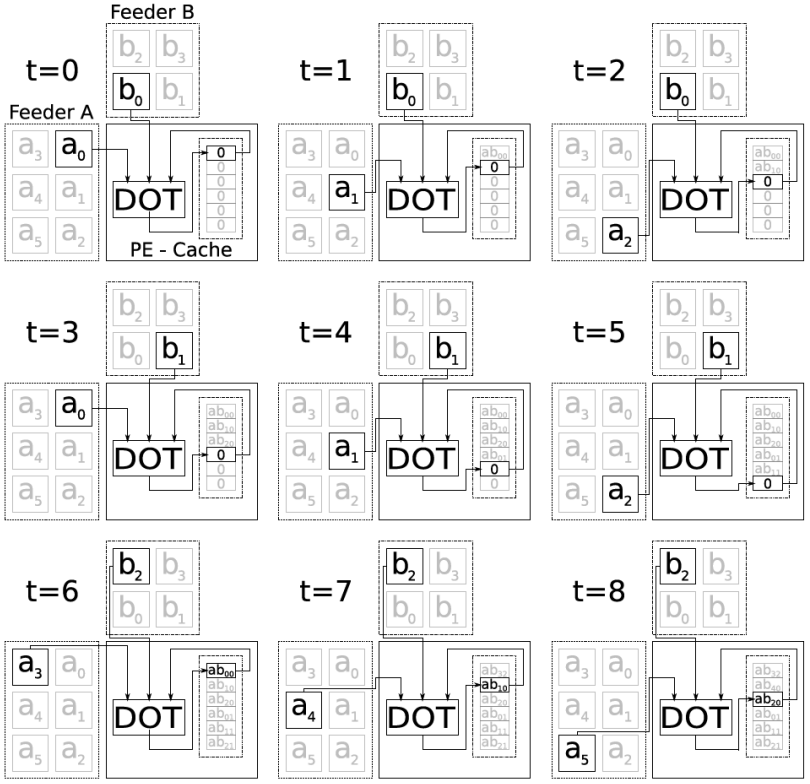
(a) Software and Hardware Stack



(b) Intel HARPv2 Platform

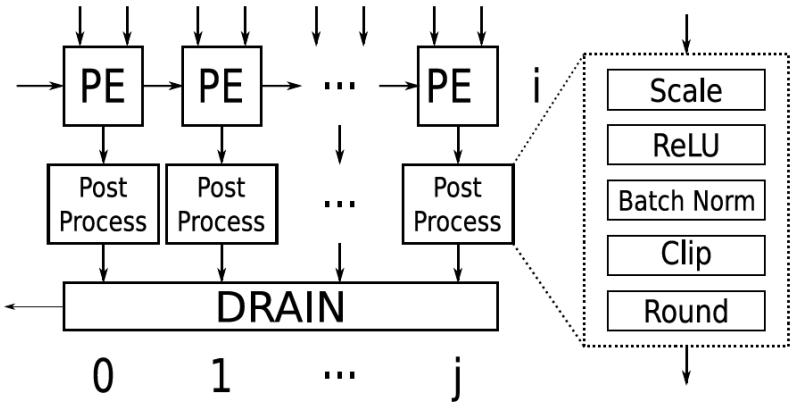
Interleaving

- Apply blocking and interleaving to maximise data reuse (2.7-4x improvement for neural networks)

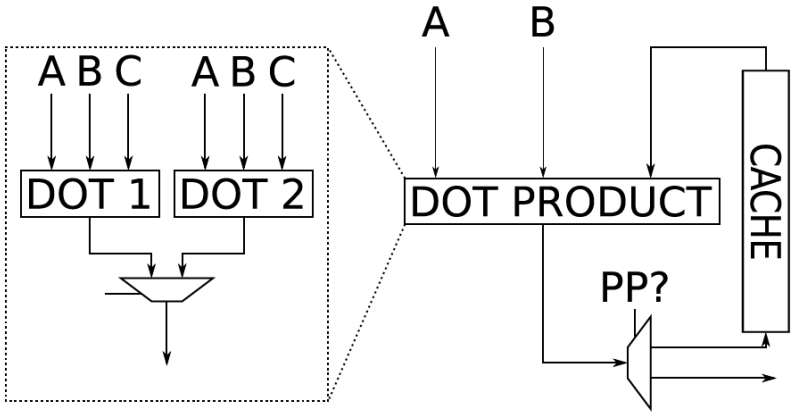


Fused Operations for DNNs

- Fused operations allow bandwidth and CPU post-processing workload to be reduced
- Dynamic dot product allows switching between types at runtime e.g. bin-bin and fp32-fp32, avoiding transfers

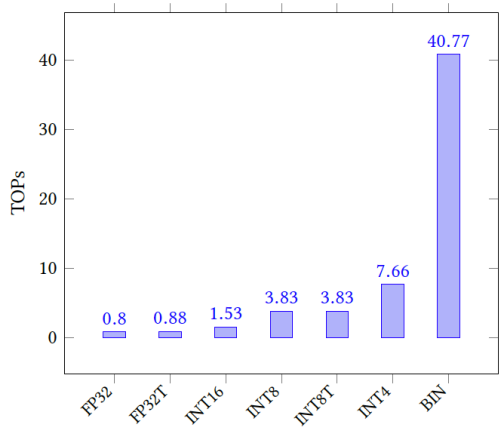


(a) Fused Operations

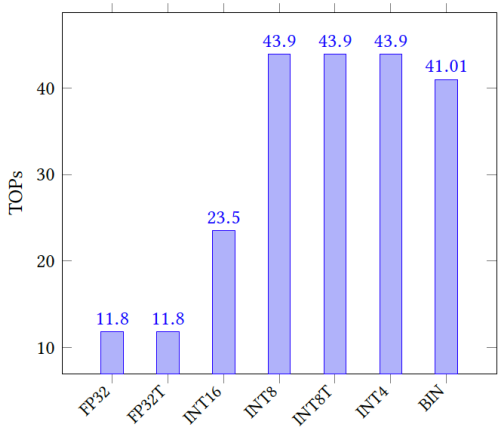


(b) Dynamic Dot Product Switching

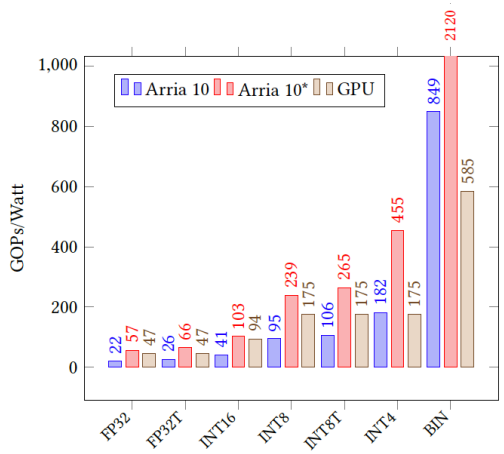
Matrix Multiplication Performance



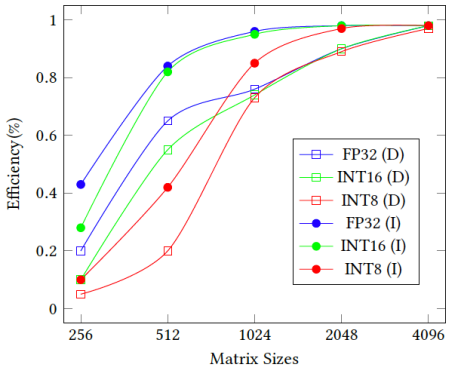
(a) Arria 10 Peak Performance



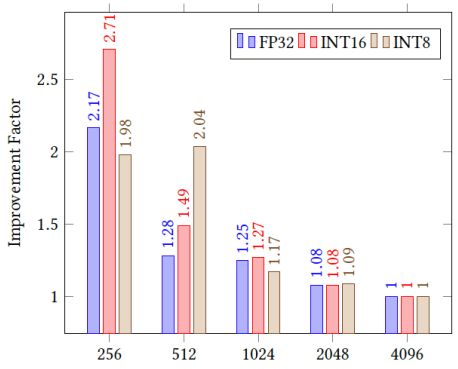
(b) NVIDIA Pascal Titan X Peak Performance



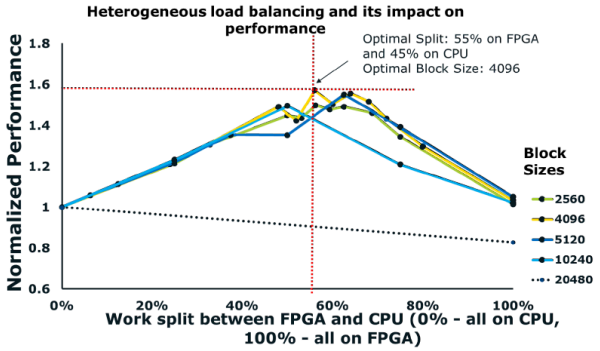
(c) Performance per Watt



(d) GEMM Efficiency vs Matrix Size



(e) Memory Interleaving Improvements



(f) Load Balancing

THE UNIVERSITY OF SYDNEY

Mixed Precision Inference and Training

Layer	Location		Type	
	Forward	Backward	Forward	Backward
...	...	...	...	...
conv	FPGA	FPGA+CPU	BINxBIN	FPxBIN
c&r	FPGA	N/A	INT	STE
relu	FPGA	FPGA+CPU	INT	FP
norm	FPGA	CPU	FP	FP
pool	CPU	CPU	FP	FP
...	...	...	...	...
fc	CPU	CPU	FPxFP	FPxFP
prob	CPU	CPU	FP	FP

BNN Inference and Backprop Peak Performance

Impl.	BINxBIN (TOPs)	FPxFP (TFLOPs)	FPxBIN (TFLOPs)	Forward (ms)	Backward (ms)	Total (ms)
SW	-	-	-	421	471	892 (1x)
(1)	40.77	0	0	158 (F)	471 (C)	630 (1.41x)
(2)	25.4	0.8	0.8	164 (F)	537 (F)	702 (1.23x)
(3)	25.4	0	0.88	165 (F)	502 (F)	668 (1.33x)
(2H)	25.4	1.4	1.4	163 (F+C)	369 (F+C)	533 (1.67x)

Where possible, the operation was performed on the FPGA (F), the 14 core CPU (C) or using heterogeneous load balancing (F+C).

(1) a single BINxBIN, (2) a BINxBIN and FPxFP, (2H) a version of (2) that performs the FPxFP GEMM utilizing heterogeneous load balancing and finally (3) a BINxBIN and FPxBIN.

THE UNIVERSITY OF SYDNEY

BNN Inference Performance

Device	FPGA				GPU			
	TOPs	GOPS/W	IPS	IPS/W	TOPS	GOPs/W	IPS	IPS/W
AlexNet	31.54	657.27	1610	33.54	37.60	568.09	1626	25.02
VGGNet	31.18	649.67	114	2.39	35.85	522.59	121	1.78

	[26]	[11]	[21]	Our Work
Platform	Zynq z7045	Kintex US KU115	Arria 10 GX1150	Arria 10 GX1150
Logic Elements (LEs)	350K	1,451K	1,150K	1,150K
Power (W)	11.3	41	-	48
TOPs (Peak)	11.612	14.8	25	40.77
MOPs / LE	33.17	10.19	-	35.45
GOPs / Watt	1027.68	360.97	-	849.38

EPIC

- Exploration (Online kernel methods)
- Parallelisation
- Integration
- Customisation

Conclusion

- **Exploration**
  - Kernel methods optimised using different algorithms, mathematical techniques, computer architectures, arithmetic
- **Parallelism**
  - Increase parallelism by reducing precision
  - Keep weights on-chip to devote more hardware to arithmetic
- **Integration**
  - In radio frequency, this allows latency to be reduced by 4 orders of magnitude
- **Customisation**
  - Supplement conventional matrix multiplication to support DNN implementation

- **FPGAs can greatly assist with the implementation of intelligent sensing**
  - Learning & inference at 70 Gbps
  - Learning & inference with 100 ns latency
  - Image processing @ 12.3 Mfps
  - Multimodal measurements
- **Radio frequency anomaly detector**
  - We are using this to predict physical and media access layer protocols
  - Could also be used as a novel diagnostic instrument - monitor RF output of electronic equipment, detect anomalies

THE UNIVERSITY OF SYDNEY

Open Source Materials

- LSTM using HLS tutorial
  - https://github.com/phwl/hlslstm
- Kernel methods code e.g. braiding
  - https://github.com/da-steve101/chisel-pipelined-olk
- FINN - can do trillions of binary operations per second
  - https://github.com/Xilinx/BNN-PYNQ

References

(all available from http://phwl.org/papers/)

- [E] Sean Fox, David Boland, and Philip H.W. Leong. FPGA Fastfood - a high speed systolic implementation of a large scale online kernel method. To appear FPGA18.
- [P] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. FINN: A framework for fast, scalable binarized neural network inference. In *Proc. FPGA*, pages 65–74, 2017. Source code available from https://github.com/Xilinx/BNN-PYNQ. (doi:10.1145/3020078.3021744)
- [I] Anomaly detector (unpublished but preprint available)
- [C] Duncan Moss, Srivatsan Krishnan, Eriko Nurvitadhi, Piotr Ratuszniak, Chris Johnson, Jaewoong Sim, Asit Mishra, Debbie Marr, Suchit Subhaschandra, and Philip H.W. Leong. A customizable matrix multiplication framework for the intel HARPv2 platform. To appear *FPGA*18.

Thank you!

Philip Leong (philip.leong@sydney.edu.au)
http://www.sydney.edu.au/people/philip.leong