A Fully Pipelined Kernel Normalised Least Mean Squares Processor For Accelerated Parameter Optimisation

Nicholas J. Fraser, Duncan J.M. Moss, JunKyu Lee, Stephen Tridgell, Craig T. Jin and Philip H.W. Leong

September 3, 2015

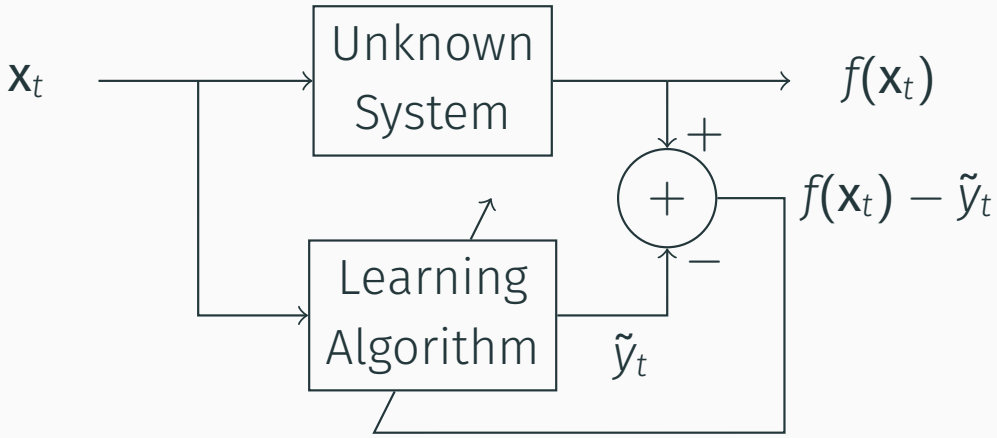
CELab, University of Sydney, Australia

Overview

Contributions:

- A fully pipelined core implementing KNLMS.
- A complete PCIe system.
- Large speedups over previous designs.
- Floating point and *fused* arithmetic designs - details in the paper!

Machine learning - creating models to fit data.



Two common problems arise:

- How do we decide which algorithm to use?
- Given an algorithm, how to do we configure it?

Motivation: KNLMS Configuration

Parameter selection can be the difference between a good and bad models.

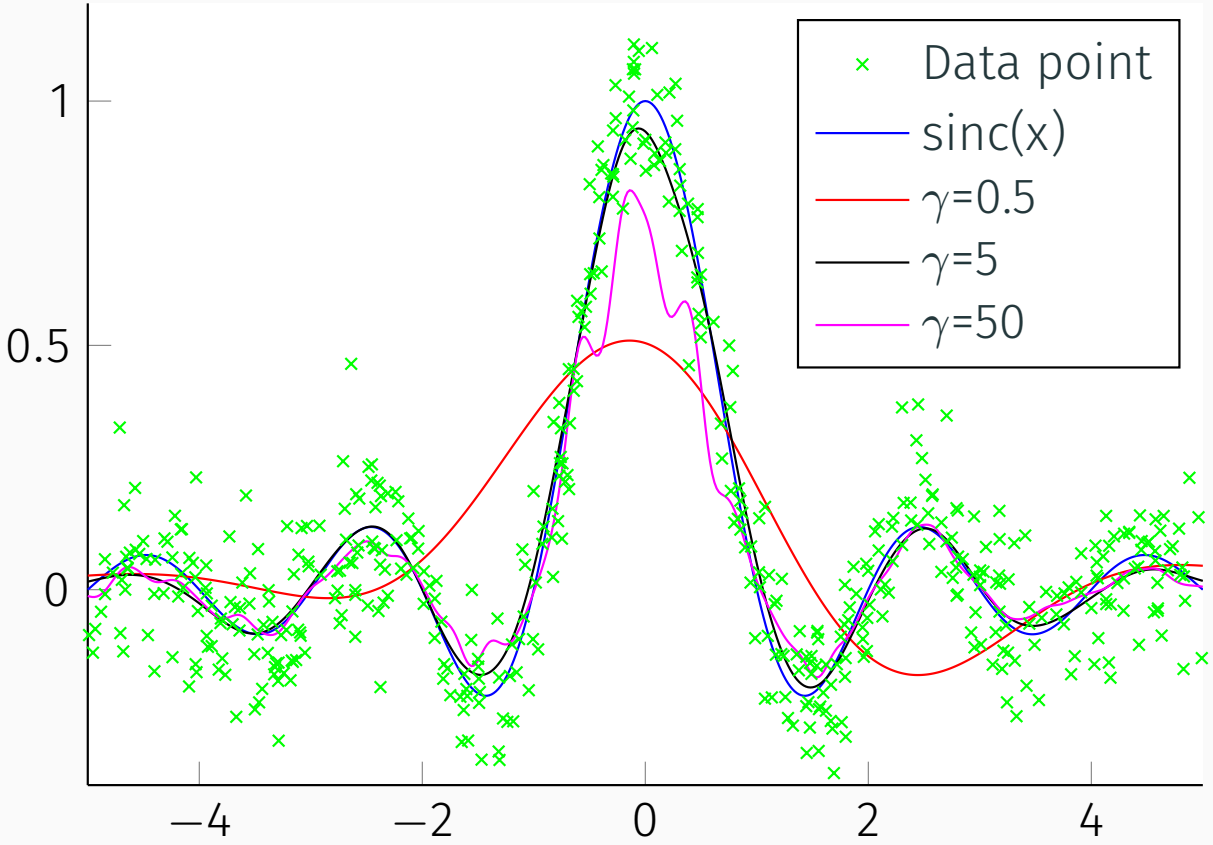
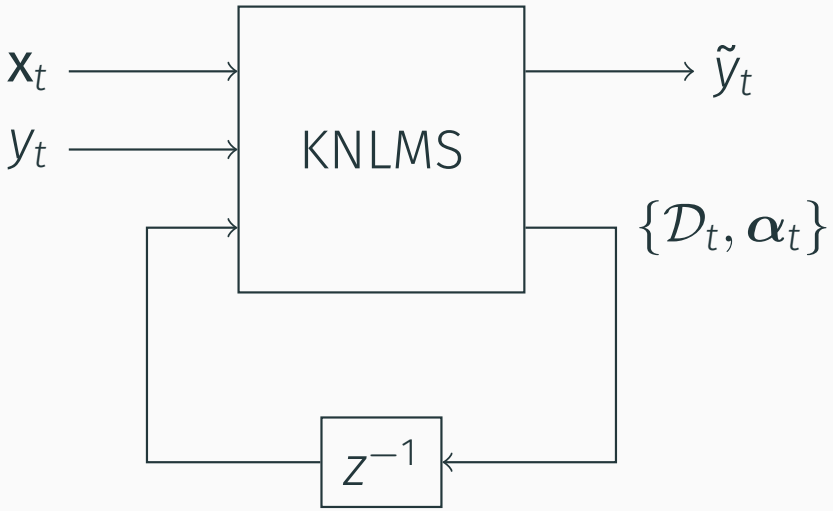


Figure 1: The accuracy of KNLMS while varying a single parameter.

Background: Kernel Methods

KNLMS model:

- A dictionary, $\mathcal{D}$.
- A vector of weights, $\boldsymbol{\alpha}$.
- A kernel function, $\kappa(\mathbf{x}_i, \mathbf{x}_j)$.



Prediction calculation (given $\mathbf{x}_t$):

$$\tilde{y}_t = \sum_{i=0}^{N} \alpha_i \kappa(\mathbf{x}_t, \tilde{\mathbf{x}}_i) = \mathbf{k}^T \boldsymbol{\alpha} \quad .$$

The expressions are recursive - this creates a dependency problem!

Avoid the dependency problem by reordering the loop.

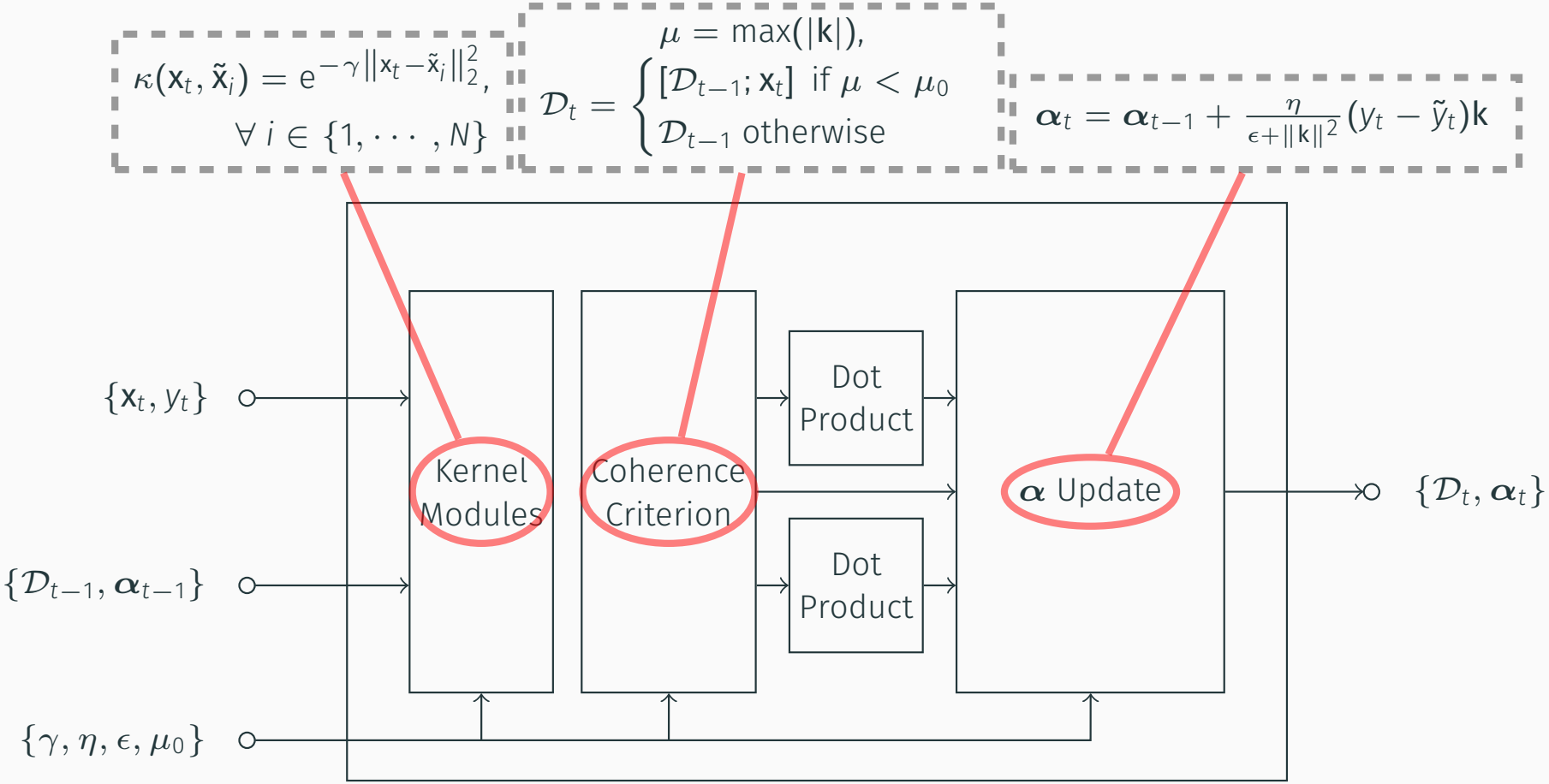
```
for (parameters) {
  for (examples) {
    learn_model();
  }
}
```

→

→

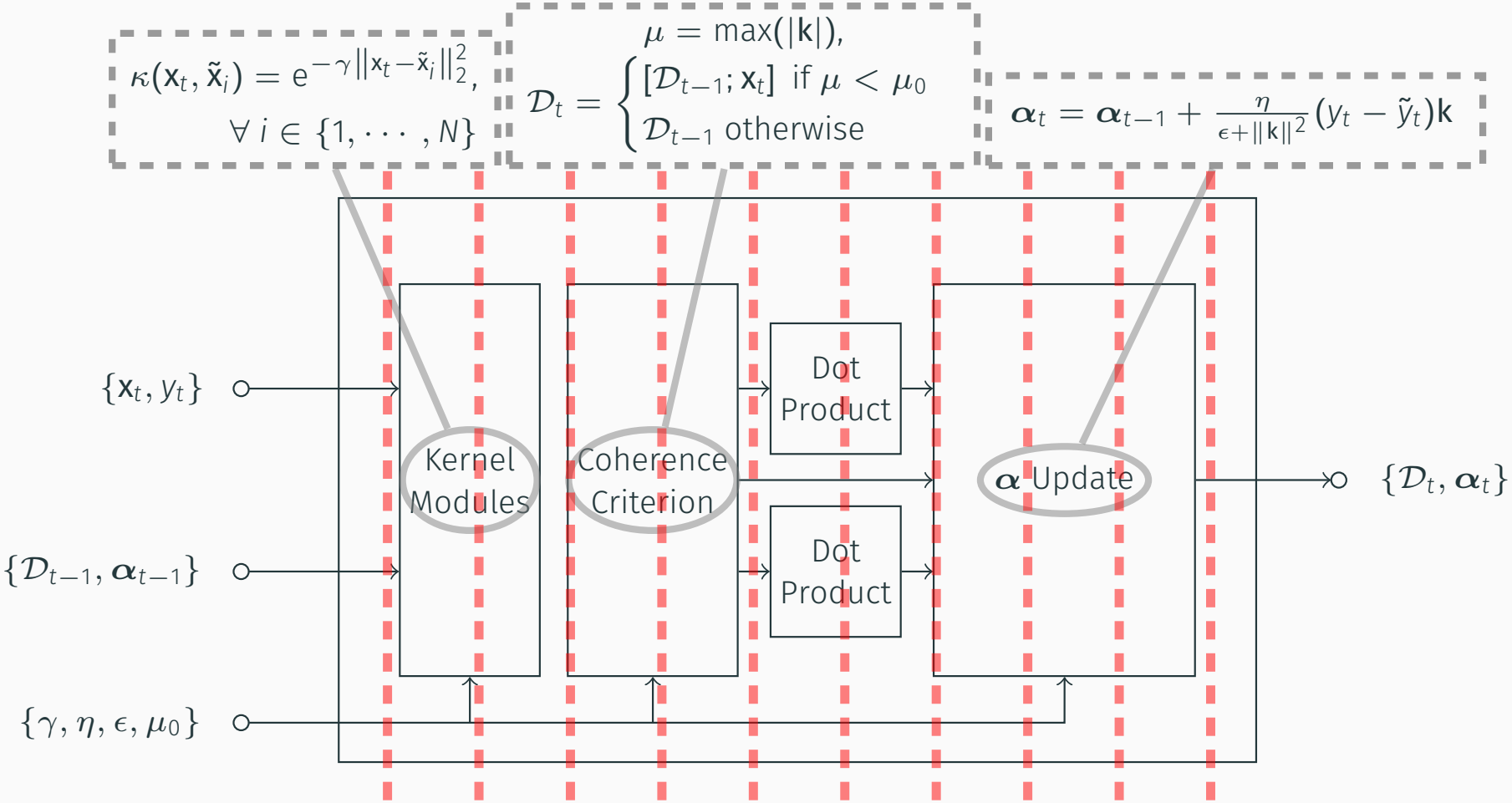
```
for (examples) {
  for (parameters) {
    learn_model();
  }
}
```

Architecture: The KNLMS Core



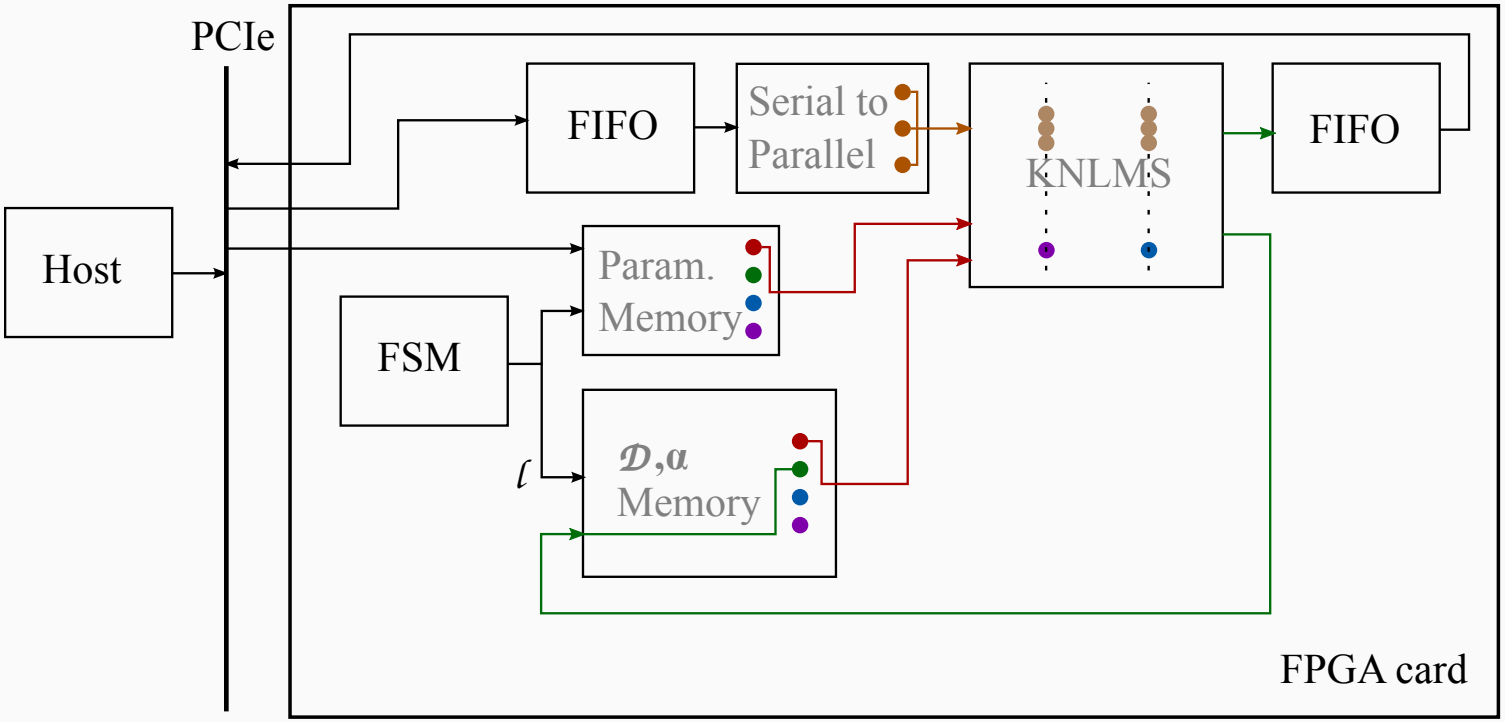
The core calculates a non-recursive portion of KNLMS.

Architecture: Pipelining the Core



$\sim$210 pipeline stages were needed to achieve $\sim$300MHz.

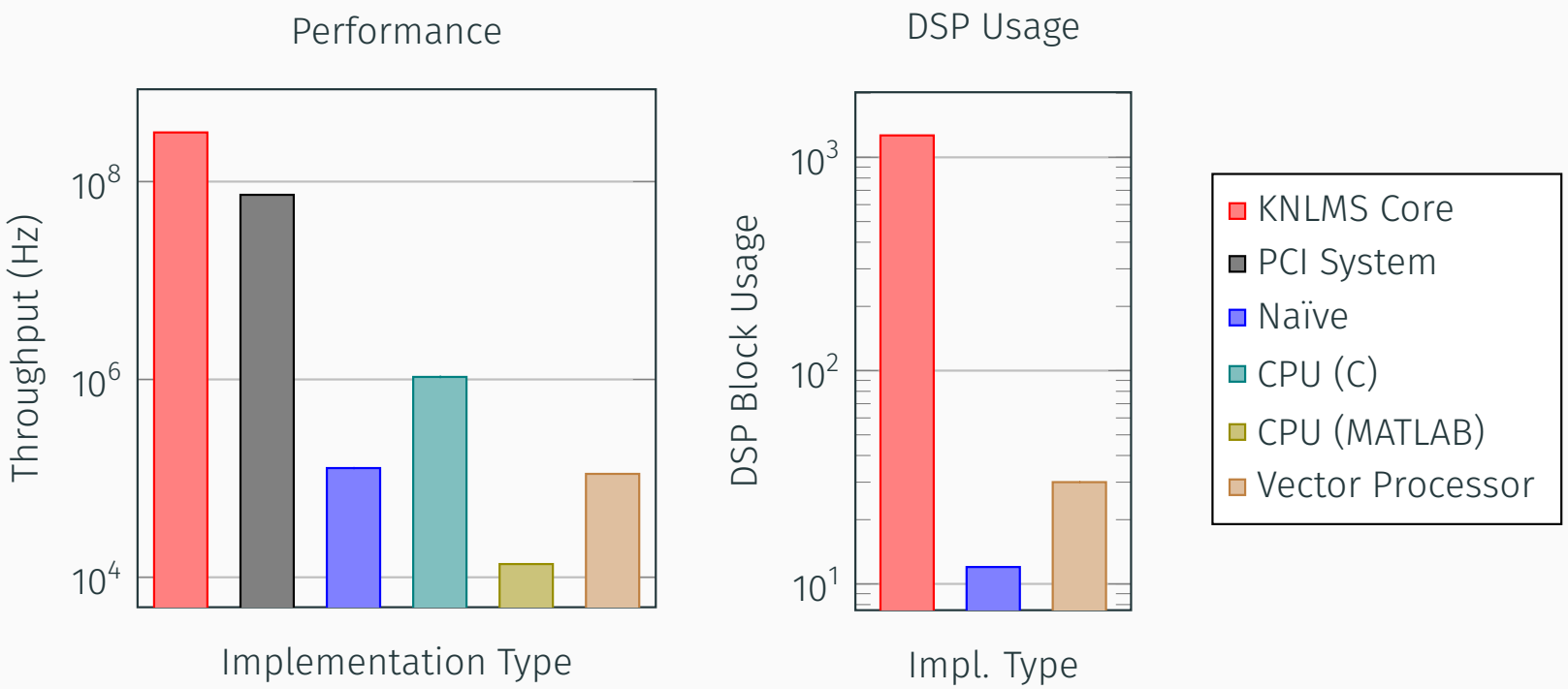
Architecture: PCI-based System



On each clock cycle, a different set of parameters is passed in.

Results: Performance

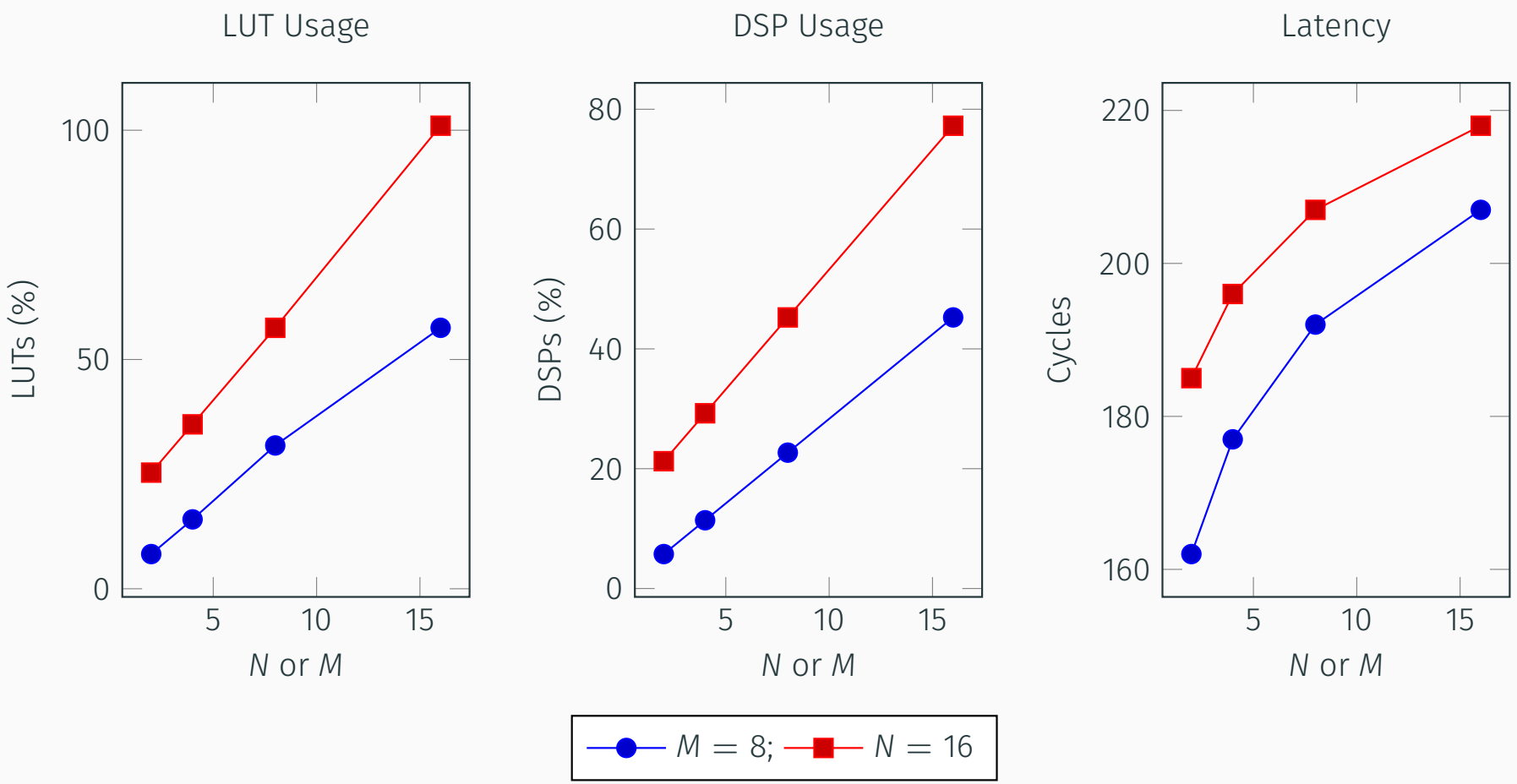
The core achieves speedups of 300×/2,800× over a CPU (C) and a vector processor respectively.



The core is capable of learning ∼210 models at ∼160GFLOPS.

Results: Scalability

- VC707 (Virtex 7): N=16, M=8. (N = dictionary entries, M = feature length)
- Virtex Ultrascale+: N=64, M=8 (estimated).



Conclusion: Summary

- Demonstration of a fully pipelined machine learning core:
  - The core achieves $\sim$300$\times$/$\sim$2,800$\times$ speedups over a CPU and a previous design.
  - A 210 stage pipeline core achieves 160GFLOPS.
  - PCI system achieves $\sim$70$\times$/$\sim$660$\times$ speedups over a CPU and a previous design.
- Floating point and fused arithmetic investigated - details in the paper!
- We hope this design enables embedded and real-time applications for machine learning.

Conclusion: Future Work

Future work includes:

- Further investigation of precision tradeoffs.
- Reducing the design latency.
- Implementing other machine learning algorithms.
- Using other platforms, such as GPUs and heterogeneous architectures.

Thank you. Any questions?

References I

Y. Pang, S. Wang, Y. Peng, N. J. Fraser, and P. H. Leong. A low latency kernel recursive least squares processor using FPGA technology. In *FPT*, pages 144–151, 2013.

C. Richard, J. C. M. Bermudez, and P. Honeine. Online prediction of time series data with kernels. *Signal Processing, IEEE Transactions on*, 57(3):1058–1067, 2009.

S. Van Vaerenbergh. Kernel methods toolbox KAFBOX: a Matlab benchmarking toolbox for kernel adaptive filtering. Grupo de Tratamiento Avanzado de Señal, Departamento de Ingeniería de Comunicaciones, Universidad de Cantabria, Spain, 2012.

Appendices

System Performance

Table 1: Comparison of online kernel method implementations. Note that $N = 16$ and $M = 8$.

Implementation	Algorithm	DSPs	Freq (MHz)	Time (ns)	Slowdown rel. to System
Naïve	KNLMS	12	96.7	7,829	576
Float	KNLMS	1267	314	3.18	0.23
System	KNLMS	691	250	13.6	1
CPU (C)	KNLMS	-	3,600	940	69
CPU (KAFBOX)	KNLMS	-	3,600	73,655	1703
Pang et al. [2013]	SW-KRLS	30	237	9,000	661

Growth Pattern

Table 2: Area utilisation of different designs obtained from synthesis. Note the computational complexity of KNLMS is $\mathcal{O}(mn)$.

Type	M	N	LUTs	DSPs	L	F_{max}
Float	2	16	77K	595	185	385
	4	16	109K	819	196	385
	16	16	307K	2163	218	385
	8	2	23K	161	162	385
	8	4	46K	319	177	385
	8	8	95K	635	192	385
	8	16	173K	1267	207	385
Fused	2	16	102K	494	161	303
	4	16	119K	595	163	303
	16	16	440K	1171	175	303
	8	2	33K	101	131	303
	8	4	64K	199	143	303
	8	8	130K	395	155	303
	8	16	247K	787	167	303