FPGA-based Distributed Edge Training of SVM

Jyotikrishna Dass, Yashwardhan Narawane, Rabi Mahapatra, Vivek Sarin
Texas A&M University

COMPUTER SCIENCE & ENGINEERING
TEXAS A&M UNIVERSITY

Overview

Goal

- ✓ To design and implement distributed training of machine learning, here, Support Vector Machines (SVM), on multiple FPGA system
- ✓ To reduce network communication while achieving fast training , memory-efficiency, and energy savings

Motivation

- SVM training in computationally expensive with high memory requirement for *kernel* matrix
- Traditional SVM training accelerators are based on inherently sequential algorithms using a single FPGA board
- Need to distribute and accelerate training on edge where the data is generated and stored across multiple devices

Background

Support Vector Machines

- Supervised machine learning algorithm for classification and regression problems
- Mathematically, a quadratic programming problem which solves for maximal separating hyperplane as a classifier

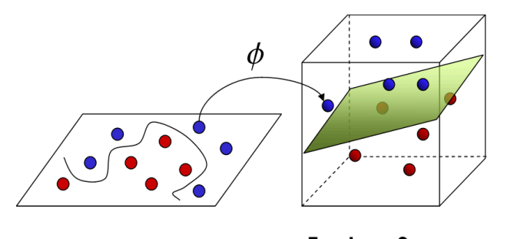


Fig. Kernel SVM: Learning hyperplane in higher feature dimension

training dataset, $\mathcal{D} = \{(x_i, y_i), i = 1, ..., n\}$
input data matrix, $X = \{x_i \in \mathbb{R}^d, i = 1...n\}$, d-dimensional space
class label vector, $y = \{y_i \in \{-1, 1\}, i = 1...n\}$

SVM *dual*

$$\min_{\alpha} \frac{1}{2}\alpha^T\left(diag(y) \times \mathbf{K} \times diag(y)^T\right)\alpha + \frac{1}{2}\alpha^T\left(\frac{1}{2C}I_n\right)\alpha + e^T\alpha$$

$$\text{subject to} \quad -I_n\alpha \le \mathbf{0}_n$$

where, α is a vector of *dual* variables
$e = -\mathbf{1}_n$
$C > 0$ is penalty parameter for misclassification
$\mathbf{K} = \{k(x_i, x_j), \forall i, j = 1...n\}$ is positive definite matrix (mostly)
$k()$ **represents the Mercer kernel function - linear/non-linear**

Substitute, $K \approx AA^T$, $A \in \mathbb{R}^{n\times k}$ and $k \ll n$.
Define, $\hat{A} = diag(y) \times A$
Substitute, $\hat{A} = QR$, where, $Q \in \mathbb{R}^{n\times n}$ is Orthogonal matrix and $R \in \mathbb{R}^{n\times k}$ is Upper Triangular matrix

QRSVM

$$\min_{\hat{\alpha}} \frac{1}{2}\hat{\alpha}^T\left(RR^T + \frac{1}{2C}I_n\right)\hat{\alpha} + (\hat{e})^T\hat{\alpha}$$

$$\text{subject to} \quad -Q\hat{\alpha} \le \mathbf{0}_n$$

where, $\hat{\alpha} = Q^T\alpha$, $\hat{e} = Q^Te$

Define, $F = -\left(R_gR_g{}^T + \frac{1}{2C}I_n\right)$ and $\hat{n} = \frac{n}{p}$

Step 1: Minimization of Lagrangian - In Parallel

At edge unit, i

$$\hat{\alpha}_i{}^{k+1} = F_i^{-1}(-\hat{\beta}_i{}^k + \hat{e}_i)$$

where,

$$F_i{}^{-1} = \begin{cases} F_1{}^{-1} & \text{if} \quad i = 1 \\ -2C & \text{if} \quad i = 2..p \end{cases}$$

Step 2: Dual variable update - In Parallel

At edge unit, i

$$\hat{\beta}_i{}^{k+1} = \hat{\beta}_i{}^k + \eta^\star(-\hat{\alpha}_i{}^{k+1})$$

$\eta^\star$ is the Optimal step size
$\hat{\beta}^k = Q^T\beta^k$

Distributed QRSVM

- QR decomposition-based distributed SVM
- Memory-efficient + negligible Communication
- Comprises of 3 stages:
  1. Initialization
  2. Distributed QR decomposition (formulation)
  3. Parallel Dual Ascent (solver)

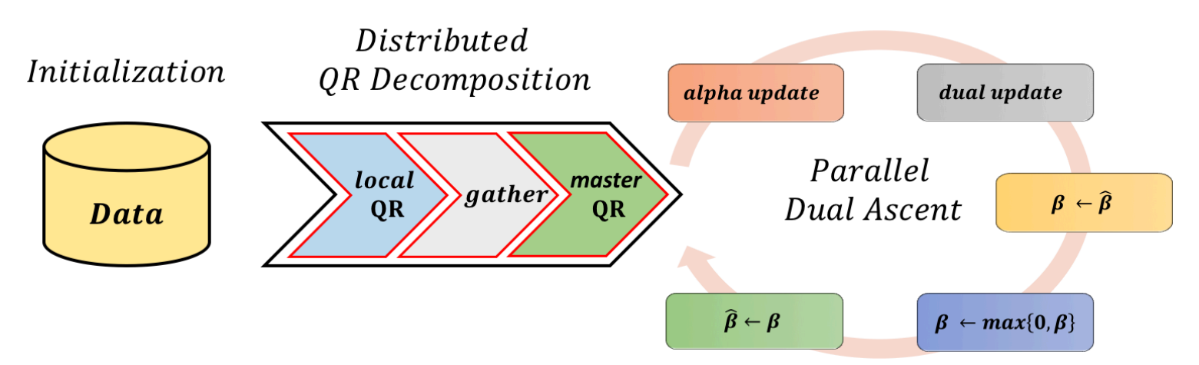

Initialization
Data
Distributed QR Decomposition
local QR
gather
master QR
alpha update
dual update
Parallel Dual Ascent
$\beta \leftarrow \hat{\beta}$
$\hat{\beta} \leftarrow \beta$
$\beta \leftarrow max\{0, \beta\}$

Fig. Process Flow for Distributed QRSVM algorithm

Algorithmic Design

Distributed QR Decomposition

```
Algorithm 1 Distributed QR decomposition
1: $k \leftarrow$ rank
2: for each edge $i$ do
3:     $\hat{A}_i \leftarrow$ local partitioned data
4:     Parallel Compute $\{q_i\}, R_i \leftarrow \hat{A}_i$        ▷ Algorithm 2
5:     GATHER $(R_i)_{k\times k}$ at Master unit
6: end for
7: $\hat{A}_g \leftarrow$ gathered or stacked $(R_i)_{k\times k}$
8: Compute $\{q_f\}, R_f \leftarrow \hat{A}_g$ at Master unit      ▷ Algorithm 2
9: Use $(R_f)_{k\times k}$
```

```
Algorithm 2 $\{q_i\}, R_i \leftarrow \hat{A}_i$, via Householder algorithm
1: $Q_{\hat{n}\times k}, (\hat{A}_i)_{\hat{n}\times k}$        ▷ $\hat{n}$ : samples per edge
2: for $j \leftarrow 1$ to $k$ do
3:     $q_{ij} \leftarrow \hat{A}_i(j:\hat{n}, j)$
4:     $q_{ij}(1) \leftarrow q_{ij}(1) + sign(q_{ij}(1)) \times \|q_{ij}\|_2$    ▷ scalar update
5:     $q_{ij} \leftarrow \frac{q_{ij}}{\|q_{ij}\|_2}$        ▷ vector normalization
6:     $\hat{A}_i(j:\hat{n}, j:k) \leftarrow \hat{A}_i(j:\hat{n}, j:k) - 2q_{ij} < q_{ij}, \hat{A}_i(j:\hat{n}, j:k) >$    ▷ Algorithm 4
7:     $R_i = \hat{A}_i(j:\hat{n}, j:k)$
8: end for
9: $\{q_i\} \leftarrow [q_{i1}, q_{i2}, \ldots, q_{ik}]$        ▷ set of k-reflectors
```

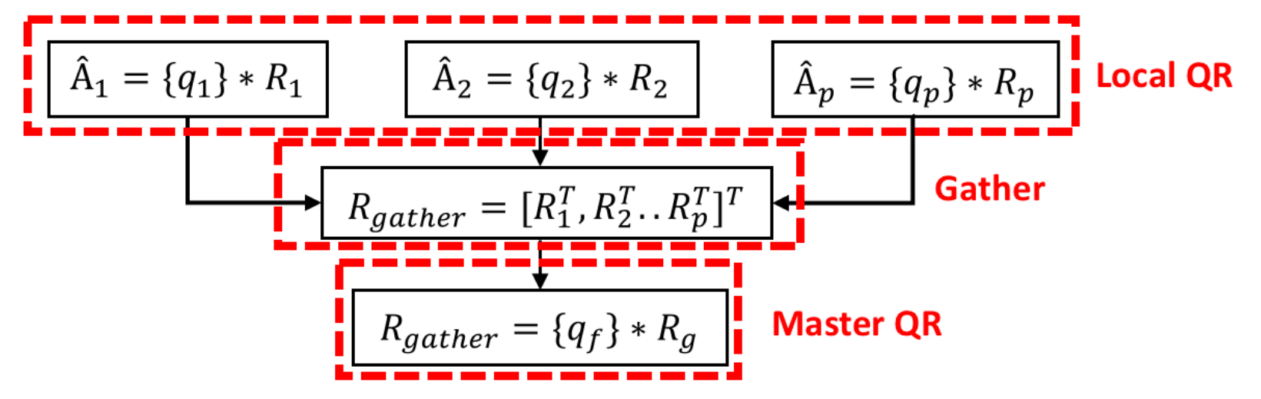

$\hat{A}_1 = \{q_1\} * R_1$
$\hat{A}_2 = \{q_2\} * R_2$
$\hat{A}_p = \{q_p\} * R_p$
Local QR
$R_{gather} = [R_1^T, R_2^T..R_p^T]^T$
Gather
$R_{gather} = \{q_f\} * R_g$
Master QR

Fig. Computational flow graph for distributed QR decomposition

```
Algorithm 4 Computing Step 6 in Algorithm 2
1: ▷ $\hat{A}_i(j:\hat{n}, j:k) \leftarrow \hat{A}_i(j:\hat{n}, j:k) - 2q_{ij} < q_{ij}, \hat{A}_i(j:\hat{n}, j:k) >$
2: for $m \leftarrow j$ to $k$ do
3:     $a_{BRAM} \leftarrow A(j:\hat{n}, m)$        ▷ Load into BRAM
4:     Compute $sum =< q_{ij}, a_{BRAM} >$        ▷ Inner Product
5:     $a_{BRAM} \leftarrow a_{BRAM} - 2 \times sum \times q_{ij}$        ▷ saxpy
6:     $A(j:n, m) \leftarrow a_{BRAM}$        ▷ Write to DDR
7: end for
```

Parallel Dual Ascent

```
Algorithm 3 Parallel Dual Ascent
1: iteration $t \leftarrow 0$
2: for each edge $i$ do
3:     while $error > threshold$ do
4:         Parallel Compute: $\hat{\alpha}_i{}^{t+1}$        ▷ alpha update
5:         Parallel Compute: $\hat{\beta}_i{}^{t+1}$        ▷ dual update
6:         Compute $\beta_i \leftarrow Q\hat{\beta}$   - - -> $\hat{\beta}_{i[j:\hat{n}]} \leftarrow \hat{\beta}_{i[j:\hat{n}]} - 2q_{ij} < q_{ij}, \hat{\beta}_{i[j:\hat{n}]} >$   $j \leftarrow k$ to $1$
7:         Parallel Compute $\beta_i \leftarrow max\{0, \beta_i\}$
8:         Compute $\hat{\beta}_i \leftarrow Q^T\beta$   - - -> $\beta_{i[j:\hat{n}]} \leftarrow \beta_{i[j:\hat{n}]} - 2q_{ij} < q_{ij}, \beta_{i[j:\hat{n}]} >$   $j \leftarrow 1$ to $k$
9:         $error \leftarrow |\hat{\beta}_i{}^{t+1} - \hat{\beta}_i{}^t|$
10:        $t \leftarrow t + 1$
11:    end while
12: end for
```

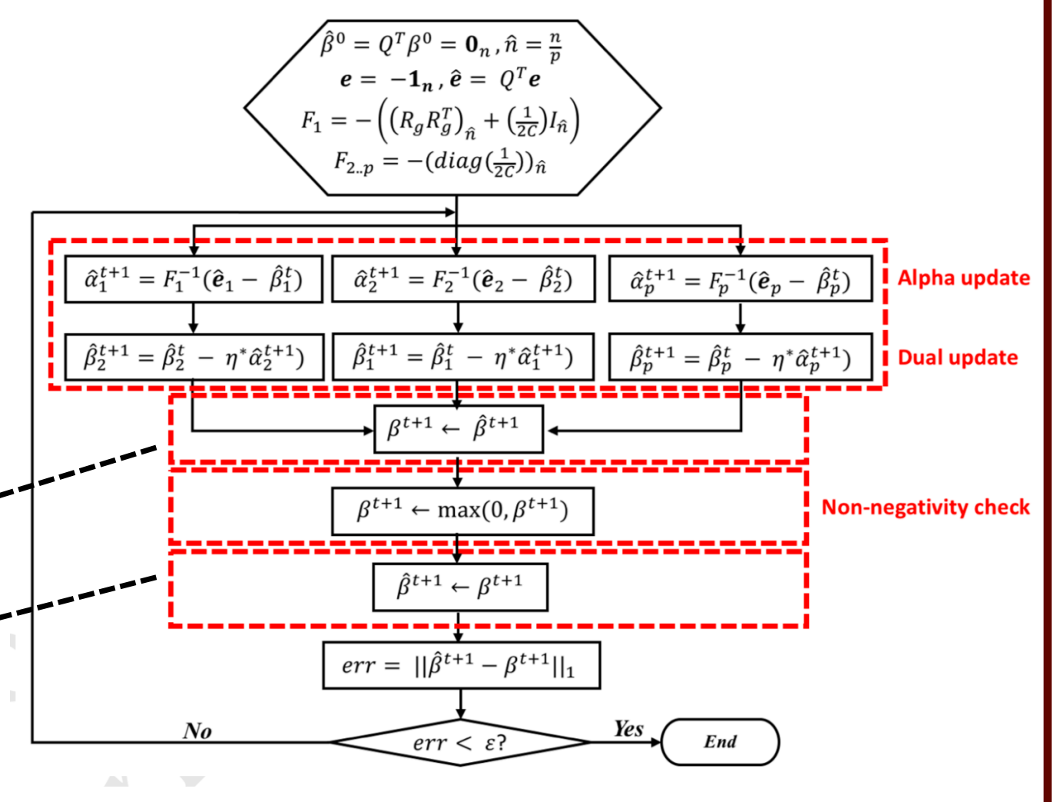

$\hat{\beta}^0 = Q^T\beta^0 = \mathbf{0}_n, \hat{n} = \frac{n}{p}$
$e = -\mathbf{1}_n, \hat{e} = Q^Te$
$F_1 = -\left((R_gR_g^T)_{\hat{n}} + (\frac{1}{2C})I_{\hat{n}}\right)$
$F_{2..p} = -(diag(\frac{1}{2C}))_{\hat{n}}$
$\hat{\alpha}_1^{t+1} = F_1^{-1}(\hat{e}_1 - \hat{\beta}_1^t)$
$\hat{\alpha}_2^{t+1} = F_2^{-1}(\hat{e}_2 - \hat{\beta}_2^t)$
$\hat{\alpha}_p^{t+1} = F_p^{-1}(\hat{e}_p - \hat{\beta}_p^t)$
Alpha update
$\hat{\beta}_2^{t+1} = \hat{\beta}_2^t - \eta^*\hat{\alpha}_2^{t+1})$
$\hat{\beta}_1^{t+1} = \hat{\beta}_1^t - \eta^*\hat{\alpha}_1^{t+1})$
$\hat{\beta}_p^{t+1} = \hat{\beta}_p^t - \eta^*\hat{\alpha}_p^{t+1})$
Dual update
$\beta^{t+1} \leftarrow \hat{\beta}^{t+1}$
$\beta^{t+1} \leftarrow \max(0, \beta^{t+1})$
Non-negativity check
$\hat{\beta}^{t+1} \leftarrow \beta^{t+1}$
$err = \|\hat{\beta}^{t+1} - \beta^{t+1}\|_1$
No
$err < \varepsilon?$
Yes
End

Fig. Computational flow graph for parallel dual scent

Hardware Implementation

Basic Computational Kernels

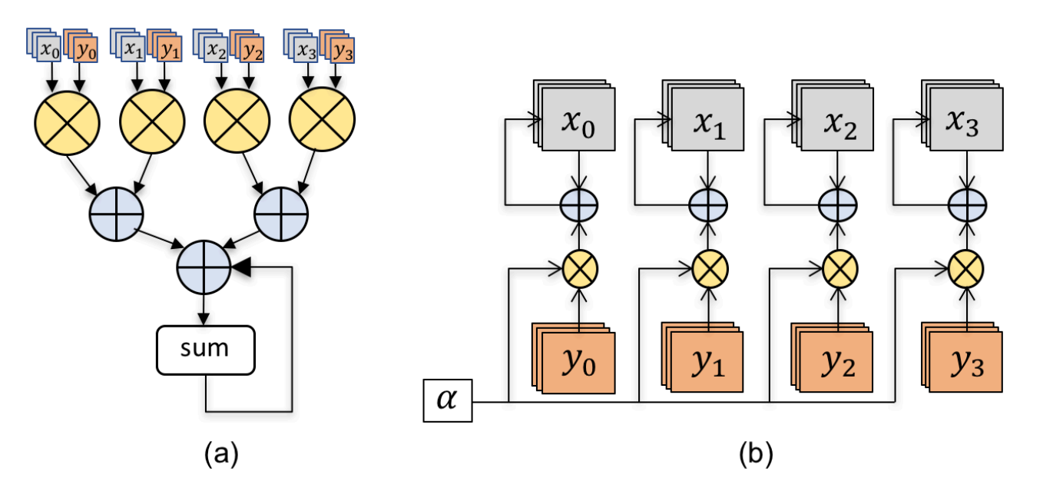


Fig. SIMD design for computational kernel (a) Inner Product *sum* = <***x,y***> via binary reduction tree (b) Scaled vector addition SAXPY, (***x*** = ***x*** +α ***y***) via fine-grained parallelism. The vectorized kernels operate on *W=floor(N/B)* data samples in each pass

Data Layout + Memory Interface

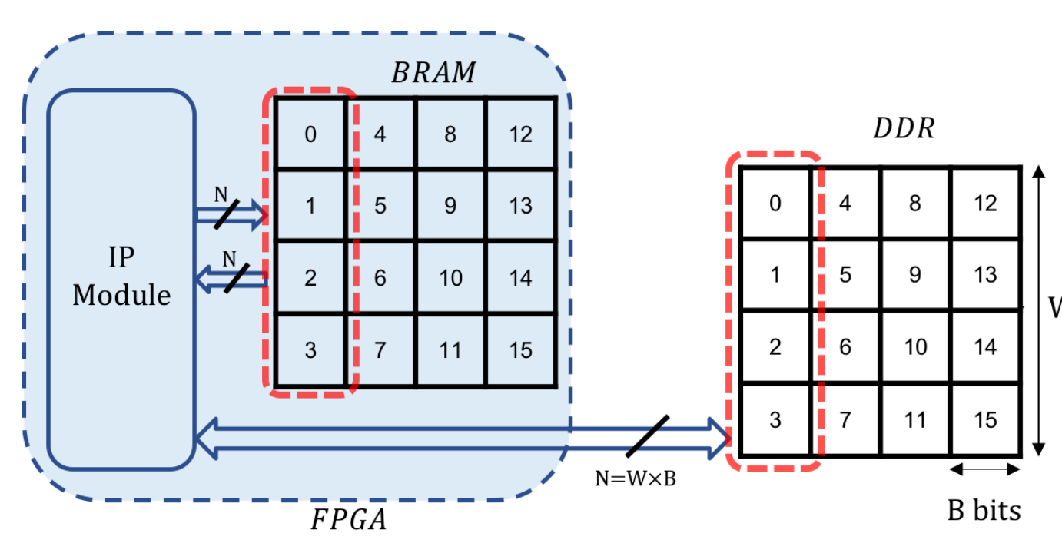


Fig. Data layout in column-major order. Memory-IP interface for on-chip Block RAM (full duplex) is called *bram* and that for off-chip DDR (half-duplex) is called *AXI Master*. N_{bram}=*1024 bits* , N_{AXI} = *2048 bits*

Increasing Throughput

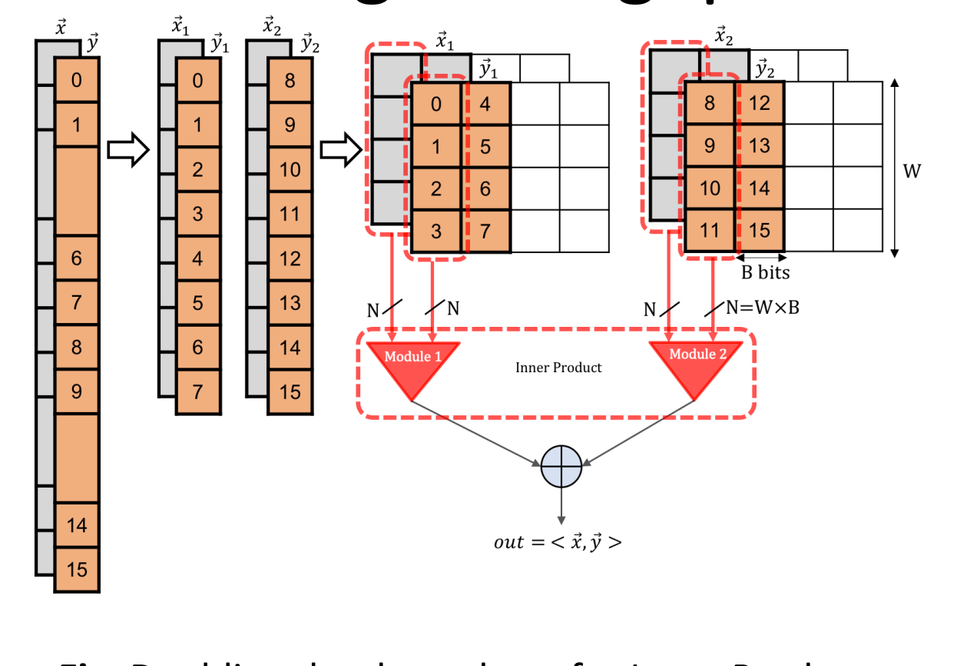


Fig. Doubling the throughput for Inner Product
<x,y> = <x_1,y_1> + <x_2,y_2>

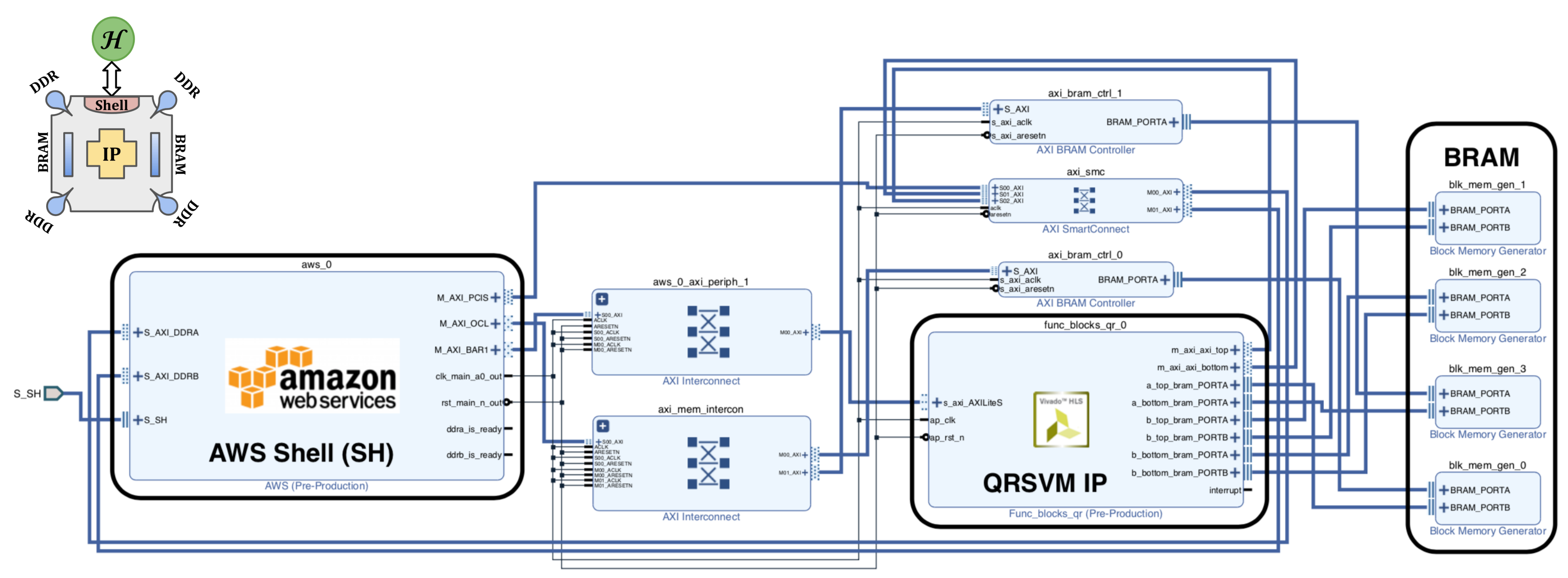
Memory Assignment

Kernel	BRAM	DDR
Inner Product	sum	x, y
SAXPY	x	y

FPGA Xilinx Virtex xcvu9p-flgb2104-2-i

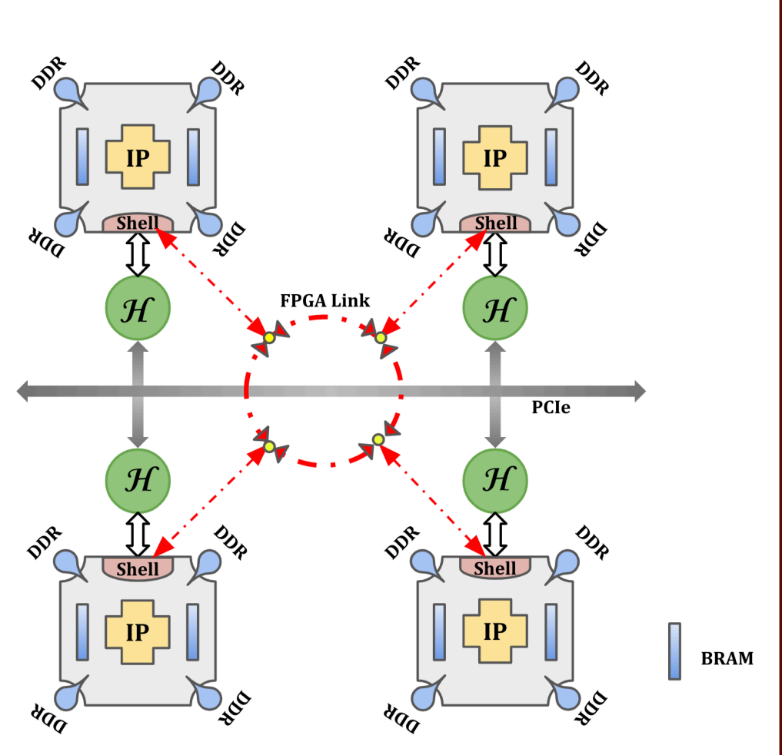
Resource	BRAM	DSP	FF	LUT
Available	2160	6840	2.36M	1.18M
Used	1405	1221	545K	450K
%	65%	18%	23%	38%

FPGA-Architecture for distributed SVM training



Multiple FPGA Network

- Each edge device comprises of FPGA IP logic + Host processor
- Illustration for $p = 4$ edges in a network
- A single QRSVM IP is synthesized per FPGA device to operate at clock frequency of *125 MHz* with *39 Watts* of power
- Computational workload is entirely with the FPGA while communication is through Host processor (*PCIe*)
- Each FPGA handles maximum of *256K* samples



Experimental Results and Discussions

Hardware Platform

Proof-of-Concept implemented on Amazon AWS F1 instance with *p={1,2,4,8}* 16nm Xilinx Virtex Ultrascale+ VU9P FPGA units forming a multiple FPGA network

Dataset Description

Benchmark	Application	#samples (n)	#features (d)	k-rank
MNIST	Image	60,000	780	128
Skin	Health	200,000	3	64
Webspam	Email	350,000	254	128
Covtype	Geography	464,810	54	64
SUSY	Physics	2,000,000	18	128

Training Time

#units	#iterations	MNIST ($C = 1, \gamma = 2^{-6}, \eta^* = 0.9$)				
p	t	T_{QR}	T_{DA}	T_p^{FPGA}	*comp*	*comm*
1	181	3.42	7.49	10.92	99.9%	0.1%
2	181	1.76	4.07	5.84	99.8%	0.2%
4	182	0.93	2.51	3.58	96%	4%
8	182	0.46	2.14	2.61	99.6%	0.4%

#units	#iterations	Skin ($C = 1, \gamma = 2^{-8}, \eta^* = 0.9$)				
p	t	T_{QR}	T_{DA}	T_p^{FPGA}	*comp*	*comm*
1	67441	2.80	4533	4536	99.9%	0.1%
2	64424	1.46	2226	2228	99.9%	0.1%
4	59761	0.74	1107	1108	99.9%	0.1%
8	54744	0.38	625	626	99.9%	0.1%

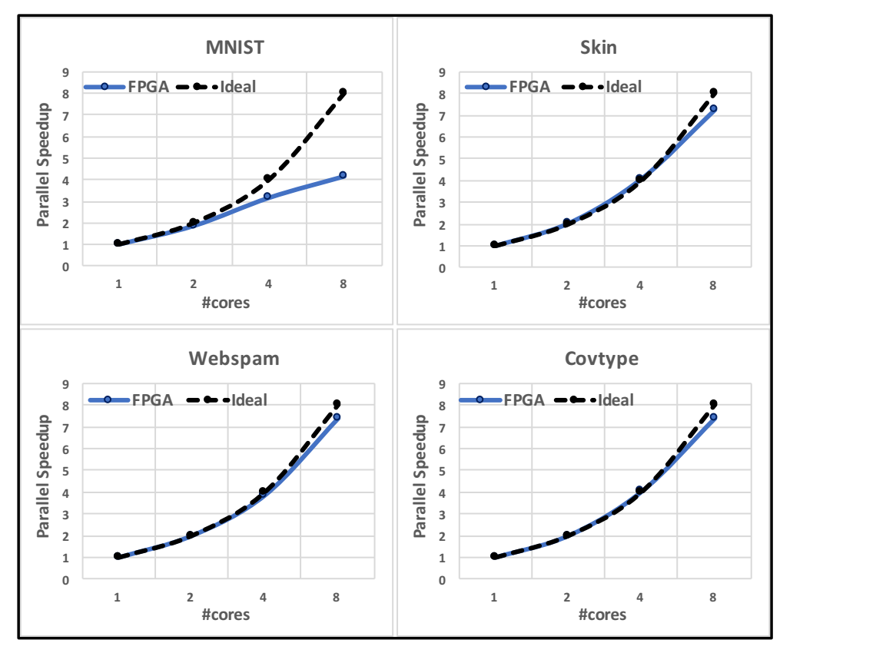
#units	#iterations	Webspam ($C = 1, \gamma = 1, \eta^* = 0.9$)				
p	t	T_{QR}	T_{DA}	T_p^{FPGA}	*comp*	*comm*
1	-	-	-	-	-	-
2	566	9.80	66.20	76.14	99.8%	0.2%
4	564	4.88	34.40	39.36	99.8%	0.2%
8	569	2.60	17.92	20.59	99.7%	0.3%

#units	#iterations	Covtype ($C = 1, \gamma = 2^3, \eta^* = 0.9$)				
p	t	T_{QR}	T_{DA}	T_p^{FPGA}	*comp*	*comm*
1	-	-	-	-	-	-
2	1125	3.35	88.02	91.45	99.9%	0.1%
4	1080	1.70	43.58	45.36	99.8%	0.2%
8	1068	0.80	23.80	24.75	99.4%	0.6%

#units	#samples	SUSY ($C = 1, \gamma = 2^{-3}, \eta^* = 0.9$)				
p	n	T_{QR}	T_{DA}	T_p^{FPGA}	*comp*	*comm*
1	250K	14.01	94.04	108.08	99.9%	0.1%
2	500K	14.04	116.8	131.02	99.8%	0.2%
4	1M	14.07	162.1	176.18	99.9%	0.1%
8	2M	14.14	285.47	299.63	99.9%	0.1%

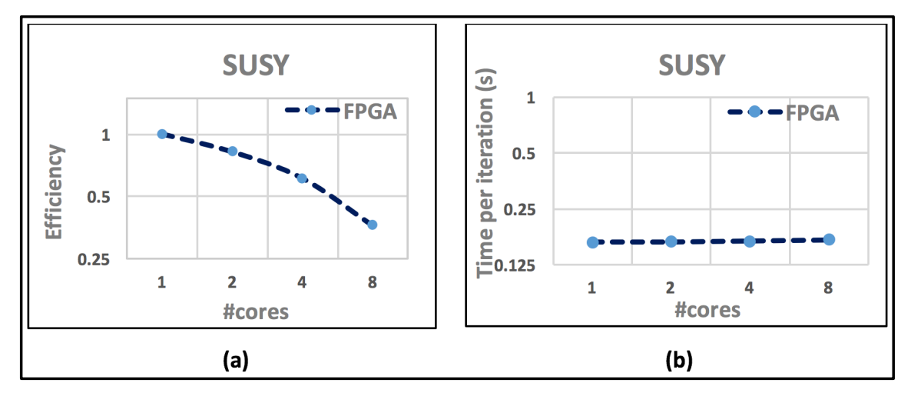
- **Parallel Dual Ascent** is computationally dominant than Distributed QR decomposition
- Near negligible communication overhead **<1%**

Strong Scaling Analysis



- Achieves near **linear parallel speedup** for larger datasets Skin, Webspam, Covtype
- For small dataset MNIST, going beyond *p = 4* seems to be overkill

Weak Scaling Analysis



- Workload per FPGA fixed at *250K* samples
- (a) T_{QR} is constant while $T_{DA} \uparrow$ with #iterations
- (b) (T_p^{FPGA}/t) is constant as desired

Energy Analysis

- Under strong scaling, the proposed FPGA design follows the **ideal energy consumption** trend that is **constant** across #FPGA units.
- Validates **fully parallel** implementation

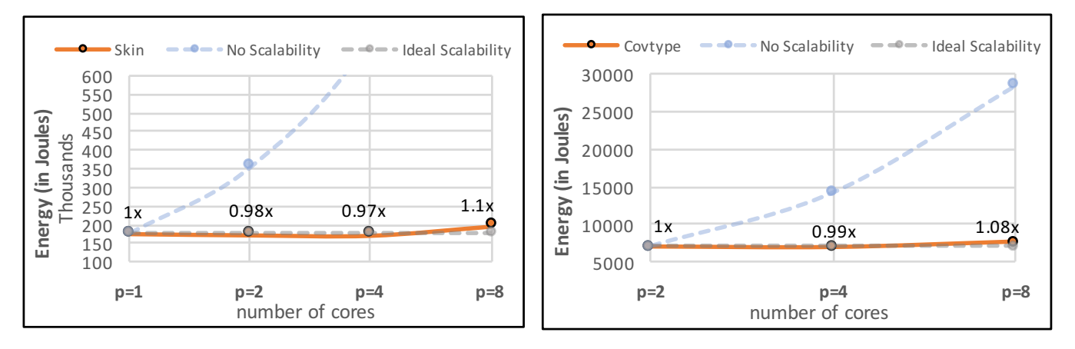


Fig. Energy consumption under Strong Scaling Skin and Covtype

- Under weak scaling, the ideal energy consumption trend is linear while no scalability trend is quadratic.
- The proposed design is closer to being linear than quadratic. Aberration at *p=8* due to large #iterations for fine tuning model with increasing overall problem size
- $(Energy/p)$ is nearly **constant** as expected with uniform workload per device

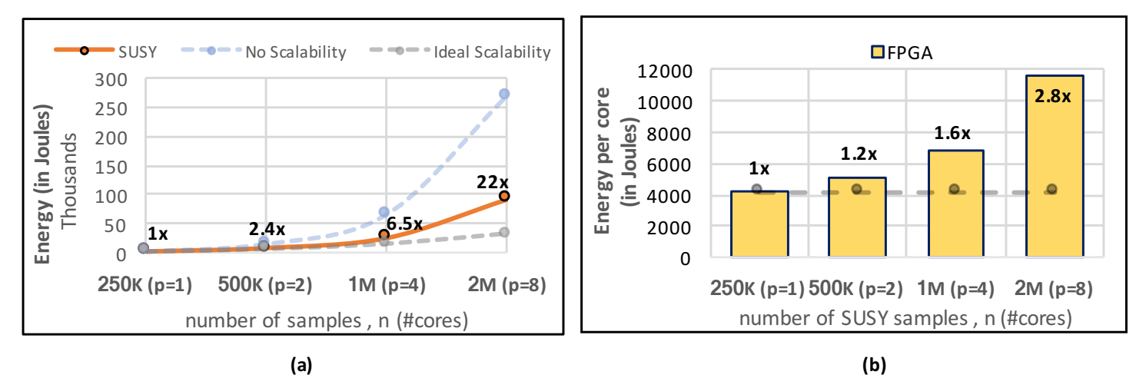


Fig. (a) Total Energy (b) Energy per core consumption under Weak Scaling for SUSY

Future Work

- ❑ To design and implement distributed training for Deep Learning models for resource constrained devices with limited memory and low power
- ❑ To explore online/incremental learning capabilities for machine learning models at edge

References

1. Burges, Christopher JC. “A tutorial on support vector machines for pattern recognition.” Data mining and knowledge discovery 2, no. 2 (1998): 121-167.
2. Papadonikolakis, Markos, and Christos-Savvas Bouganis. "A scalable FPGA architecture for non-linear SVM training." In 2008 International Conference on Field-Programmable Technology, pp. 337-340. IEEE, 2008.
3. Cadambi, Srihari, Igor Durdanovic, Venkata Jakkula, Murugan Sankaradass, Eric Cosatto, Srimat Chakradhar, and Hans Peter Graf. "A massively parallel FPGA-based coprocessor for support vector machines." In 2009 17th IEEE Symposium on Field Programmable Custom Computing Machines, pp. 115-122. IEEE, 2009.
4. Rabieah, Mudhar Bin, and Christos-Savvas Bouganis. "FPGA based nonlinear support vector machine training using an ensemble learning." In 2015 25th International Conference on Field Programmable Logic and Applications (FPL), pp. 1-4. IEEE, 2015.
5. Si, Si, Cho-Jui Hsieh, and Inderjit S. Dhillon. "Memory efficient kernel approximation." The Journal of Machine Learning Research 18, no. 1 (2017): 682-713.
6. Dass, Jyotikrishna, Vivek Sarin, and Rabi N. Mahapatra. "Fast and Communication-Efficient Algorithm for Distributed Support Vector Machine Training." IEEE Transactions on Parallel and Distributed Systems (2018).