Petrakos Emmanouil
Electrical & Computer Engineering School
Technical University of Crete

Reconfigurable Logic (FPGA)-based System Architecture for the Acceleration of Federated Learning in Neural Networks

Outline

Introduction

What is Federated Learning (FL)?

FL is a decentralized & collaborative training method for AI models

- training at the data collection point (edge devices)
- edge devices share model weights instead of raw data
- can exploit sensitive data, while ensuring privacy
- compliant with recent legislation like GDPR & CCPA
- used by Google to train Gboard

Motivation

FL literature mostly disregard the underlying hardware of the edge devices, where the training actually takes place.

- Few wall-clock time examples.
- Interactions between training hardware and FL remain unexplored.
- Are FPGAs compatible and efficient with on-edge FL?

Contributions

In this thesis:

- developed FL system, usable with any model, training method & implementation
- conducted in-depth robustness analysis of FL with small client pools
- implemented training of a CNN on a FPGA, optimized for the parameter space where FL is most efficient
- produced wall-clock timings by actual runs on real hardware
- compared with equivalent CPU & GPU implementations

Theoretical Background

Artificial Neuron

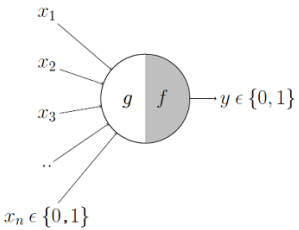


Figure: McCulloch-Pitts Neuron

Fundamental element of Artificial Neural Networks (ANNs)

$$y = \Phi(b + \sum_{i=1}^{I} x_i * w_i)$$

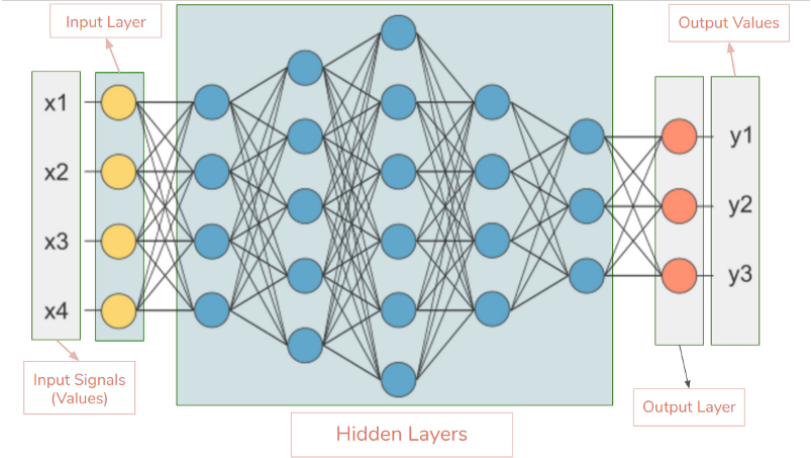
ReLU:

$$\Phi(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases}$$

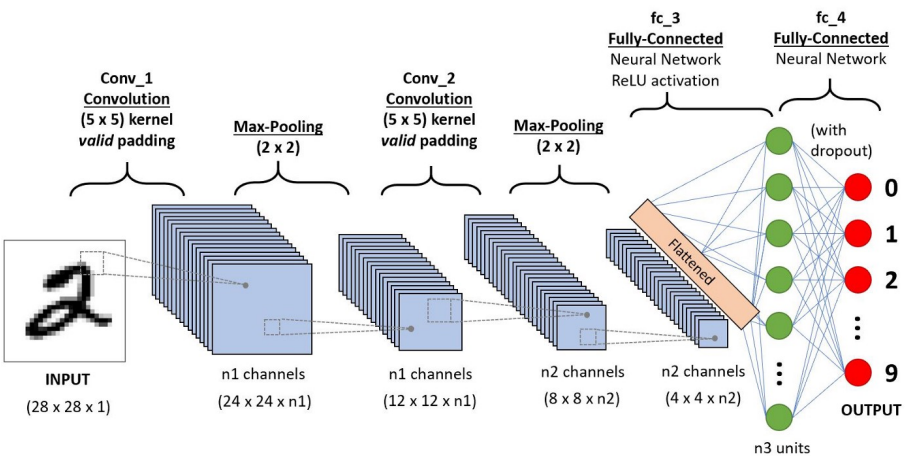
Softmax:

$$\Phi(x)_i = \frac{e^{x_i}}{\sum_{k=1}^{K} e^{x_j}}$$

Deep Neural Network



Convolutional Neural Network



Training Artificial Neural Networks

Supervised learning:

1. forward propagation
2. back propagation
3. calculate gradients
4. update parameters

Batching: Repeats steps 1 to 3 multiple times before running step 4.

FL Entities

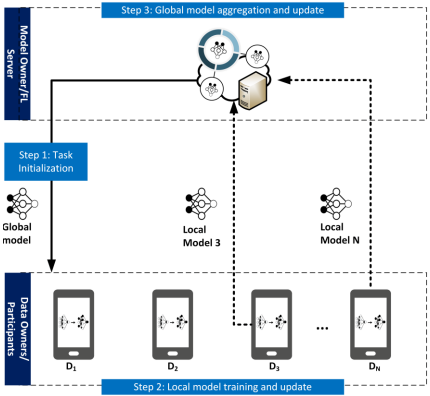
Server:

- model owner
- orchestrates training
- no access to any data

Clients:

- data owners
- train the model
- share the locally trained model

Typical Federated Training Process



Typical FL topology

Global Epoch (GE):

1. Server broadcast global model to clients.
2. Clients generate local models using the global model and their private data.
3. Server receives local models and generate new global model.

Related Work

Communication-Efficient Learning of Deep Networks from Decentralized Data

by HB McMahan et al.

- Lays down the fundamentals concepts and algorithms of FL
- Focuses on algorithmic convergence and tries to minimize the number of GEs
- Does not take in consideration the latency of each GE

PipeFL: Hardware/Software co-Design of an FPGA Accelerator for Federated Learning

by Zixiao Wang et al.

- Integrated an FPGA-based accelerator in an open-source FL framework
- Fully focused on the data-center FL setting, using high-end FPGAs
- Generic architecture that can compute any model, but not optimized for any single one
- Compares only against a CPU implementation

Unique Characteristics and Challenges of FL

FL literature has identified the following characteristics:

System Heterogeneity

Statistical Heterogeneity

Expensive Communication

System Heterogeneity

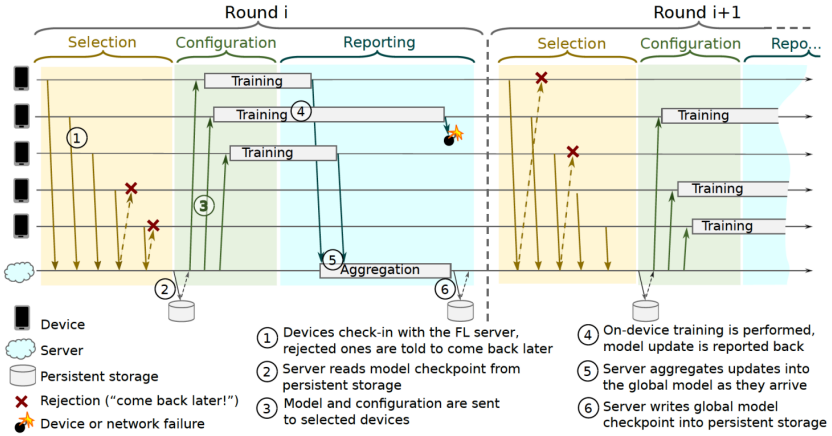
Clients can differ to each other in:

- computational capabilities, hardware architecture & resources
- communication capabilities, reliability & bandwidth

Consequences:

- "stragglers" impede the entire process
- random disconnections
- unavailable clients

System Heterogeneity



Statistical Heterogeneity

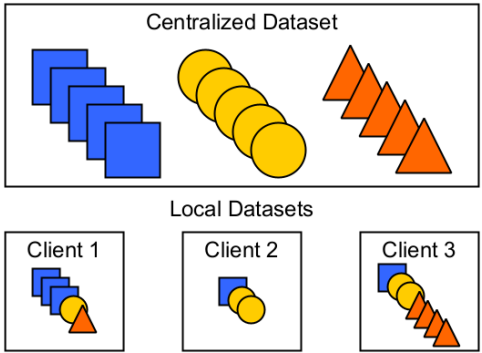
Independent and Identically Distributed (IID) dataset:

- all examples have the same probability distribution
- all examples are mutually independent

In practice, clients have data collection biases, thus non-IID datasets are more common.

Huge ramifications in model convergence and overall training time!

Statistical Heterogeneity



Centralized vs. Local Datasets

Data distribution can be non-IID due to:

- label distribution skew
- concept drift
- quantity skew
- etc.

Expensive Communication

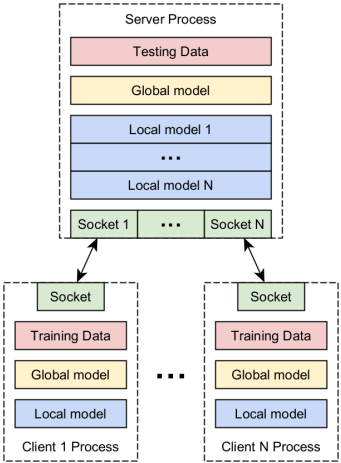
Complete models are shared between the server and the clients -> large messages

edge devices usually have slow and unreliable connections

Communication is a critical bottleneck!

FL architecture & design

Process & Memory Layout



Each entity is a distinct process with its own memory space.

POSIX sockets with TCP protocol.

TCP segments large messages, server have to keep copies of all models.

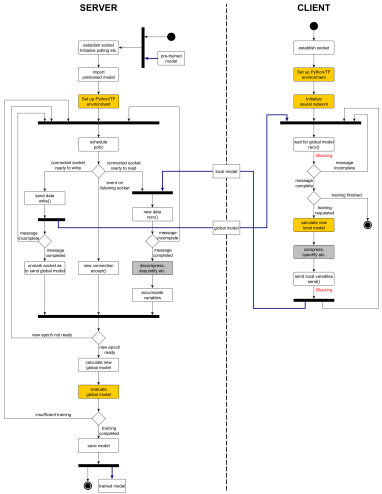
Communication Scheme

Server to client message
flags
GE
global model variables
...

Client to server message
GE
local loss
local accuracy
model variables / deltas
...

Table: The format of the communication between the server and the clients.

Synchronization



Event-driven server paradigm.

Large messages, multiple events are required to send & receive them.

Typical master-slave relationship between the server and the client.

Multiple clients are connected to the server.

Connectivity issues with a client, do not affect the communication with the rest.

Model Library

TensorFlow is incorporated to test the developed FL system.

10 models used, including:

- DNNs
- CNNs
- based on the Inception Module
- RNNs

Most models are using ReLU and Softmax activations. Their sizes range between 60 thousand to 60 million parameters.

Robustness Analysis

Preamble

Goals:

- Prove algorithmic soundness & robustness of developed FL system
- Discover the parameter space where FL is most efficient

Benchmark values:

- Model accuracy
- Dataset repetitions

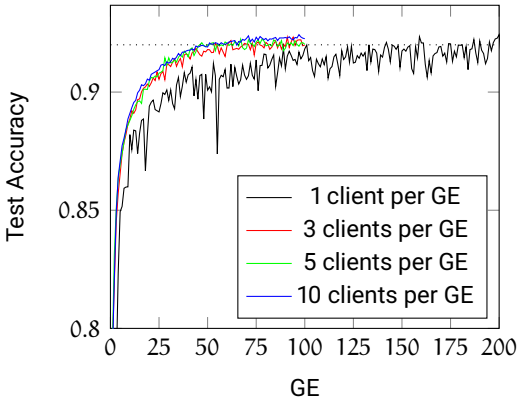
Client Participation

parameters	FedAvg
total clients	10
local epochs	1
steps per epoch	300
batch size	20
initial LR	1e-2
LR decay	0.977

Experiment with different numbers of participating clients per GE.

- 6-layer CNN with ~420.000 variables
- Federated Averaging with IID datasets

Client Participation



client per GE	0.92 @GE
1	159
3	61 (2.6×)
5	46 (3.4×)
10	51 (3.1×)

GEs for target accuracy and relative speedup

General Conclusions

Training with non-IID datasets can be done efficiently in a far narrower parameter space, and requires a lot more training rounds.

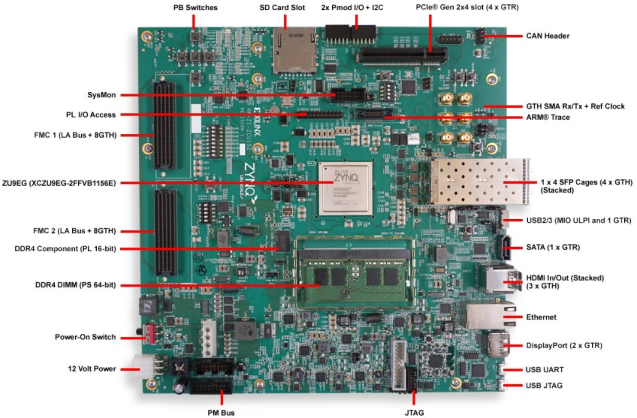
The most important parameter is the batch size. The convergence rate and final accuracy are dictated by it.

LR and its decay have a great impact, but they are depended on other factors.

Consuming the local datasets multiple times per GE can be beneficial or detrimental, depending on other factors.

FPGA Implementation

Xilinx ZCU102 Evaluation Kit



- SoC with ARM CPU
- Multiple peripherals
- Great for prototyping edge applications

Tools Used: Vitis High Level Synthesis (HLS)

- Synthesize "C/C++ like" code to RTL
- Automate large parts of the implementations
- Provide optimized libraries for data types, interfaces, etc.

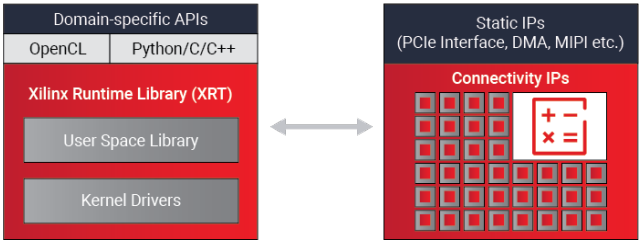
Designer's Responsibilities
Macro Architecture
Design Intent
Constrains

HLS tool automation
FSM Generation
Operation Scheduling
Clock
Register Pipelining
Resource Sharing
Timing
Verification

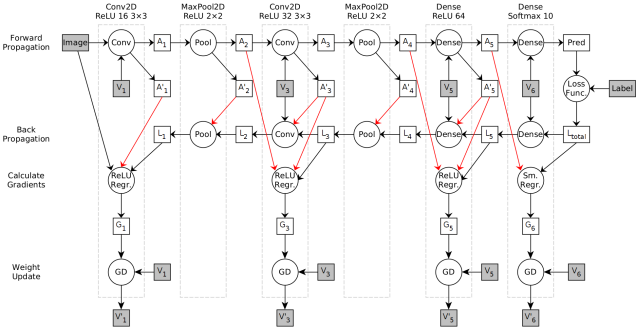
Distribution of work during HLS design.

Xilinx Runtime library (XRT)

- Facilitate communication between PS & PL
- Manage shared memory
- Portable & open-source



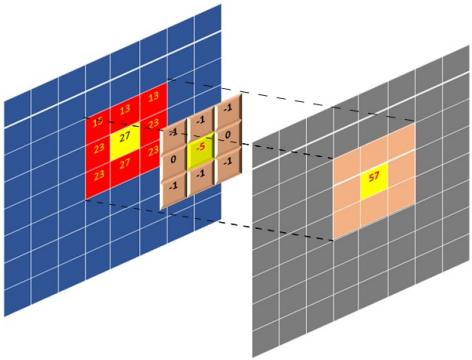
CPU-based C++ CNN implementation



Dataflow diagram of the implemented CNN

- 6 layers & ~105000 variables
- Convolutional, max-pooling & dense layers
- ReLU & Softmax Activations
- Float32 implementation

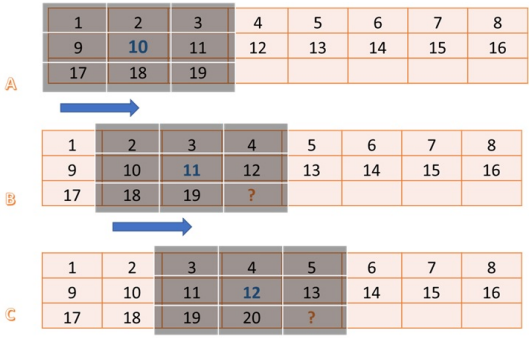
FPGA-based CNN implementation - 2D Convolutional Layers- Access Pattern



Data access patern of 2D convolution

- Input pixels are accessed serially
- Two dimensional filters with non-serial access pattern
- Input pixels have to be reused
- Float32 implementation

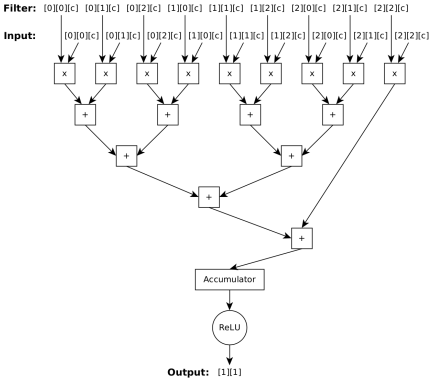
FPGA-based CNN implementation - 2D Convolutional Layers - Line Buffers



Line Buffers with 3×3 windows

- Write input pixels serially
- Access them through 3×3 windows with stride 1
- Padding required
- Expandable for multiple input channels

FPGA-based CNN implementation - 2D Convolutional Layers - Computations



Conv2D: order of calculations

- Multiple input channels operate additively, accumulator required
- Reordered Calculations to fit hardware better
- Floats are not commutative, tiny difference in results

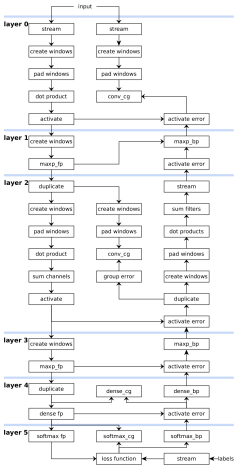
FPGA-based CNN implementation - Dense Layers

Two components: matrix multiplication and an activation function.

Dense Layers require all inputs to produce an output. They operate as barriers!

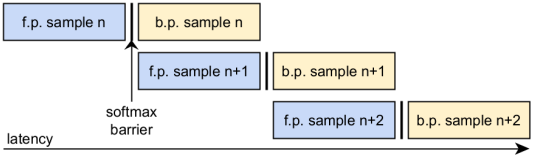
All data before them have been produced. If any of them skips them, must be stored in memory.

FPGA-based CNN implementation - Gradients Calculation Pipeline



- Dataflow pipeline, hardware functions can run simultaneously
- The Softmax layer split the pipeline to two region that can not operate simultaneously for a single image.
- Inputs are accessed twice, special treatment required
- First Layer does not require back-propagation
- Most hardware functions are data transformations

FPGA-based CNN implementation - Batching Inputs



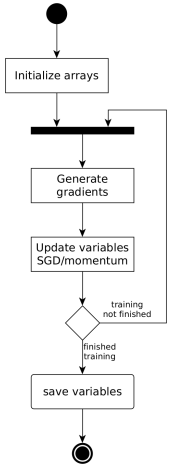
- The effect of the barriers can be removed by batching inputs.
- Vitis HLS trivialize implementation.

Data that skips layers may be produced by the forward propagation of the second image, before they are consumed by the back propagation of the first one!

Each sample adds latency equal to the function with the highest latency.

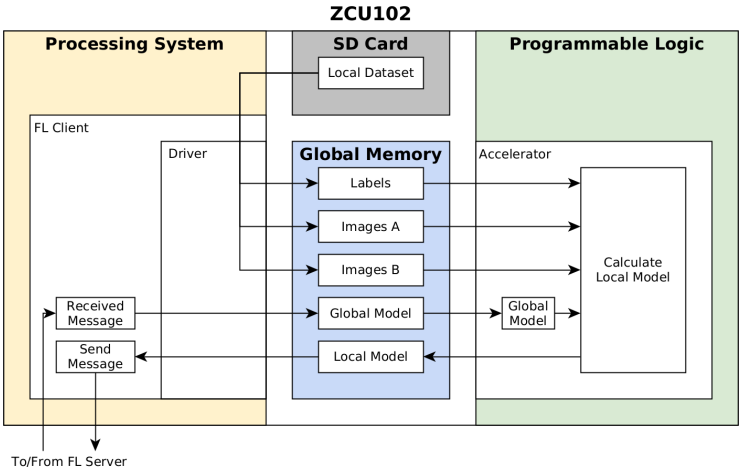
The time between the start of the pipeline and the first time the barrier is reached, is referred as pipeline wind-up latency.

FPGA-based CNN implementation - Top Function



- Data that have a life-cycle over a pipeline repetition, require initialization
- Distinct pipeline to update variables
- The pipelines loop for the number of batches

Embedded Systems Perspective



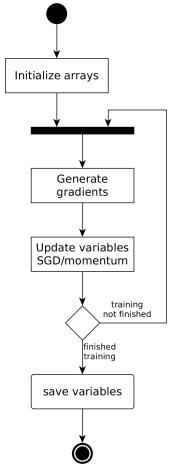
Results

FPGA Implementation Analysis

Comparison with Other Technologies

FL & FPGA Interaction Analysis & Comparison

FPGA Implementation Analysis - Timing Analysis



Latency of training for a single epoch:

$$Accel_{latency} = I + e \cdot \frac{d}{b}[G(b) + U] + S$$

Where:

I = Initialize arrays	G = Generate gradients
U = update variables	S = Save variables
e = local epochs	d = dataset size
b = batch size	

FPGA Implementation Analysis - Timing Analysis

G(b) transforms to: $$G(b) = G_{up} + b \cdot E + G_{down}$$

Where:

G_{up} = latency to wind-up the pipeline
G_{down} = latency to wind-down the pipeline
E = latency added by one example

For a single epoch with the whole dataset (e = 1 & d = 60000):

$$Accel_{latency} = I + S + 60000[E + \frac{G_{up} + G_{down} + U}{b}]$$

If small batches are important, U is an easy target for optimization.

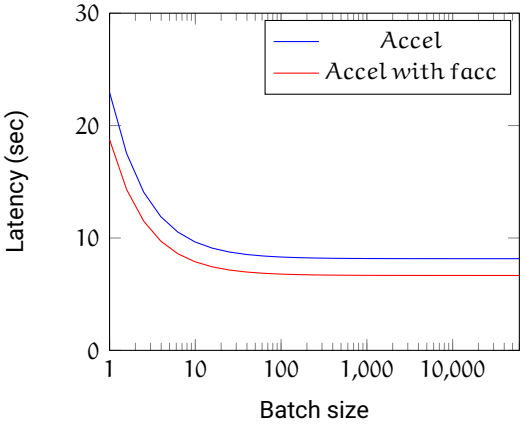
FPGA Implementation Analysis - Timing Analysis

Final latency: $Accel_{latency} = 8.157 + \frac{14.794}{b}(sec)$

Due to a bug in Vitis HLS 2022.1, full accuracy facc, fmacc, etc. are unavailable. Instead fadd is used and the clock is restricted at 245MHz. With accumulators the clock can easily be raised to 300 MHz.

With accumulators: $Accel_{latency} = 6.664 + \frac{12.086}{b}(sec)$

FPGA Implementation Analysis - Timing Analysis



- For batch size ~50 or more, the second part of the equation is insignificant
- Average consumption of 9.356 W

Comparison with other technologies - Specifications of Compared Platforms - i7-9750H

CPU: i7-9750H

- released in 2019
- aimed for mobile platforms (laptops, tablets, etc.).

Core / Threads	6 / 12
Clock Frequency	2.6 - 4.5 GHz
Cache	12 MB Intel Smart Cache
Supplied Memory	16GB DDR4-2666
Max Memory Bandwidth	41.8 GB/s $\times$ 2 channels
Instruction Set Extensions	SSE4.1, SSE4.2, AVX2
Average Power Consumption	45 W

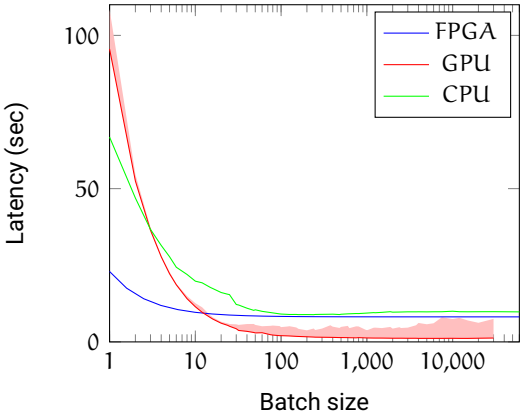
Comparison with other technologies - Specifications of Compared Platforms - GTX1660Ti

GPU: GTX1660Ti

- released in 2019
- aimed for mobile platforms (laptops, tablets, etc.).

CUDA cores	1536
Clock Speed	1500 - 1770 MHz
Memory Configuration	6GB GDDR6, 1500 MHz
Memory Interface	192-bit
Memory Bandwidth	288 GB/s
Single Precision Compute Power	5437.44 GFLOPS
Average Power Consumption	120 W

Comparison with other technologies - Latency



Train a single epoch with the whole dataset (60000 samples).

- FPGA faster for small batches
- overtaken by GPU over large batches
- GPU shows great variance
- CPU consistently slower than the other options

FL & FPGA Interaction Analysis & Comparison - Methodology

Throughput comparison is inadequate!

$$\mathrm{FL}_{latency} = \mathrm{GE}_{num} \times \mathrm{GE}_{latency}$$

As shown in Robustness Analysis, the number of GEs depends on multiple factors.

The following experiments focus on the most important one, the batch size.

Repeated with different LR decay values. Only best results are shown.

FL & FPGA Interaction Analysis & Comparison - Methodology

Only a single FPGA/GPU available. Real-time FL is impossible. Instead:

- Train on CPU to find the number GEs required to reach target accuracy.
- Replace the training latency with that of the desired device.

Communication latency can be approximated:

- $\frac{\text{Message\_size}_{\text{bits}}}{\text{Communication\_speed}_{\text{bps}}}$
- Server communication speed: 1Gbps Up, 1Gbps Down.
- Client communication speed: 10Mbps Up, 1Mbps Down.

FL & FPGA Interaction Analysis & Comparison - IID dataset - Setting

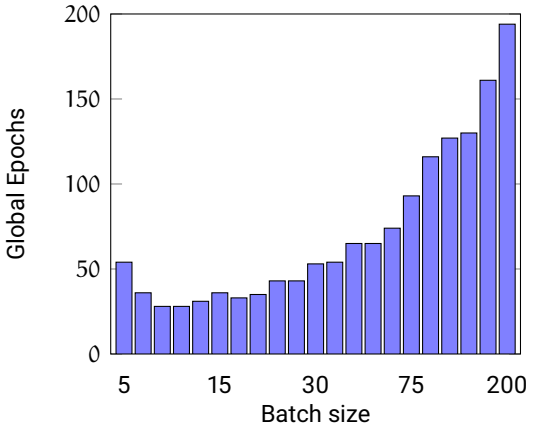
The Fashion-MNIST dataset is split randomly and evenly across 10 clients.
Federated Averaging is used.

parameters	FedAvg
total clients	10
clients per GE	5
local epochs	1
initial LR	1e-2

Parameters of the IID FL experiment.

Target accuracy is set at 91%.

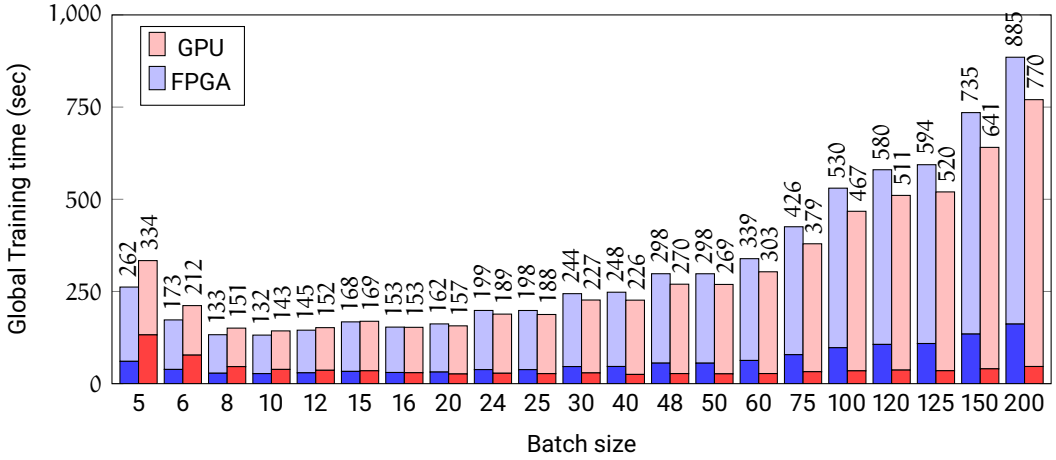
FL & FPGA Interaction Analysis & Comparison - IID dataset - Convergence



GEs to reach 91% accuracy, per batch size.

- Batch sizes 8 & 10 require the least GEs, 28.
- for batch sizes smaller than 5 or larger than 200, the model does not converge or require over 200 GEs.

FL & FPGA Interaction Analysis & Comparison - IID dataset - Total time



FL & FPGA Interaction Analysis & Comparison - Non-IID dataset - Setting

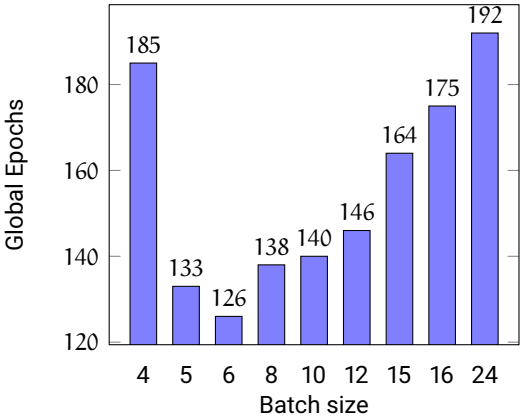
The Fashion-MNIST dataset is split between 5 clients. Each client is the exclusive owner of two labels. Federated Averaging is used.

parameters	FedAvg
total clients	5
clients per GE	5
local epochs	1
initial LR	1e-2

Parameters of the non-IID FL experiment.

Target accuracy is set at 85%.

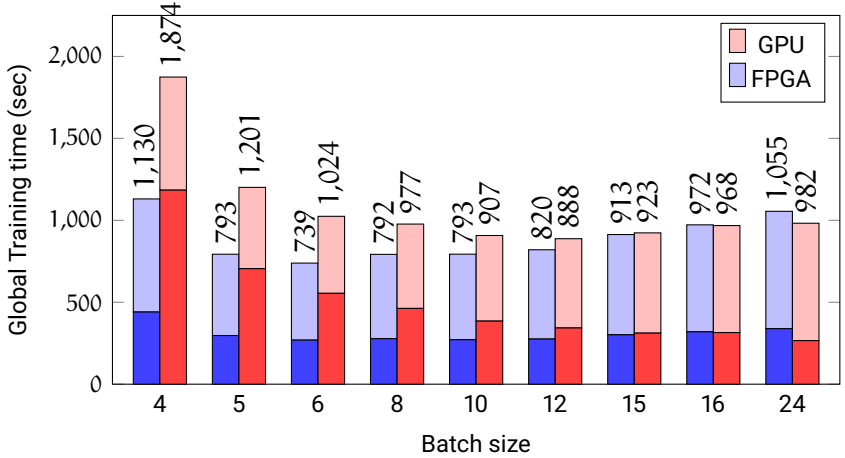
FL & FPGA Interaction Analysis & Comparison - Non-IID dataset - Convergence



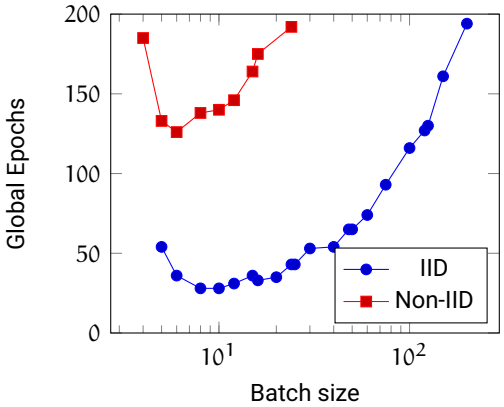
GEs to reach 85% accuracy, per batch size.

- Few batch sizes reach the target accuracy.

FL & FPGA Interaction Analysis & Comparison - Non-IID dataset - Total time



FL FPGA Interaction Analysis Comparison - Summary



Dataset Distribution	Total Time (s) GPU	FPGA
IID	143	132 (1.08×)
non-IID	888	739 (1.2×)

Dataset Distribution	Total Energy (J) GPU	FPGA
IID	4637	255 (18.18×)
non-IID	41.16k	2.57k (16.35×)

Conclusions & Future Work

Conclusions

- In any FL application, the parameter space should be explored to discover its optimum point.
- The re-configurability of FPGAs can be exploited to develop an accelerator that is optimized for that point.
- Time and power wise, FPGA-based implementations of FL clients can be superior to equivalent CPU & GPU based implementations.

Future Work

- **Quantization**: In on-edge FL, to improve the communication-to-computation ratio, it has been proposed to quantize the communication. Furthermore, quantization is also used to speed-up training on ANNs. FPGAs are a fitting platform to implement that.
- **Scale**: This work experiments with 2 to 20 clients with a few thousand samples each. In other settings, the optimal point of the parameter space may be different, thus making FPGAs less or more fitting.
- **Models**: Larger models could be locally trained more before overfitting, thus having a better communication-to-computation ratio. It should be explored if FPGAs are less or more fitting to such a setting.

Thank You!

Any Questions?

Appendix

Convolutional Neural Network

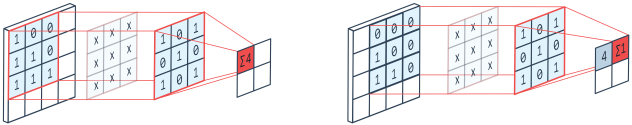


Figure: 2D convolution

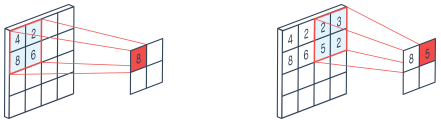


Figure: 2D max pooling

FL Algorithms

Distributed SGD

- initially intended for training in datacenters
- workers compute new models from distinct mini-batches in parallel
- synchronization service aggregates workers' models
- can be generalized for FL, referred as Federated SGD

FederatedAveraging

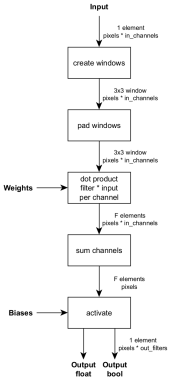
- Cornerstone of FL, most FL algorithms are its derivatives
- clients compute new models from distinct datasets in parallel
- server aggregates clients' models
- each GE, local datasets are split in batches and exhausted completely

Thesis Approach

Many components, requiring different development tools. Split the development in distinct steps.

1. develop the synchronization & communication parts of the FL system
2. implement local training using TensorFlow
3. explore the behaviour of the complete FL system
4. create an optimized FPGA-based CNN training accelerator
5. merge the FPGA-based accelerator with the FL system

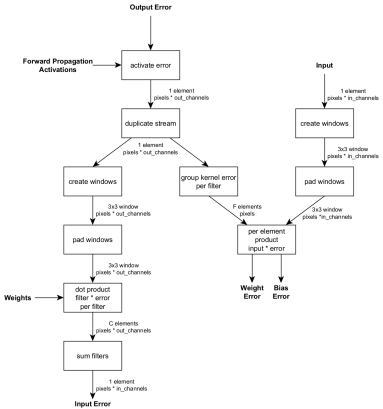
FPGA-based CNN implementation - 2D Convolutional Layers - Forward Propagation



Conv2D: Forward Propagation

- Inputs and outputs are streamed serially to maintain compatibility with other layers
- Internal streams are more flexible
- Filters are calculated in parallel

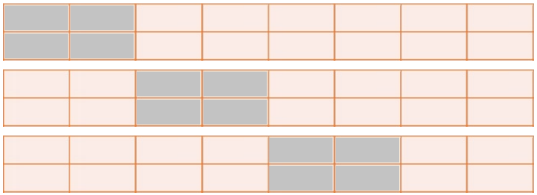
FPGA-based CNN implementation - 2D Convolutional Layers - Back Propagation & Gradient Calculation



Conv2D: Back Propagation

- ReLU activate the error
- Compared to forward propagation, filters and input channels exchange places

FPGA-based CNN implementation - 2D Max Pooling Layers - Line Buffers



Line Buffers with 2×2 windows and stride 2

- Write input pixels serially
- Access them through sliding 2×2 windows with stride 2
- Padding is not required
- Expandable for multiple input channels
- After that, only a comparison is required

FPGA-based CNN implementation - Data Movement & Storage

The majority of data movements are implemented with FIFOs. Implementation provided by the Vitis HLS "hls_stream.h" library.

FIFOs that skip layers require a depth of 1.5-2$\times$ the data that are produced by a single image.

Weights, biases, gradients and momenta are stored in BRAM. The models in Global Memory are accessed only at the start & finish of the accelerator's operation.

Images and labels are always read from Global Memory. Images are stored twice so they can be easily accessed as required by the gradients calculation pipeline.

FPGA Implementation Analysis - Resource Utilization Analysis

Float32 accuracy, back-propagation and SGD with momentum increase utilization considerably, even for such a small CNN.

Resource	Utilization	Available	Utilization %
LUT	161274	274080	58.84
LUTRAM	14270	144000	9.91
FF	260050	548160	47.44
BRAM	573	912	62.83
DSP	854	2520	33.89

Post place & route resource utilization.