Reconfigurable Computing

Case Study: Kernel Adaptive Filters

Philip Leong | Computer Engineering Laboratory
School of Electrical and Information Engineering,
The University of Sydney

THE UNIVERSITY OF SYDNEY

Overview

- Motivation
- Kernel methods
  - Vector processor
  - Pipelined
  - Braided
  - Distributed
- Conclusion

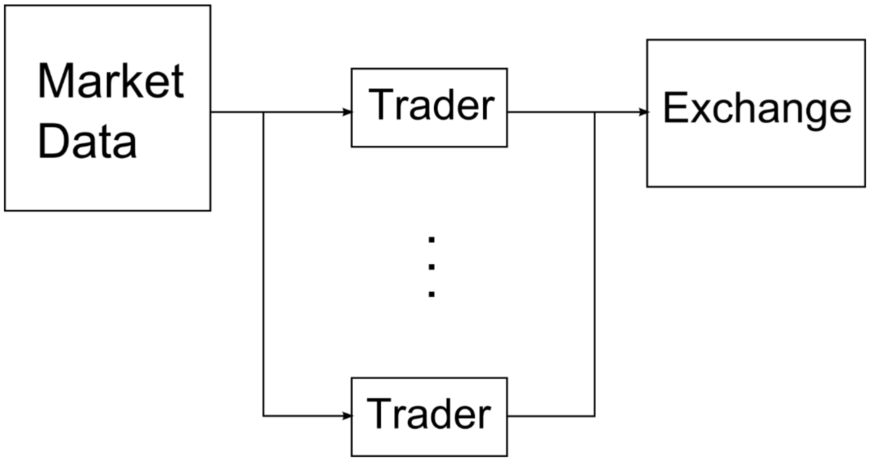
THE UNIVERSITY OF SYDNEY

Motivation (latency)

How to beat other people to the money (latency)

› Low latency trading looks to trade in transient situations where market equilibrium disturbed

- 1ms reduction in latency can translate to $100M per year



› Latency also important: prevent blackouts due to cascading faults, turn off machine before it damages itself, etc

Information Week: Wall Street's Quest To Process Data At The Speed Of Light

Motivation (latency)

Exablaze Low-Latency Products

ExaLINK Fusion 48 SFP+ port layer 2 switch for replicating data typical 5 ns fanout, 95 ns aggregation, 110 ns layer 2 switch

Xilinx Ultrascale KU115 FPGA, QDR SRAM, ARM processor

ExaNIC X10 typical raw frame latency 60 bytes 780 ns

What we can't do: ML with this type of latency

Source: exablaze.com

THE UNIVERSITY OF SYDNEY

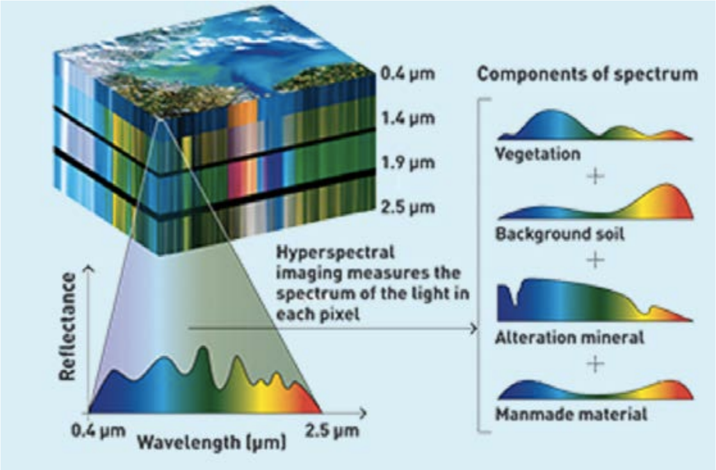
Motivation (throughput)

› Ability to acquire data improving (networks, storage, ADCs, sensors, computers)

- e.g. hyperspectral satellite images, Big Data e.g. SIRCA has 3PB of historical trade data

› Significant improvements in ML algorithms

- Deep learning (model high-level abstractions in data) for leading image and voice recognition problems; support vector machines to avoid overfitting



What we can't do: learning with this data rate

EPIC Technology Needed

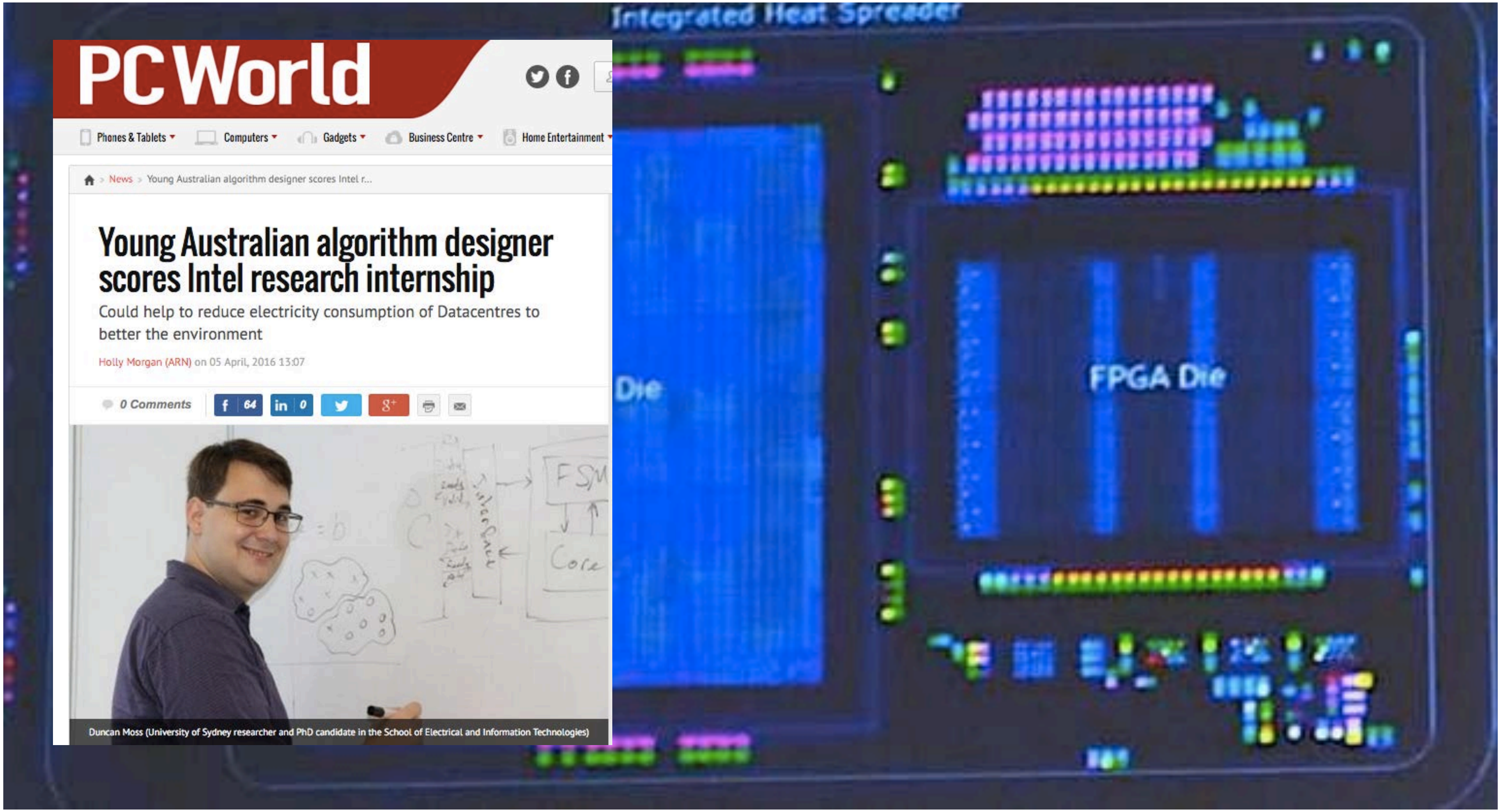
› To provide ML algorithms with higher throughput and lower latency we need

- Low **E**nergy – so power doesn’t become a constraint, operate off batteries (satellite and mobile)
- **P**arallelism – so we can reduce latency and increase throughput
- **I**nterface – so we don’t need to go off-chip which reduces speed and increases energy
- **C**ustomisable – so we can tailor entire design to get best efficiency

› **Using FPGAs, develop improved algorithms and system implementations for ML**

Why is FPGA research timely?

(Intel paid $16.7B for Altera; Intel presentation at OCP Summit 14/3/2016)



Overview

› Motivation

› **Kernel methods**

  - Vector processor
  - Pipelined
  - Braided
  - Distributed

› Conclusion

THE UNIVERSITY OF SYDNEY

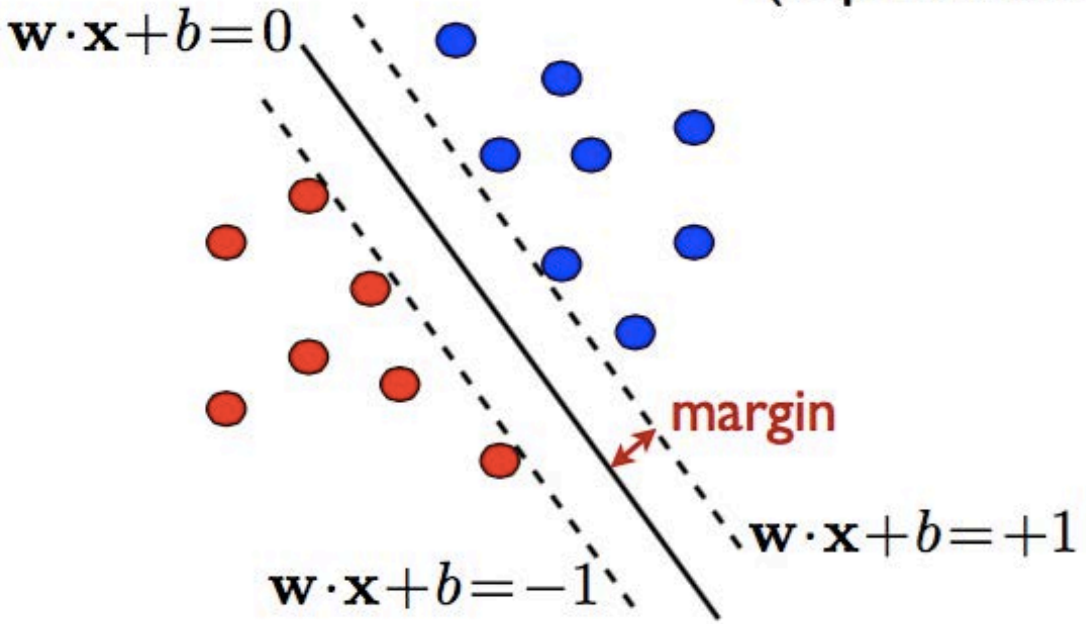
Linear Techniques

- Linear techniques extensively studied
  - Solution has form y = $\mathbf{w}^T\mathbf{x}$ + b
  - Use training data x to get maximum likelihood estimate of **w** or a posterior distribution of **w**
- Pros
  - Sound theoretical basis
  - Computationally efficient
- Cons
  - Linear!
- There is an equivalent dual representation

$$f(x) = \langle w, x \rangle + b = \sum \alpha_i y_i \langle x_i, x \rangle + b$$

e.g. Max Margin Hyperplane

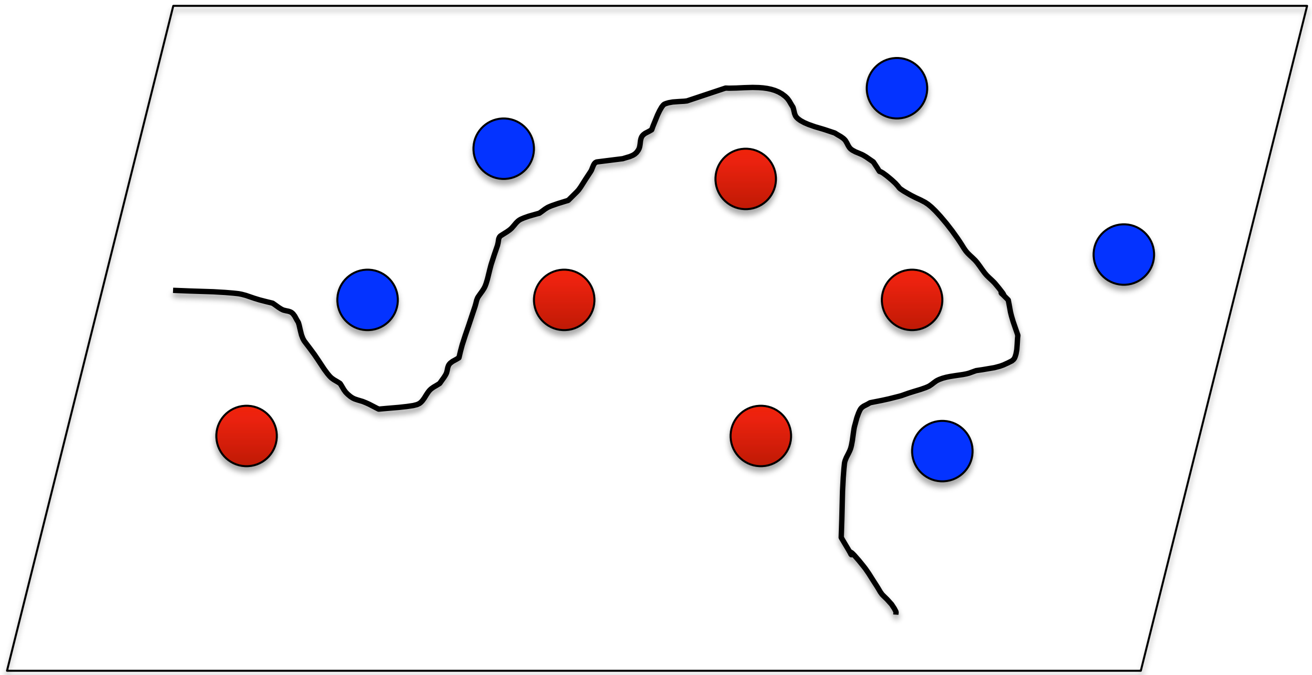
(Vapnik and Chervonenkis, 1964)



$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2$$

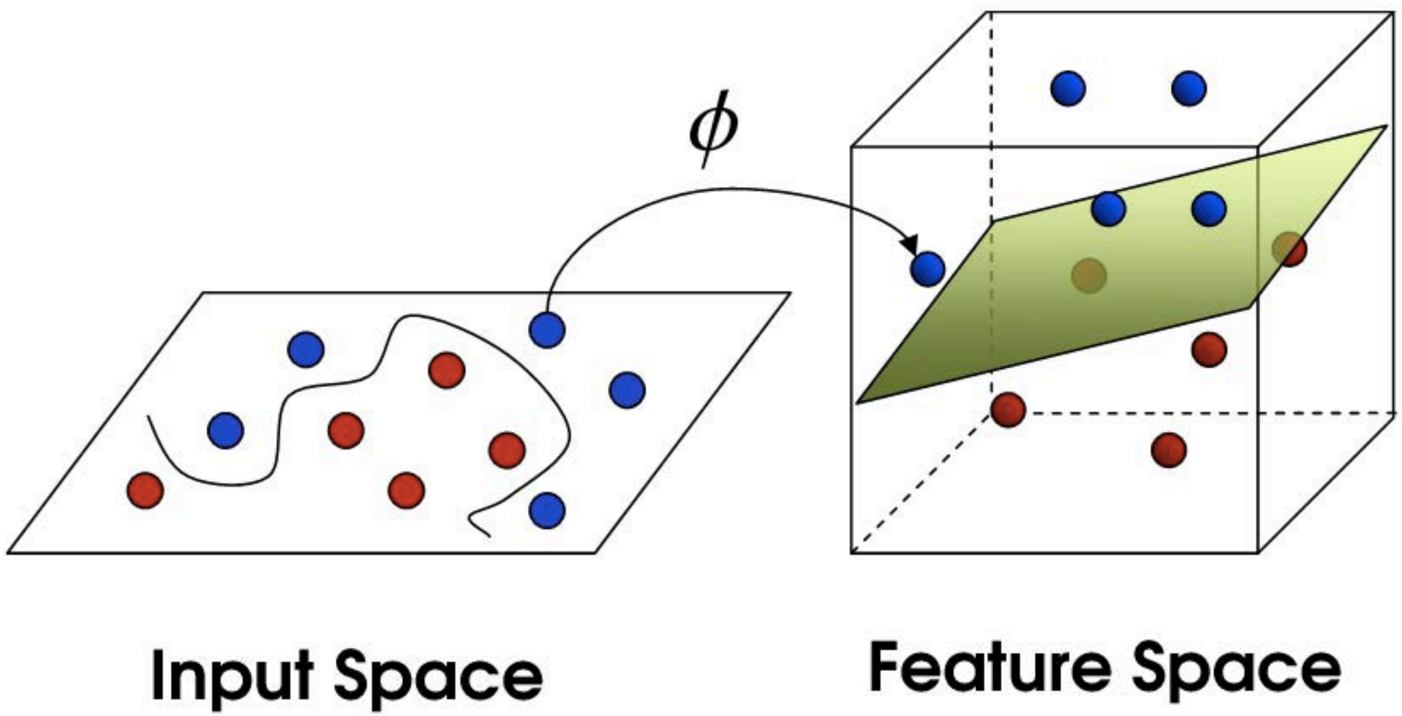
$$\text{subject to} \quad y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \geq 1, i \in [1, m].$$

What do we do if given this problem?



› Map the problem to a **feature space**

Mapping to a Feature Space



- Choose high dimensional feature space (so easily separable)
- **BUT computing Φ is expensive!**

Kernel Trick

› Kernel is a similarity function

- defined by an implicit mapping ϕ, (original space to feature space)

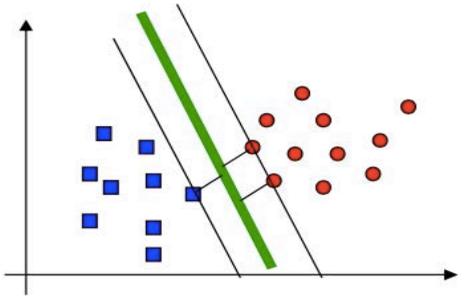
$$\kappa(x, x') = \phi(x)^T \phi(x') = \langle \phi(x), \phi(x') \rangle$$

- e.g. Linear kernel $\kappa(x,x') = \langle x, x' \rangle$
- e.g. Polynomial kernel $\kappa(x,x') = (1 + \langle x, x' \rangle)^d$ for d=2: $\phi(x) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)$
- e.g. Gaussian kernel (universal approximator) $k(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\,\sigma^2}\right)$
  - $\Phi(x)$ infinite in dimension!

› **Modify linear ML techniques to kernel ones by replacing dot products with the kernel function (kernel trick)**

- e.g. linear discriminant analysis, logistic regression, perceptron, SOM, K-means, PCA, ICA, LMS, RLS, …
- **While we only describe prediction here, also applied to training equations**

Support Vector Machine



- **The decision boundary:**

$$\hat{\mathbf{w}}^T\mathbf{x} + w_0 = \sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i{}^T \mathbf{x}) + \text{b}$$

- **Classification decision:**

$$\hat{y} = \text{sign}\left[\sum_{i \in SV} \hat{\alpha}_i y_i (\mathbf{x}_i{}^T \mathbf{x}) + \text{b}\right]$$

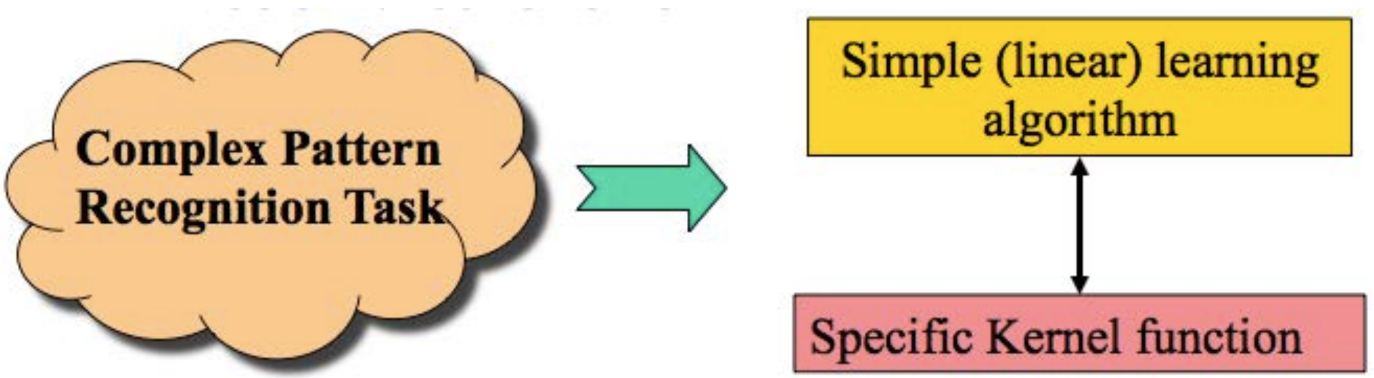
$$\mathbf{x} \rightarrow \varphi(\mathbf{x})$$

$$K(\mathbf{x}, \mathbf{x}') = \varphi(\mathbf{x})^T \varphi(\mathbf{x}')$$

Never explicitly compute Φ(x), computing K(x,x') is O(m)
e.g. poly kernel Φ(x), dimension (d+m-1)!/d!(m-1)!
For d=6, m=100 this is a vector of length 1.6e9

Approach

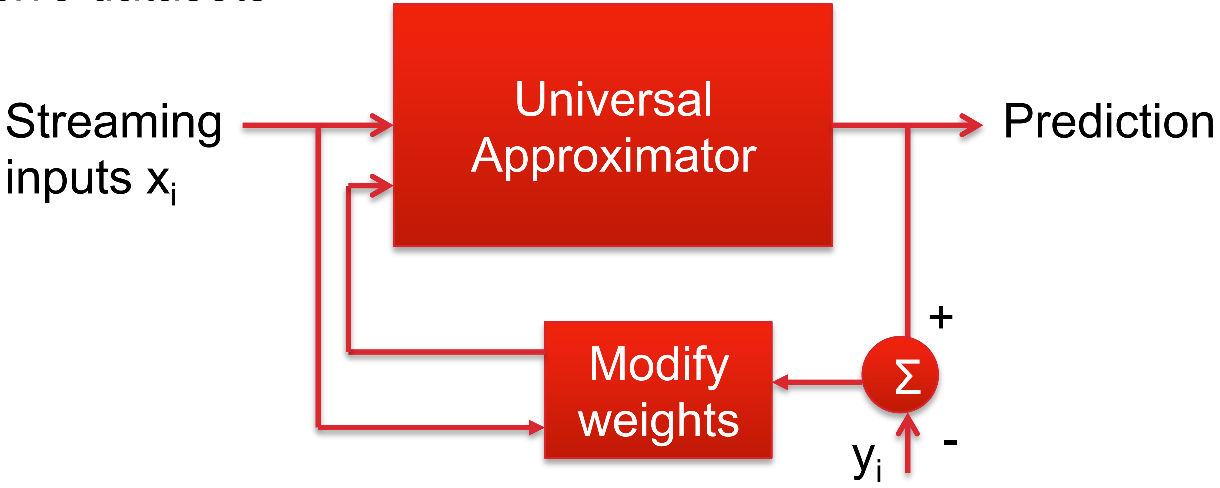
- In Kernel-based learning algorithms, problem solving is now decoupled into:
  - A general purpose learning algorithm often linear (well-founded, robustness, …)
  - A problem specific kernel (we focus on time series but kernels exist for text, DNA sequences, NLP)



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



Kernel Online Algorithms

Two extensively studied types of online kernel methods:

› Kernel Least Mean Squares (KLMS)
  - $O(N)$
  - Converges slowly (steepest descent)
  - Takes a 'step' towards minimising the instantaneous error
  - e.g. KNLMS, NORMA

› Kernel recursive least squares (KRLS)
  - $O(N^2)$
  - Converges quickly (Newton Raphson)
  - Directly calculates least squares solution based on previous training examples using Matrix Inversion Lemma (matrix-vector multiplication)
  - e.g. SW-KRLS

Convergence

RICHARD *et al.*: ONLINE PREDICTION OF TIME SERIES DATA WITH KERNELS

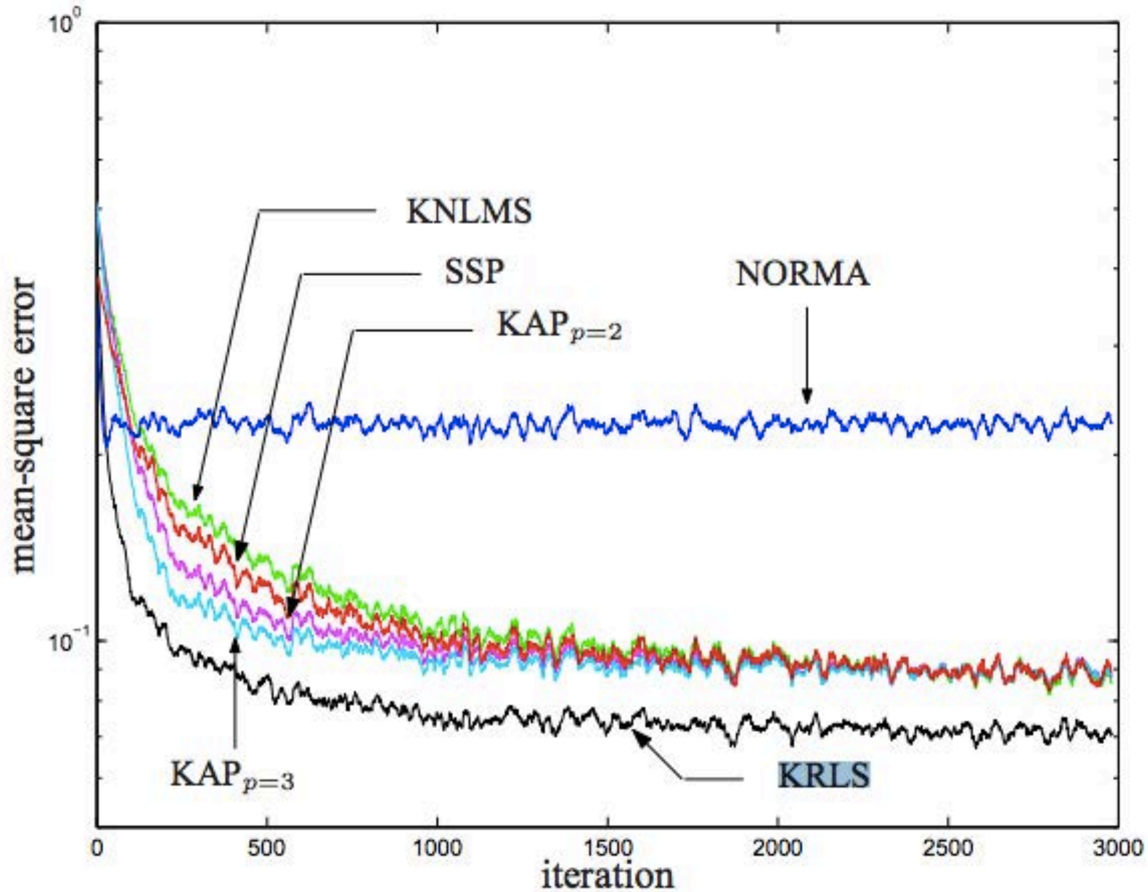


Fig. 2. Learning curves for KAP, KNLMS, SSP, NORMA and KRLS obtained by averaging over 200 experiments.

Overview

- Motivation
- Kernel methods
  - Vector processor
  - Pipelined
  - Braided
  - Distributed
- Conclusion

SW-KRLS Algorithm

The pseudo code of the SW-KRLS algorithm

Initialize $K_0 = (1+c)I$ and $K_0^{-1} = I/(1+c)$.

for n=1,2.. **do**

  Get $\tilde{K}_n$ from K_{n-1} with Eq.(1)

  Calculate $\tilde{K}_{n-1}^{-1}$ with Eq.(2)

  Get K_n with Eq.(3)

  Calculate K_n^{-1} with Eq.(4)

  Get the updated solution $\alpha_n = K_n^{-1} Y_n$

end for

Computation complexity: $O(N^2)$.

$$\tilde{K}_n = \begin{bmatrix} K_{n-1} & k_n(x_n) \\ k_n(x_n)^T & k_{nn} + c \end{bmatrix} \quad (1)$$

$$\hat{\mathbf{K}}_n^{-1} = \begin{bmatrix} \mathbf{K}_{n-1}^{-1}(\mathbf{I} + \mathbf{b}\mathbf{b}^T \mathbf{K}_{n-1}^{-1T} g) & -\mathbf{K}_{n-1}^{-1}\mathbf{b}g \\ -(\mathbf{K}_{n-1}^{-1}\mathbf{b})^T g & g \end{bmatrix} \quad (2)$$

where $b = k_{n-1}(x_n)$ $d = k_{nn} + c$ $g = (d - b^T K_{n-1}^{-1} b)^{-1}$

$$K_n = \begin{bmatrix} k_{n-N,n-N} + c & p^T \\ p & \tilde{K}_{n-1} \end{bmatrix} \quad (3)$$

where $p = [k(x_{n-N}, x_{n-N+1}), \ldots, k(x_{n-N}, x_{n-1})]^T$

$$K_n^{-1} = G - \frac{ff^T}{e} \quad (4)$$

where $\tilde{K}_n^{-1} = \begin{bmatrix} e & f^T \\ f & G \end{bmatrix}$

Key idea

Vector add C = A + B

- **Microprocessor** O(N) cycles

```
for (i = 0; i < N; i++)
    C[i] = A[i] + B[i];
```

- **Vector processor** O(1) cycle

```
VADD(C, A, B)
```

- Implemented as a custom KRLS vector processor using FPGA technology

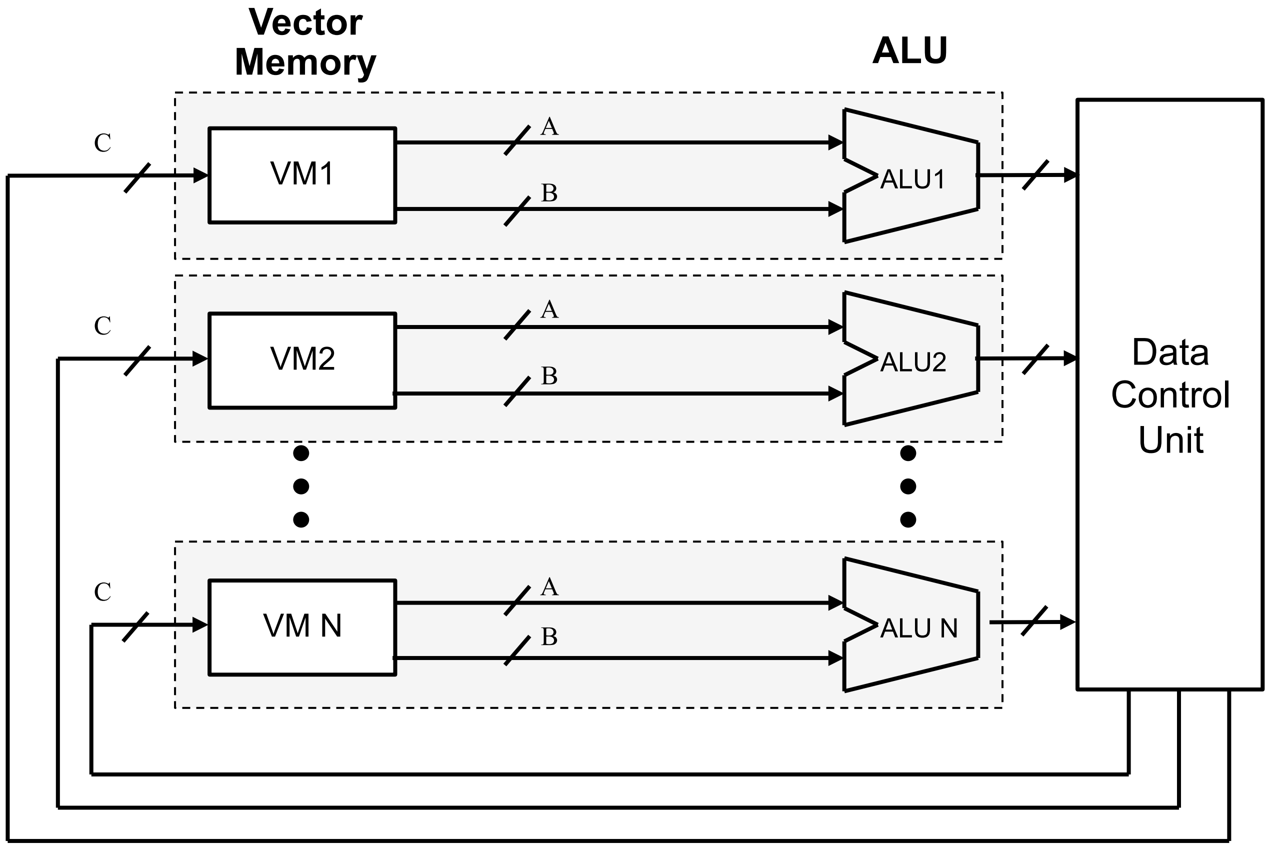
Instruction Set

ALL i , j AND L INDEXES RANGE FROM 1 TO N

Microcode (Opcode)	Function	Total Cycles
NOP(000)	No operation	1
BRANCH (0111)	BRANCH	4
VADD (0001)	**Vector add**	**14**
VSUB (0010)	**Vector subtract**	**14**
VMUL (0011)	**Array multiply**	**10**
VDIV (0100)	**Vector divide**	**N+28**
VEXP (0110)	**Vector exponentiation**	**N+21**
S2VE (1000)	**Clone a vector N times**	**N+4**
PVADD (1001)	***N x Vector add***	***N+13***
PVSUB (1010)	***N x Vector subtract***	***N+13***
PVMUL (1011)	***N x Vector multiply***	***N+9***
PVDOT (0101)	***N x Vector dot product***	***N+9+10***

SW-KRLS and other kernel methods implemented efficiently using this simple instruction set

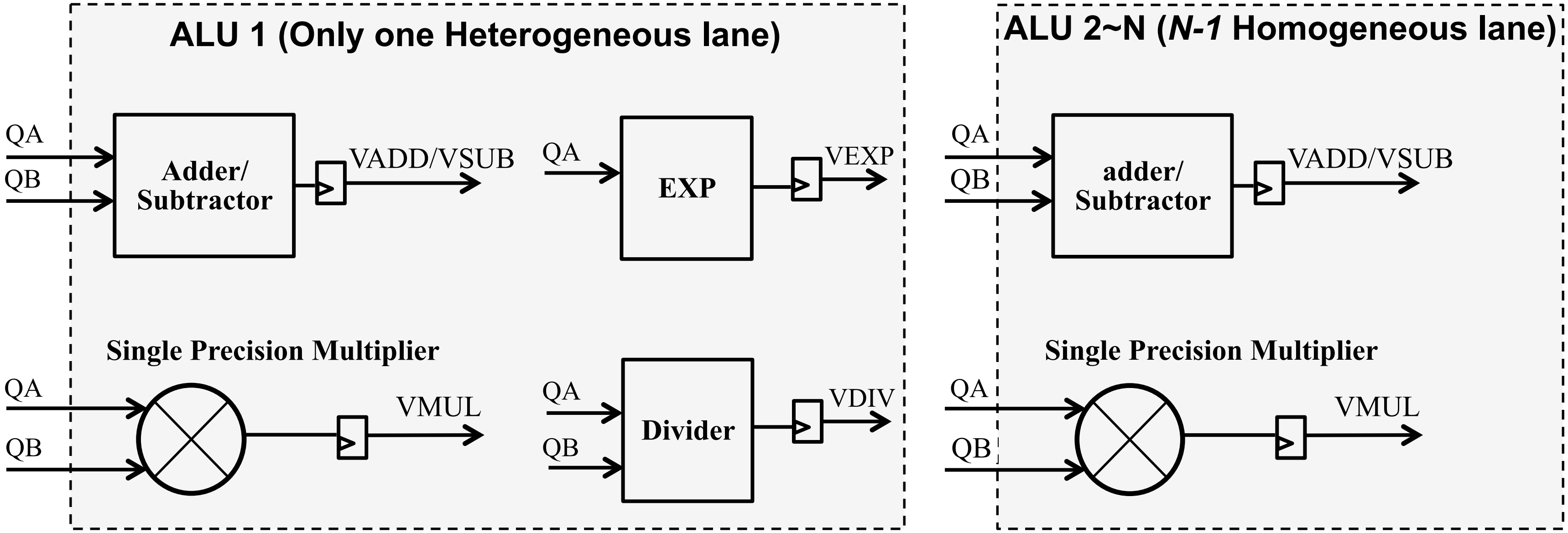
Datapath



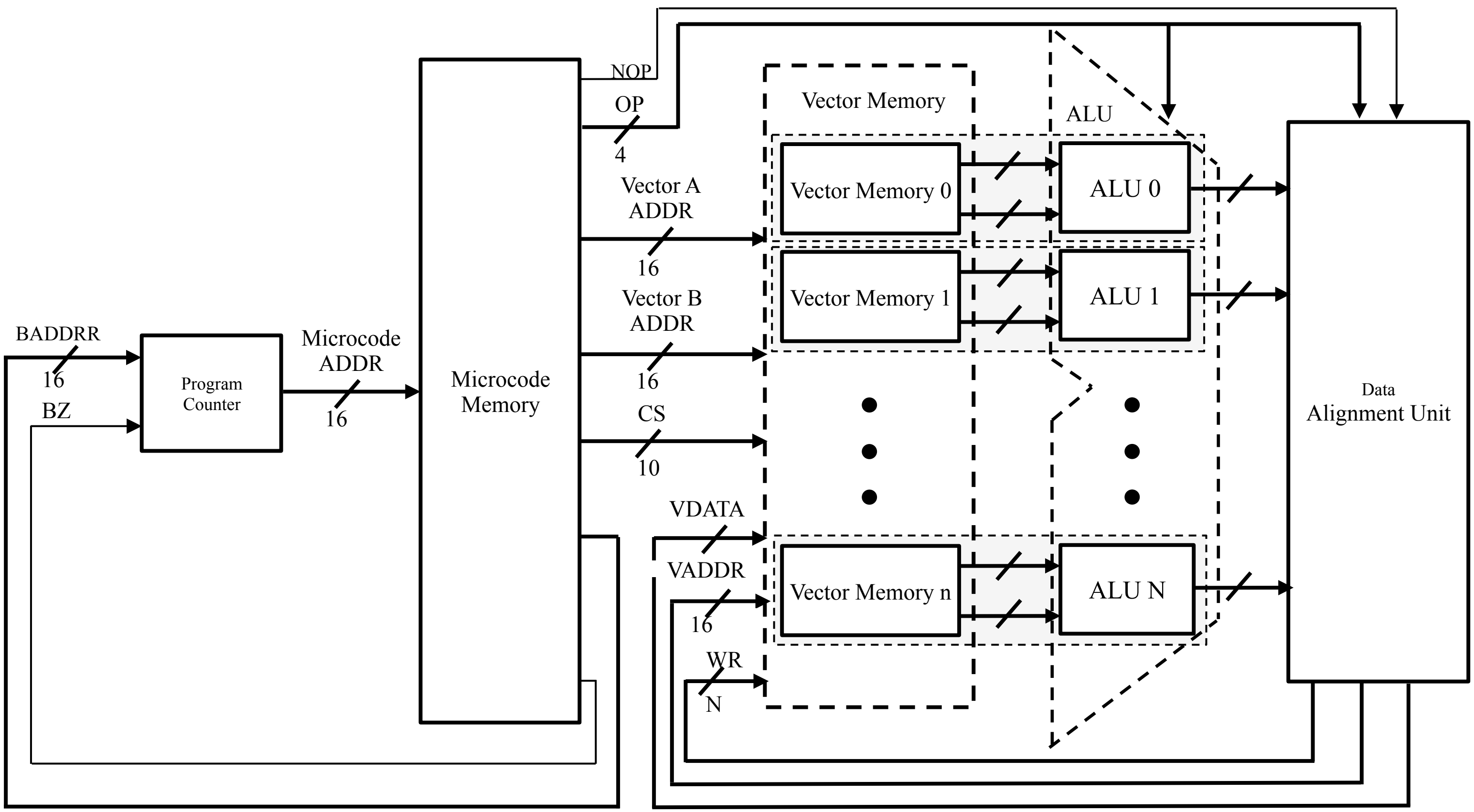
Vector Memory
ALU
C
VM1
A
B
ALU1
C
VM2
A
B
ALU2
C
VM N
A
B
ALU N
Data Control Unit

ALU architecture

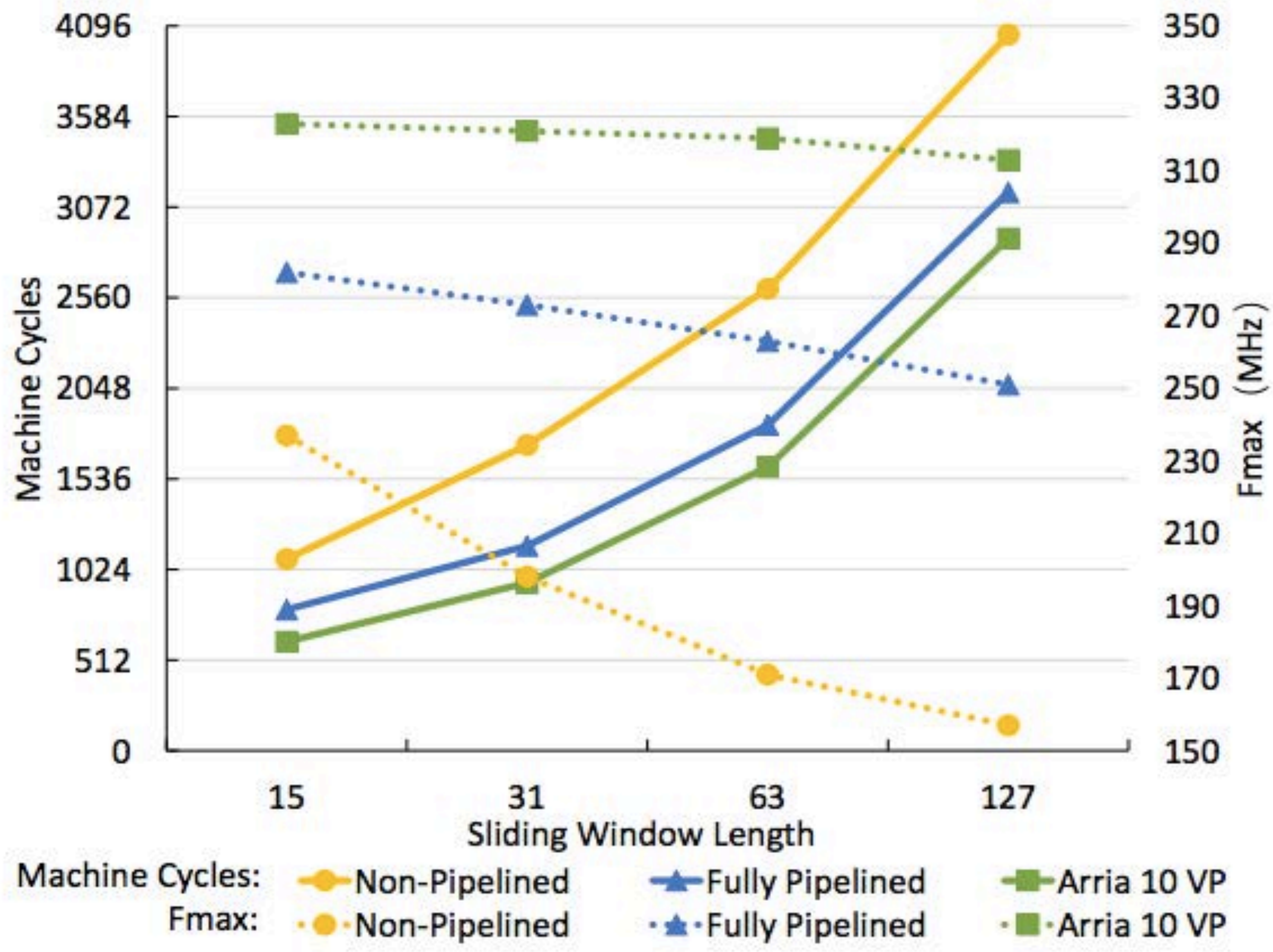
- ALU 1 - adder, multiplier, exp and divider
- ALU 2 .. ALU N - only adder and multiplier



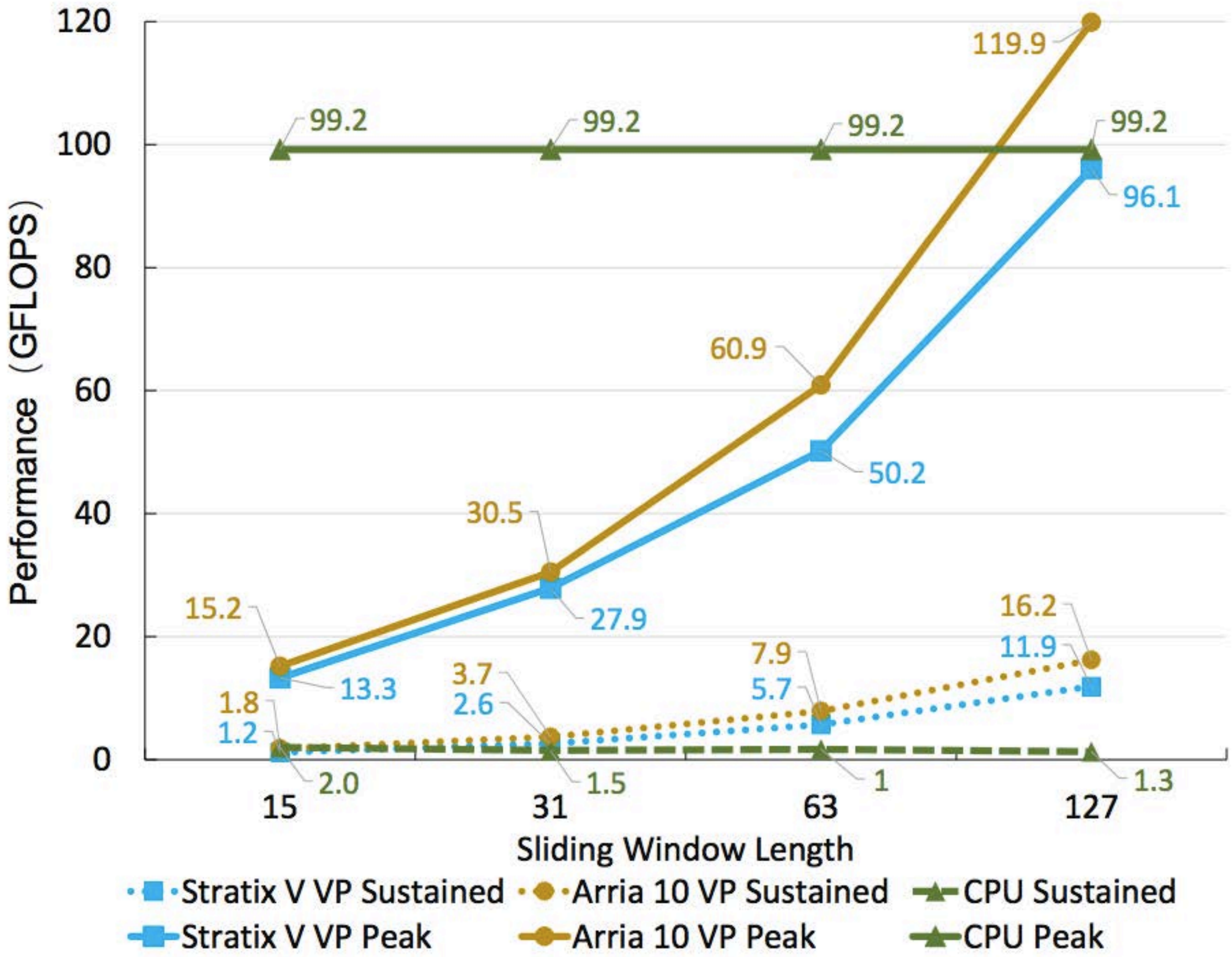
Detailed Datapath



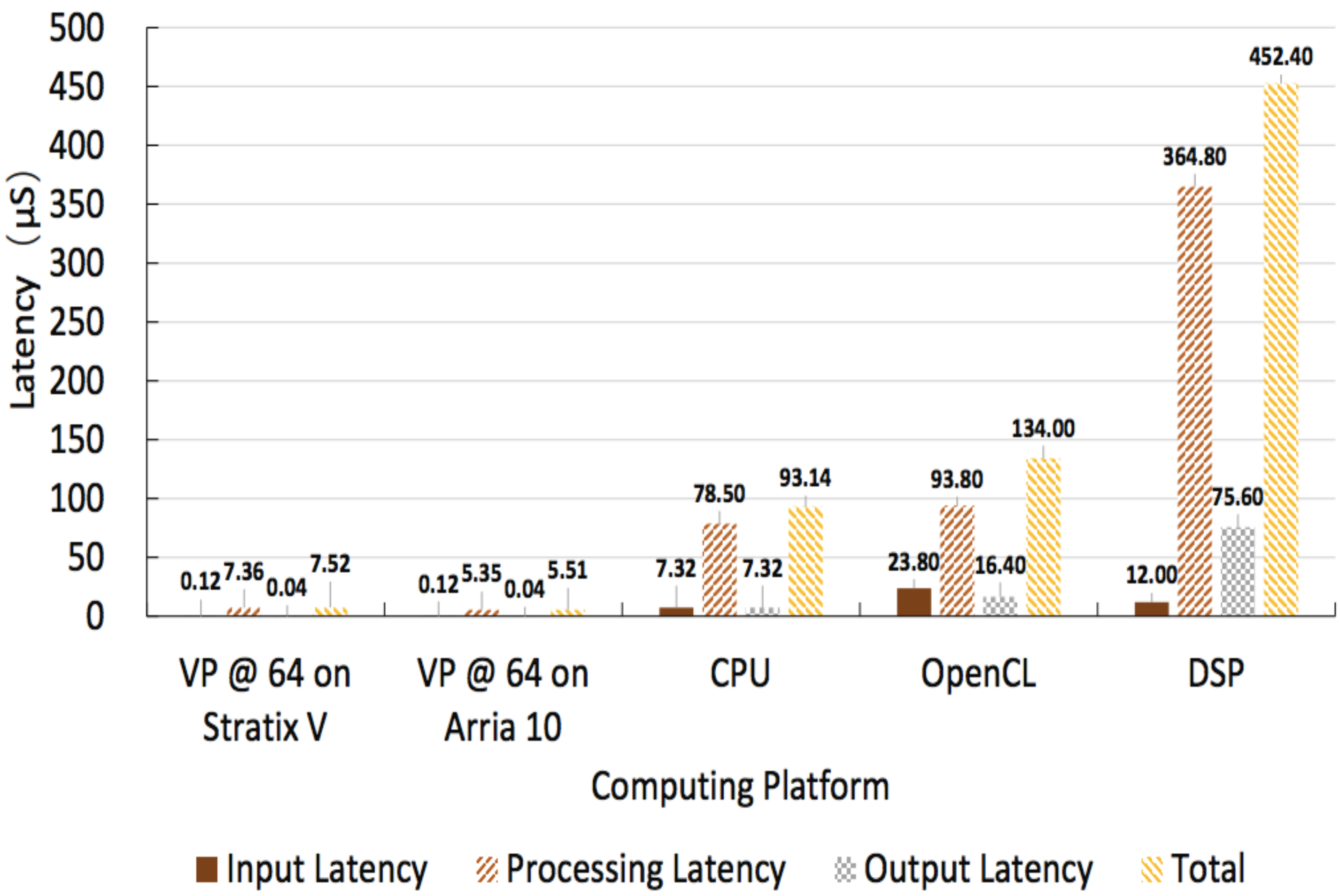
Arria 10 vs Stratix V vs CPU Single P&R



Arria 10 vs Stratix V vs CPU Single Performance



Arria 10 vs Stratix V vs CPU Single Latency



Performance Summary (Stratix V)

- SW-KRLS N=64

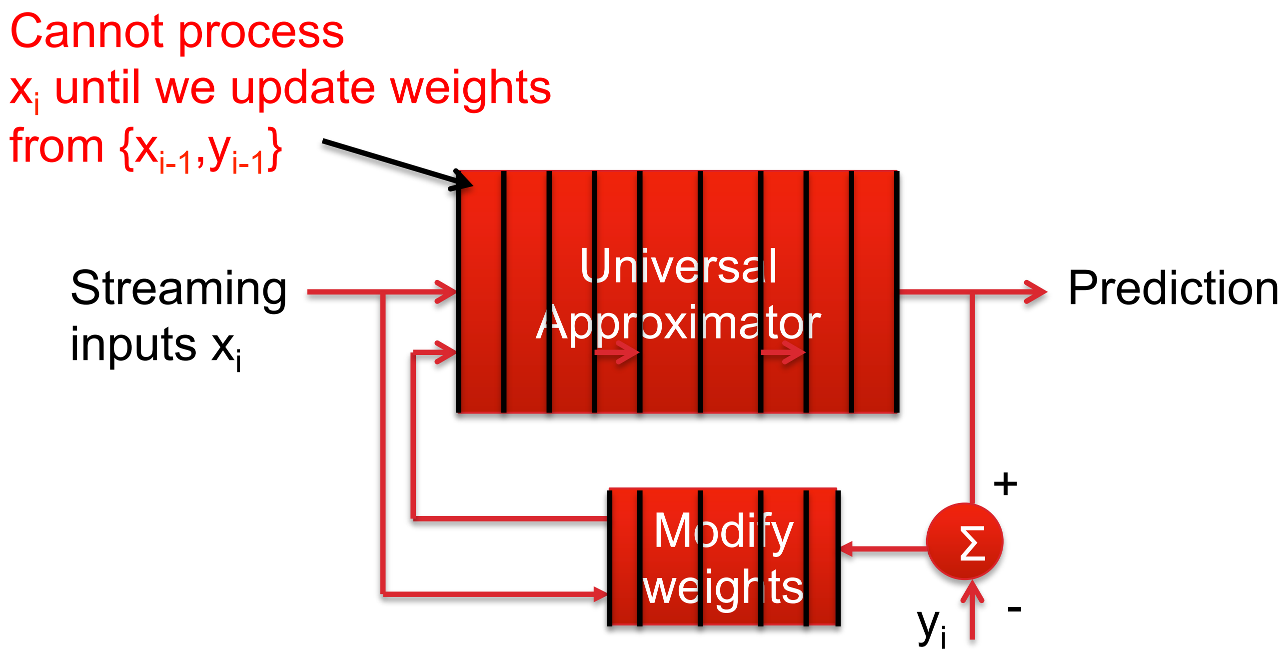
Platform	Power (W)	Latency (uS)	Energy (10^-5 J)
Our processor (DE5 5SGXEA7N)	2 (27)	**1 (12.6)**	**1 (34)**
DSP (TMS320C6678**)**	**1 (13)**	355 (4476)	181 (6167)
CPU (i5-2400@3.1GHz)	1 (13)	16 (201)	8 (269)

Overview

- Motivation
- Kernel methods
  - Vector processor
  - **Pipelined**
  - Braided
  - Distributed
- Conclusion

Obstacle to Pipelining

Dependency Problem



KNLMS Regression

- Finds D (dictionary which is subset of input vectors), and α (weights) for function

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

- Is a stochastic gradient descent style kernel regression algorithm. Given a new input/output pair, $\{x_n, y_n\}$, weight update is:
- 1. Evaluate κ between x_n and each entry of D_{n-1}, creating kernel vector, k.
- 2. If max(k) < μ_0, add x_n to the dictionary, producing D_n
- 3. Update the weights using:

$$\alpha_n = \alpha_{n-1} + \frac{\eta}{\varepsilon + k^T k}(y_n - k^T \alpha_{n-1})k$$

- How can we chose κ, μ_0, η and ϵ? We must do a parameter search.

Removing Dependencies

› Training is usually:

for (hyperparameters)

  for (inputs)

    learn_model()

› Alternative is to find L independent problems

- E.g. monitor L different things

› Our approach: run L independent problems (different parameters) in the pipeline

- Updates ready after L subproblems
- Less data transfer

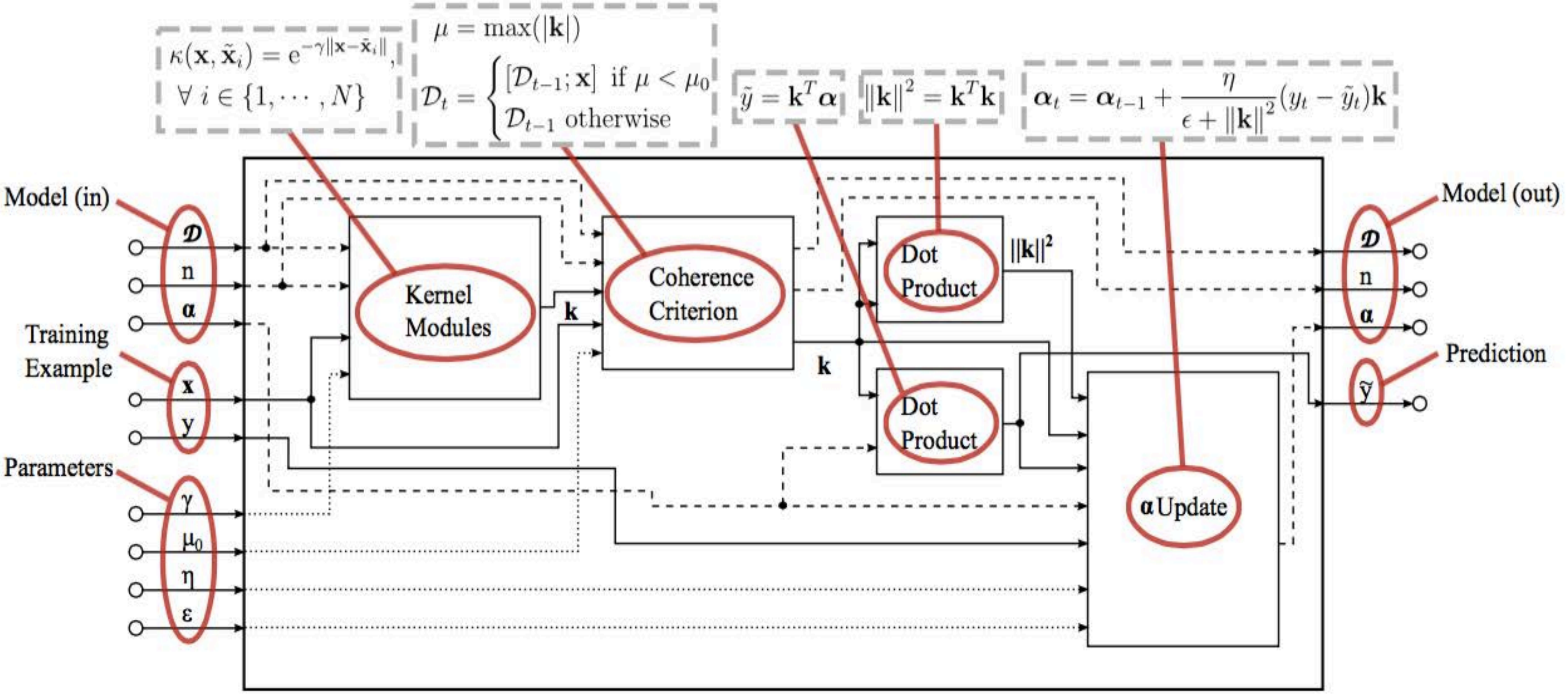
for (inputs)

  for (hyperparameters)

    learn_model()

- Similar approach for multiclass classification (train C(C-1)/2 binary classifiers)

High Throughput KNLMS



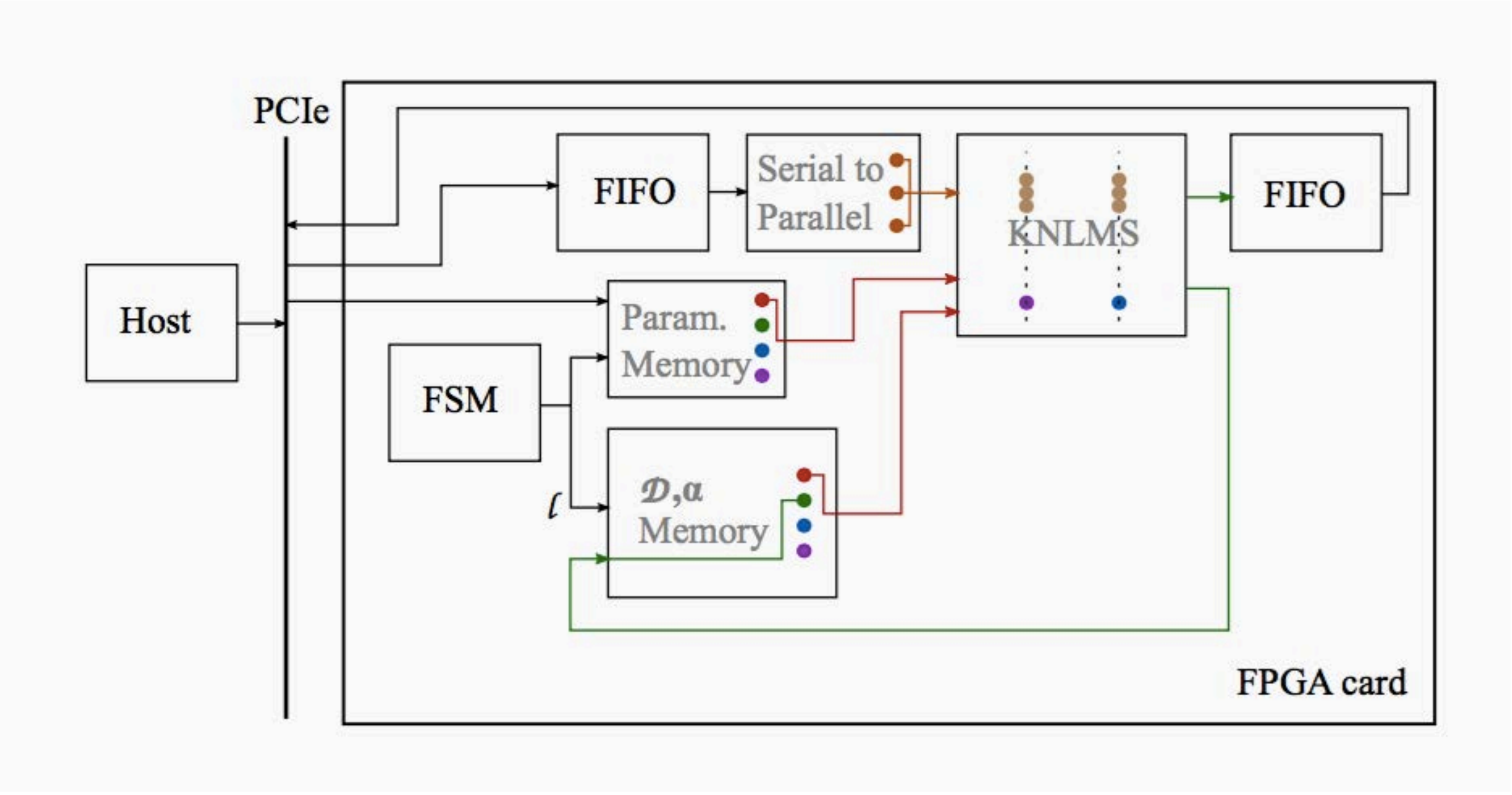
Scalability

- Area O(MN)
- Memory O(MN)
- Latency O($\log_2$N+$\log_2$M)

	+ (11)	× (7)	/ (30)	exp (20)	< (4)
Operation	$2MN + 2N$	$MN+$ $4N + 1$	1	N	$N - 1$
Latency	$\log_2 N+$ $\log_2 M + 3$	5	1	1	$\log_2 N$

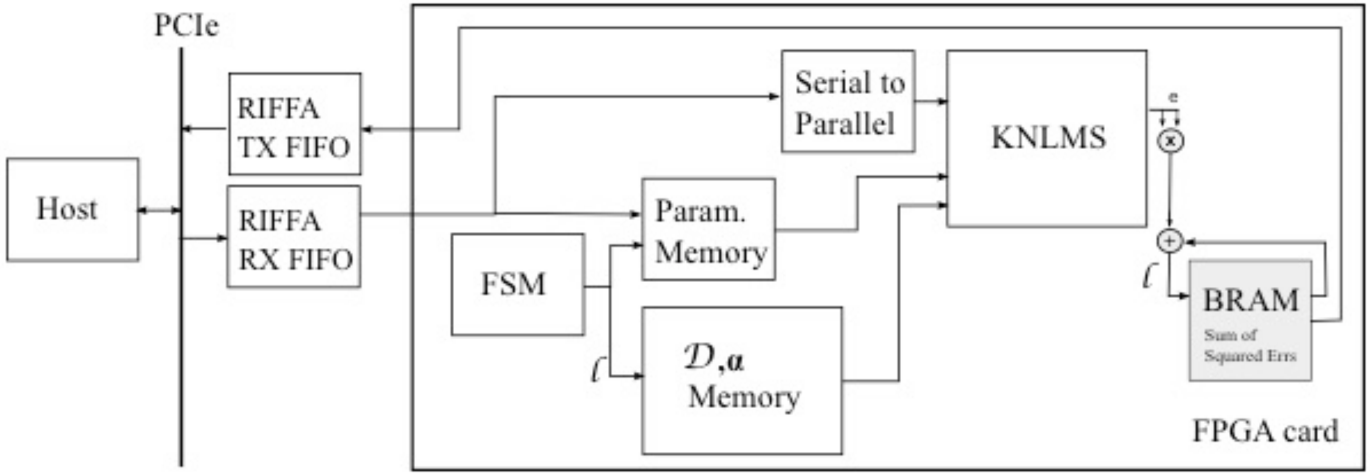
Original Implementation

- Break feedforward/feedback path and sythesised with Vivado HLS
- RIFFA 2.2.0 used for PCIe interface



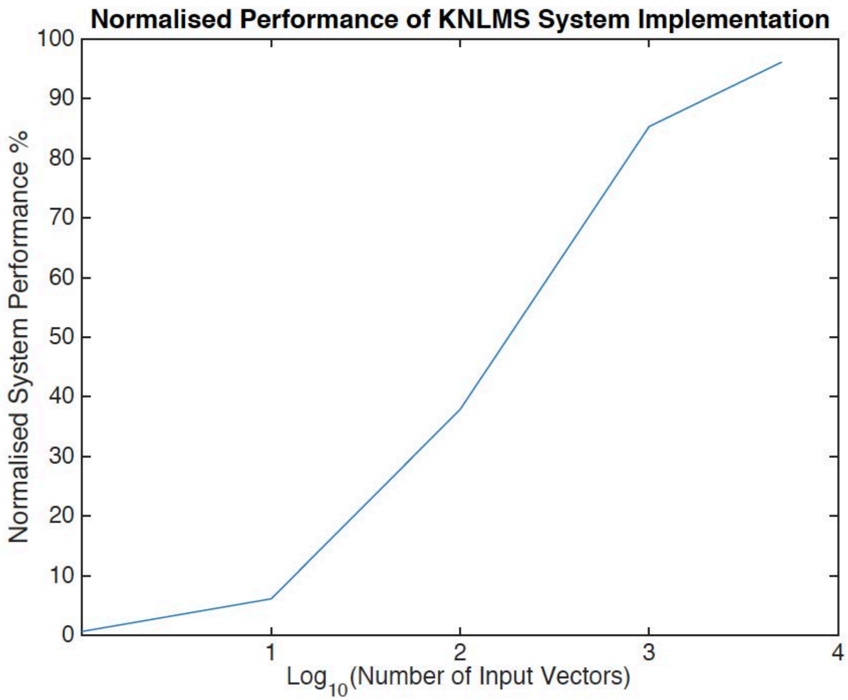
Optimised Search

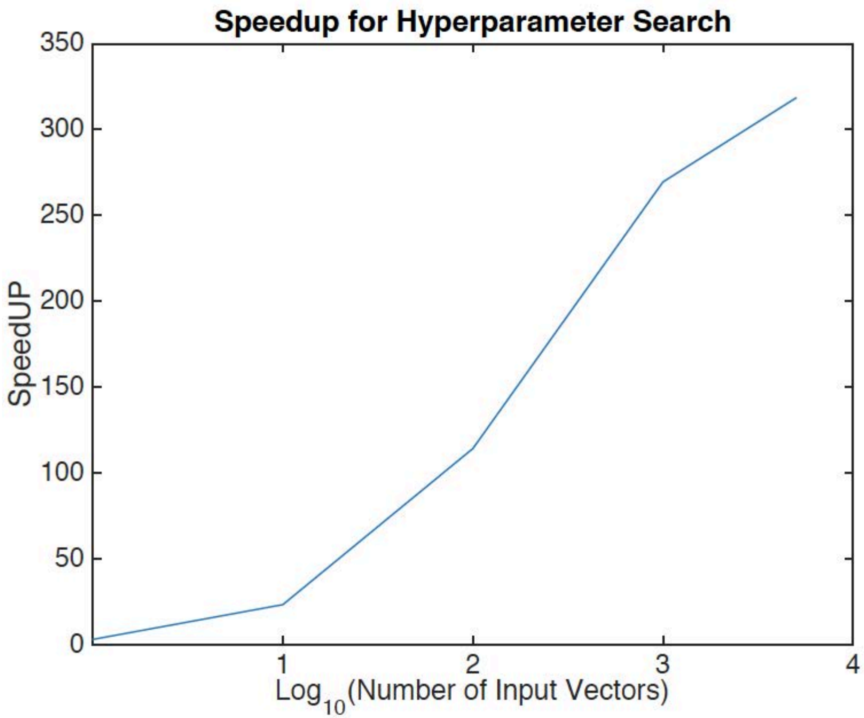
- Float KNLMS core integrated with RIFFA on Xilinx VC707 board (# ParamSets = 256)



- Original version read back all the predicted values
- Optimised by send all data and then read back accumulated square error
- 3x faster than original interface

Performance of KNLMS Core vs Number of Train Samples





- 96% of the peak core performance has been achieved with RIFFA integration when # of train samples reaches 4999

Performance

Core with input vector M=8 and dictionary size N=16 (KNLMS)

Implementation	Freq (MHz)	Time (ns)	Slowdown
Float	314	3	1
System	250	4	1.3
Naive	97	7,829	2,462
CPU (C)	3,600	940	296
Pang et al (2013)	282	1,699	566

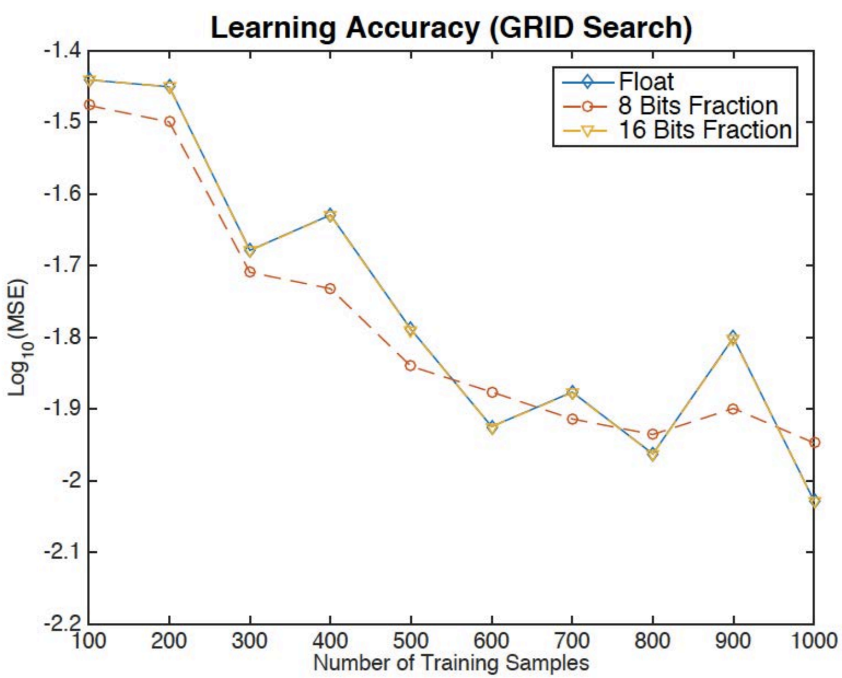
- Energy efficient, Parallelism (pipelining), Integrated with PCIe and Customised (problem changed to remove dependencies)
- Can do online learning from 200 independent data streams at 70 Gbps (160 GFLOPS)

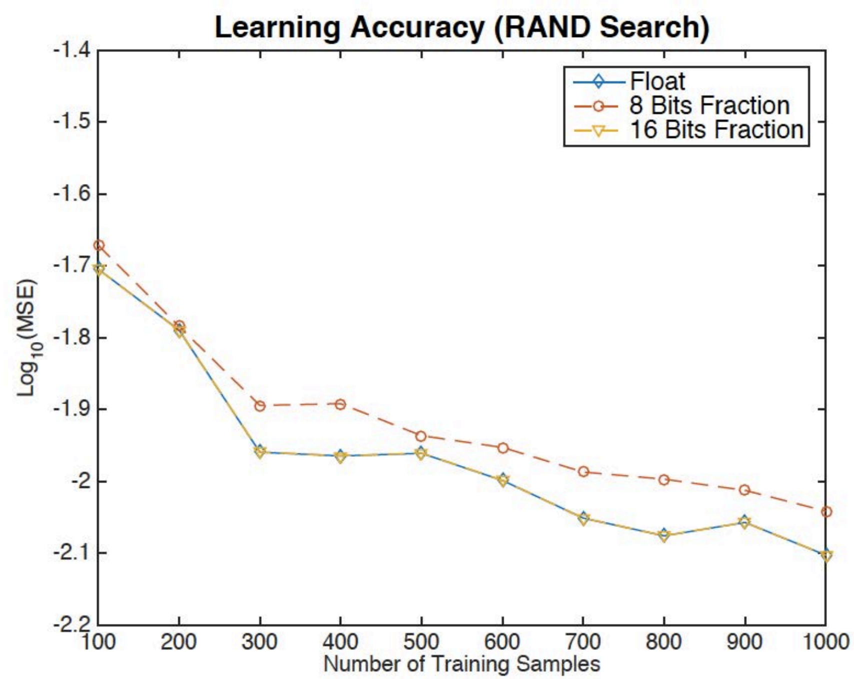
Precision and Learning Accuracy for KNLMS

- Trained and Tested with Mackey Glass data

- Trained with 100-1000 data samples/Tested with 999 data samples

- Hyperparameter set chosen from Grid/Random Search





- 16 fraction bits are enough to produce equivalent learning accuracies with Float

Overview

- Motivation
- Kernel methods
  - Vector processor
  - Pipelined
  - Braided
  - Distributed
- Conclusion

NORMA

Naive Online regularised Risk Minimization Algorithm

› Finds D (dictionary which is subset of input vectors), and α (weights) for function

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

› Minimise instantaneous risk of predictive error ($R_{inst,\lambda}$) by taking a step in direction of gradient

$$f_{t+1} = f_t - \eta_t \partial_f R_{inst,\lambda}[f, x_{t+1}, y_{t+1}]\Big|_{f=f_t}$$

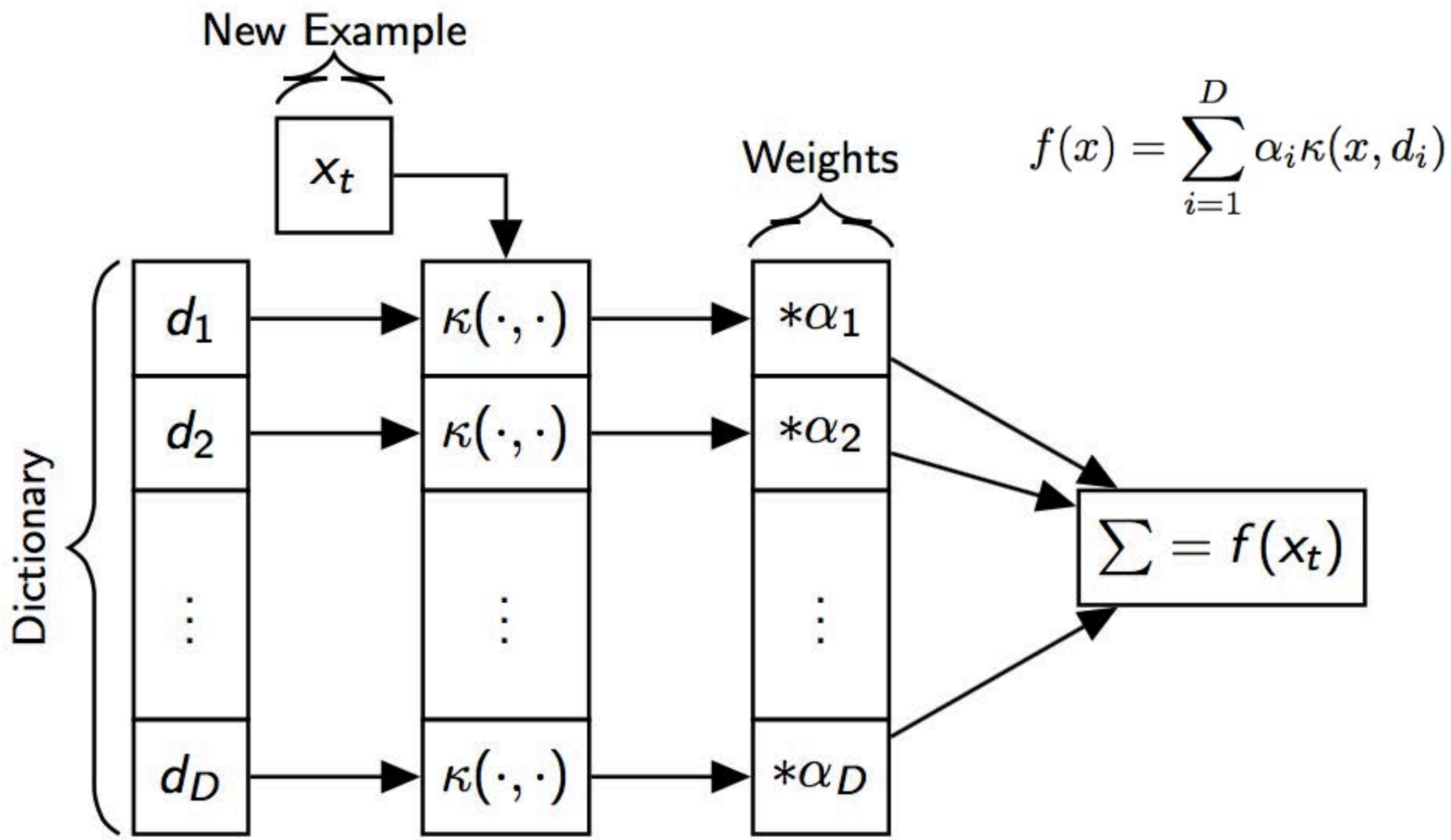
› Can be used for classification, regression, novelty detection

› Update for novelty detection

$$(\alpha_i, \alpha_t, \rho) = \begin{cases} (\Omega\alpha_i, 0, \rho + \eta\nu) \text{ if } f(x_t) \geq \rho \\ (\Omega\alpha_i, \eta, \rho - \eta(1-\nu)) \text{ otherwise} \end{cases}$$

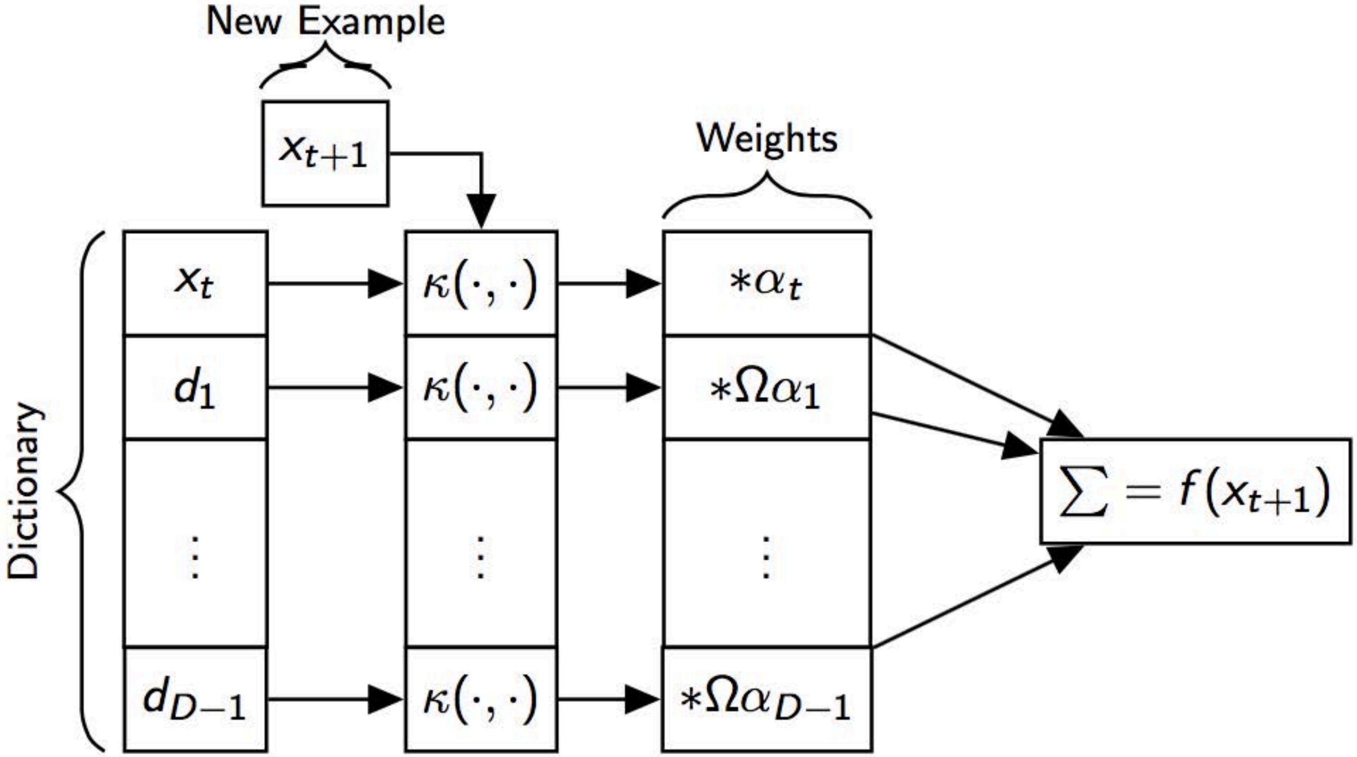
Add x_{t+1} to dictionary

Datapath for NORMA



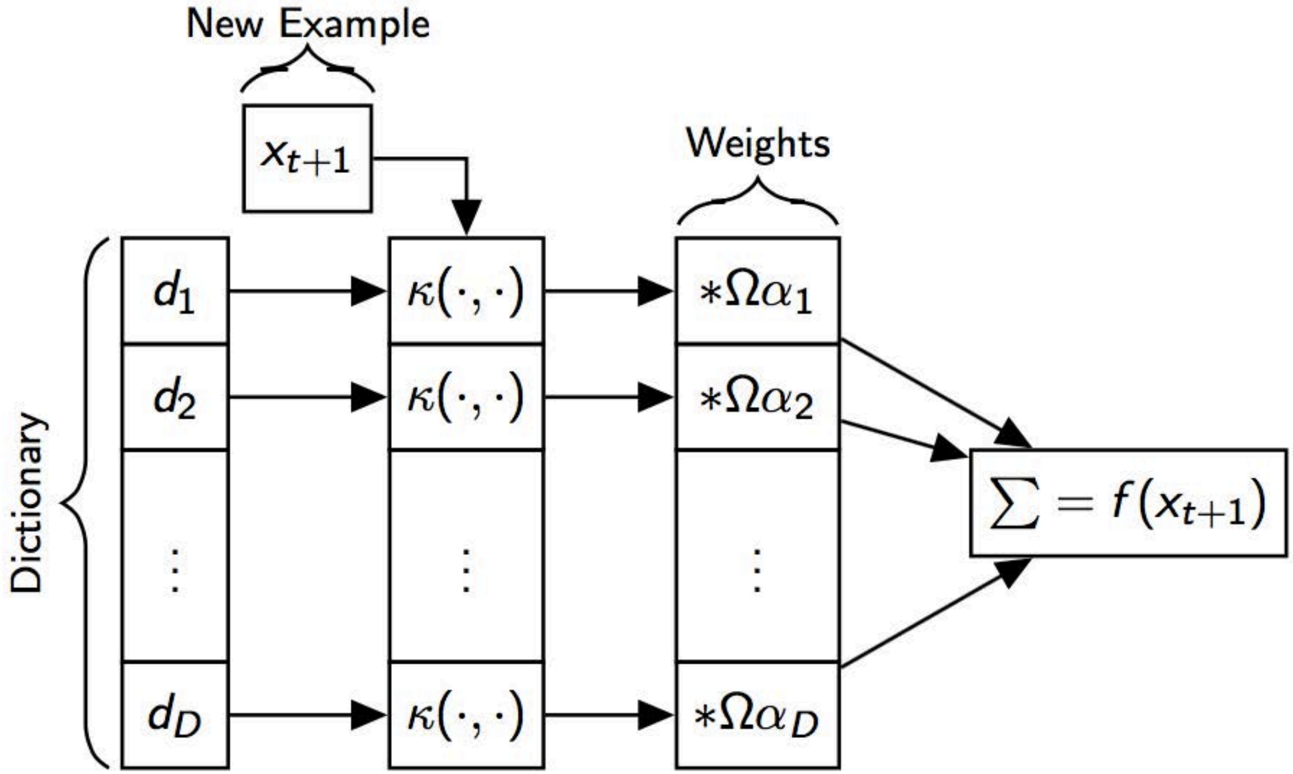
NORMA Update (Case 1)

$$(\alpha_i, \alpha_t, \rho) = \begin{cases} (\Omega\alpha_i, 0, \rho + \eta\nu) \text{ if } f(x_t) \geq \rho & \textbf{Add } \mathbf{x_{t+1}} \textbf{ to dictionary} \\ (\Omega\alpha_i, \eta, \rho - \eta(1-\nu)) \text{ otherwise} \end{cases}$$



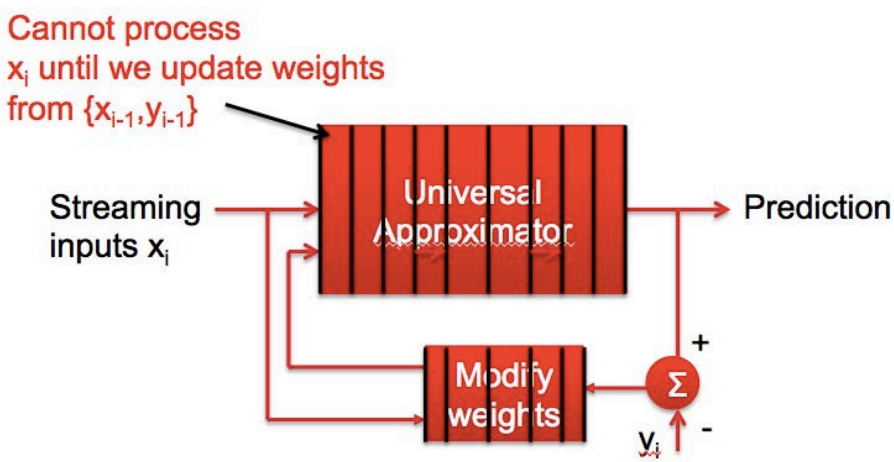
NORMA Update (Case 2)

$$(\alpha_i, \alpha_t, \rho) = \begin{cases} (\Omega\alpha_i, 0, \rho + \eta\nu) \text{ if } f(x_t) \geq \rho & \textbf{Add } \mathbf{x_{t+1}} \textbf{ to dictionary} \\ (\Omega\alpha_i, \eta, \rho - \eta(1-\nu)) \text{ otherwise} \end{cases}$$



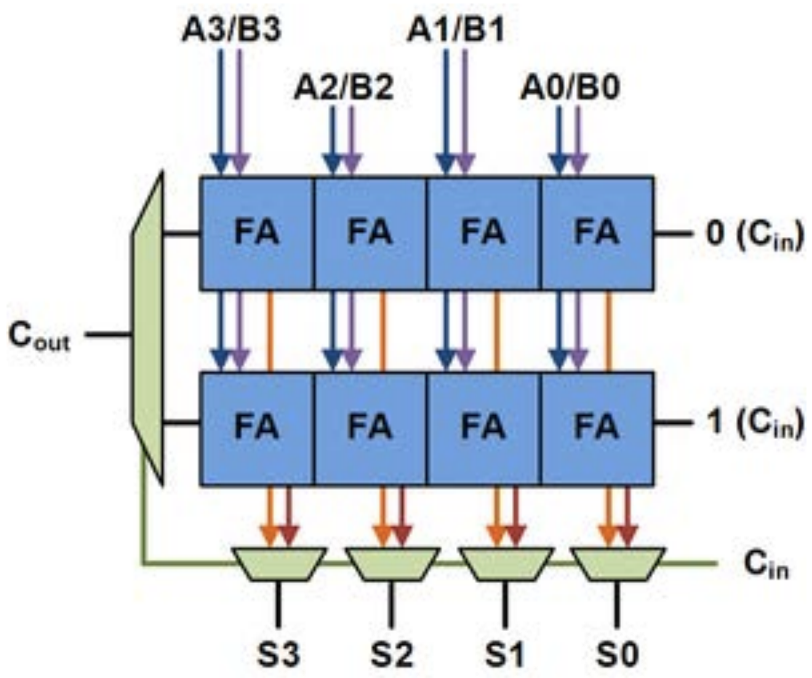
Properties of NORMA

- NORMA is a sliding window algorithm
  - If new dictionary entry added $[d_1,\cdots d_D] \rightarrow [x_t,d_1,\cdots d_{D-1}]$
  - Weight update is just a decay $\alpha_i \rightarrow \Omega\alpha_i$
  - Update cost is small compared to computing $f(x_t)$
- **Is this really true?**



Idea

› Recall carry select adder
  - implement both cases in parallel and select output



Braiding

$$f(x_{t+1}) = \sum_{i=1}^{D} \alpha_i \kappa(x_{t+1}, d_i)$$

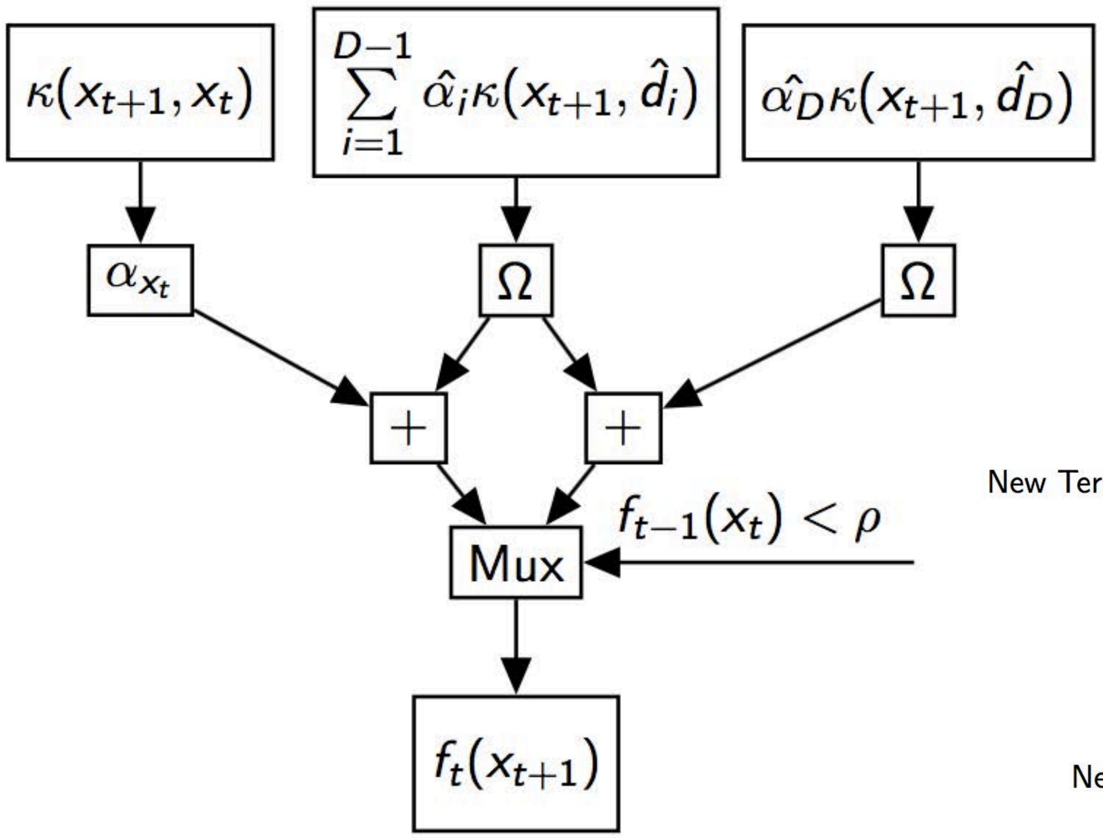
Use the previous dictionary for x_t denoted $\hat{d}_i$

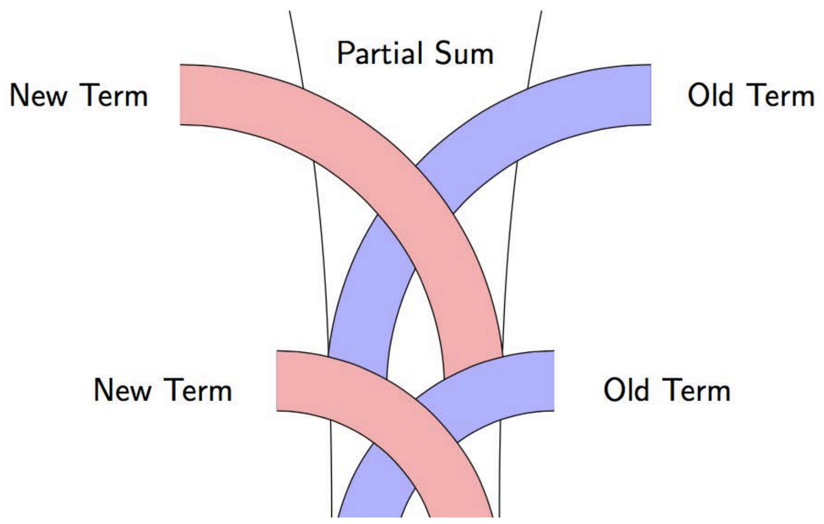
$$f(x_{t+1}) = \sum_{i=1}^{D-1} \Omega \hat{\alpha}_i \kappa(x_{t+1}, \hat{d}_i) + \text{something}$$

if x_t is added then this term $= \alpha_{x_t} \kappa(x_{t+1}, x_t)$

if x_t is not added then this term $= \Omega \hat{\alpha_D} \kappa(x_{t+1}, \hat{d}_D)$

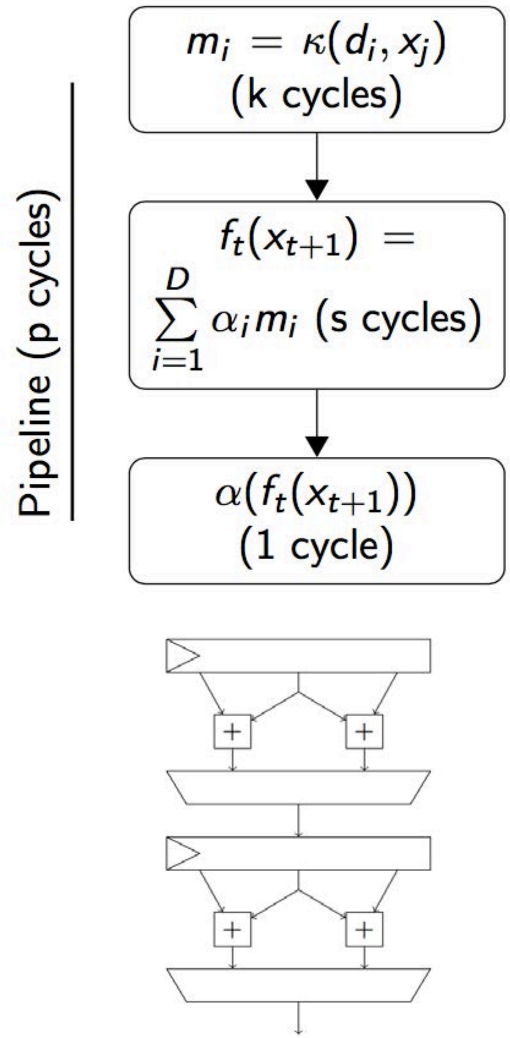
Braiding Datapath





Generalised to p cycles

$$
\begin{aligned}
f_t(x_{t+1}) = \sum_{i=1}^{D-p} \Omega^p \hat{\alpha}_i \kappa(x_{t+1}, \hat{d}_i) \\
q \left\{ \begin{array}{l}
+ \begin{cases} 0 \text{ if } x_{t+1-p} \text{ is not added} \\ \Omega^{p-1} \alpha_{x_{t+1-p}} \kappa(x_{t+1}, x_{t+1-p}) \text{ otherwise} \end{cases} \\
+ \begin{cases} 0 \text{ if } x_{t+2-p} \text{ is not added} \\ \Omega^{p-2} \alpha_{x_{t+2-p}} \kappa(x_{t+1}, x_{t+2-p}) \text{ otherwise} \end{cases} \\
\qquad \vdots \\
+ \begin{cases} 0 \text{ if } x_{t} \text{ is not added} \\ \alpha_{x_t} \kappa(x_{t+1}, x_t) \text{ otherwise} \end{cases}
\end{array} \right. \\
+ \sum_{i=D-p+1}^{D-q} \Omega^p \hat{\alpha}_i \kappa(x_{t+1}, \hat{d}_i)
\end{aligned}
$$



Implementation

- Implemented in Chisel
- On XC7VX485T- 2FFG1761C achieves ~133 MHz
- Area O(FDB^2) (F=dimensionality of input vector), time complexity O(FD)
- Speedup 500x compared with single core CPU i7-4510U (8.10 fixed)

F=8, D=	16	32	64	128	200
Frequency (MHz)	133	138	137	131	127
DSPs (/2,800)	309	514	911	1,679	2,556
Slices (/759,000)	4615	8194	14,663	29,113	46,443
Latency (cycles)	10	11	12	12	13
Speedup (×)	47	91	178	344	509
Latency reduction (×)	4.69	8.30	14.9	28.7	39.2

Comparison of Architectures

› Core with input vector F=8 and dictionary size D=16

Design	Precision	Freq MHz	Latency Cycles	T.put Cycles	Latency nS	T.put nS
Vector KNLMS	Single	282	479	479	1,699	1,699
Pipelined KNLMS	Single	314	207	1	659	3.2
Braided NORMA	8.10	113	10	1	89	8.8

Open source (GPLv2): github.com/da-steve101/chisel-pipelined-olk

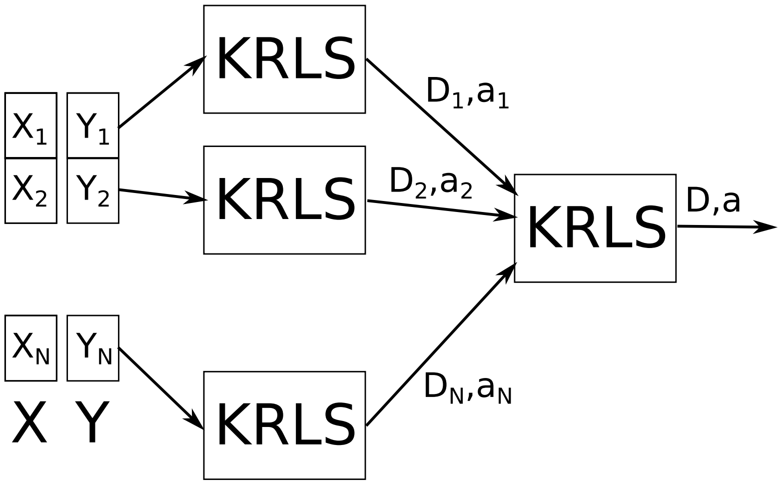
Overview

- Motivation
- Kernel methods
  - Vector processor
  - Pipelined
  - Braided
  - **Distributed**
- Conclusion

THE UNIVERSITY OF SYDNEY

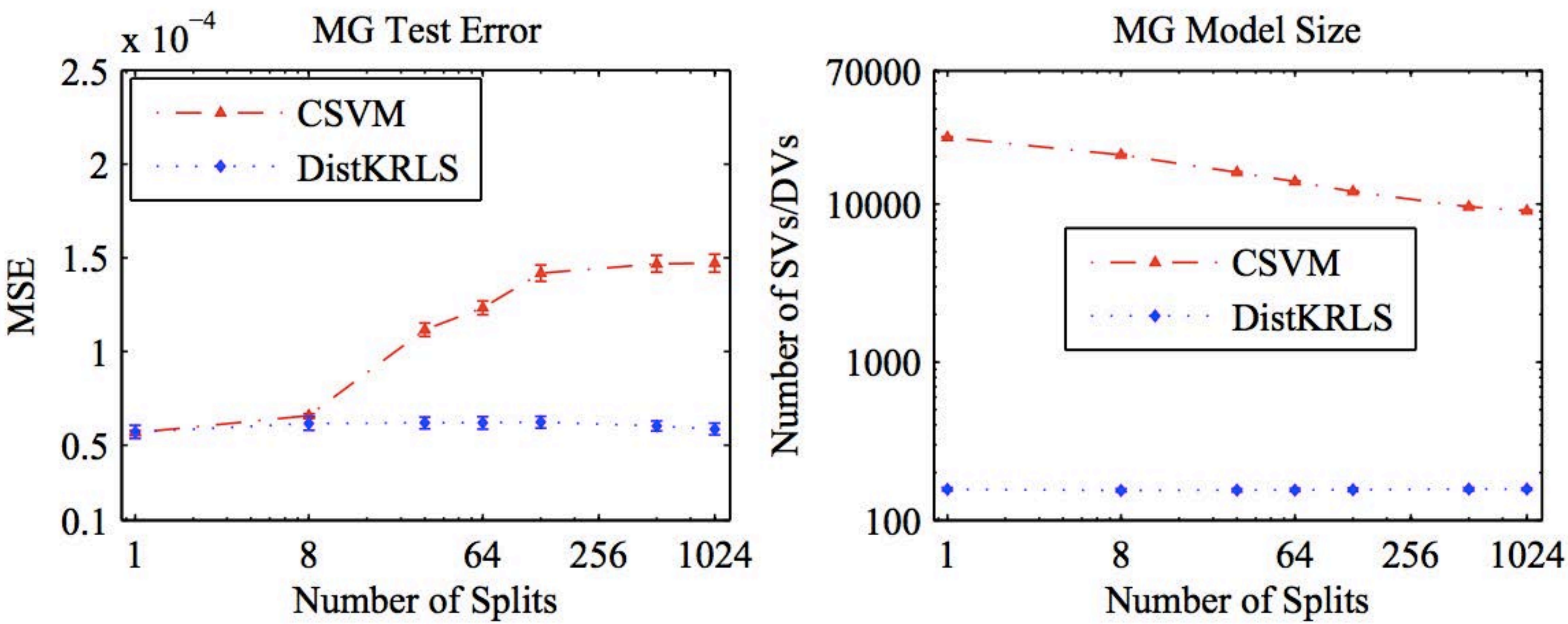
Distributed KRLS

- One problem with KRLS is how to get scalable parallelism
- Proposed a method, which uses KRLS (Engel et al. 2004) to create models on subsets of the data.
- These models can then be combined using KRLS again to create a single accurate model
  - We have shown an upper bound on the error introduced



Distributed KRLS Vs Cascade SVM

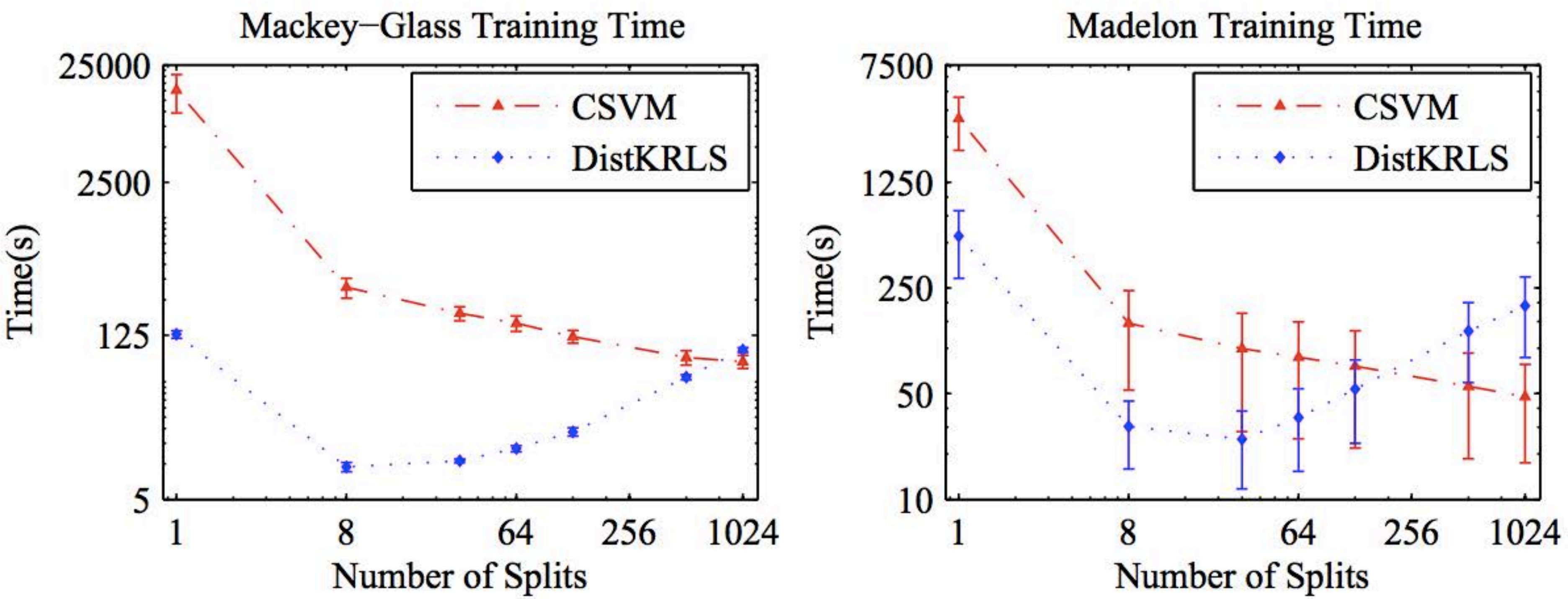
› Accuracy comparison



Speedup

Distributed KRLS Vs Cascade SVM

- Average Speedup about 20x on a 16 node cluster

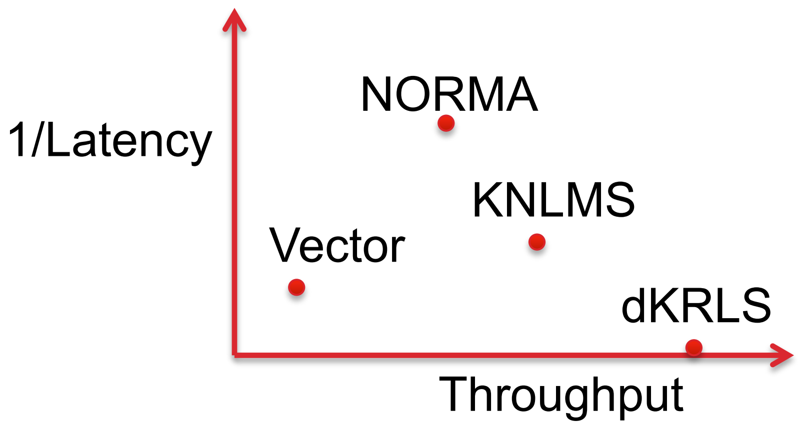


Overview

- Motivation
- Kernel methods
  - Vector processor
  - Pipelined
  - Braided
  - Distributed
- Conclusion

Conclusion

› Demonstrated high-performance applications in ML



› Machines of the future will need to interpret and process data using ML

- FPGAs are a key enabling technology for energy-efficient, fast implementations
- A lot more to do!

THE UNIVERSITY OF SYDNEY

References

› Stephen Tridgell, Duncan J.M. Moss, Nicholas J. Fraser, and Philip H.W. Leong. Braiding: a scheme for resolving hazards in NORMA. In Proc. International Conference on Field Programmable Technology (FPT), pages 136–143, 2015. (doi:10.1109/FPT.2015.7393140)

› Nicholas J. Fraser, Duncan J.M. Moss, JunKyu Lee, Stephen Tridgell, Craig T. Jin, and Philip H.W. Leong. A fully pipelined kernel normalised least mean squares processor for accelerated parameter optimisation. In Proc. International Conference on Field Programmable Logic and Applications (FPL), pages 1–6, 2015. (doi:10.1109/FPL.2015.7293952)

› Nicholas J. Fraser, Duncan J.M. Moss, Nicolas Epain, and Philip H.W. Leong. Distributed kernel learning using kernel recursive least squares. In Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 5500–5504, 2015. (doi:10.1109/ICASSP.2015.7179023)

› Yeyong Pang, Shaojun Wang, Yu Peng, Xiyuan Peng, Nicholas J. Fraser, and Philip H. W. Leong. A microcoded kernel recursive least squares processor using FPGA technology. ACM Transactions on Reconfigurable Technology and Systems, pages In press, 2016.