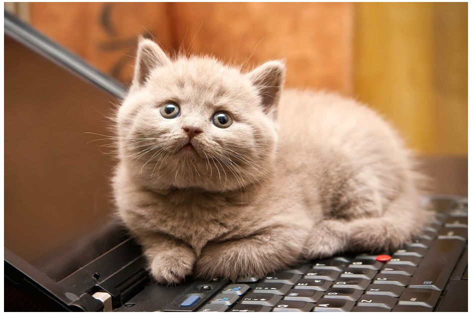
TABLA: A Framework for Accelerating Statistical Machine Learning

Presenters:
MeiXing Dong, Lajanugen Logeswaran

Intro

- Machine learning algorithms widely used, computationally intensive
- FPGAs get performance gains w/ flexibility
- Development for FPGAs expensive and long
- Automatically generate accelerators (TABLA)



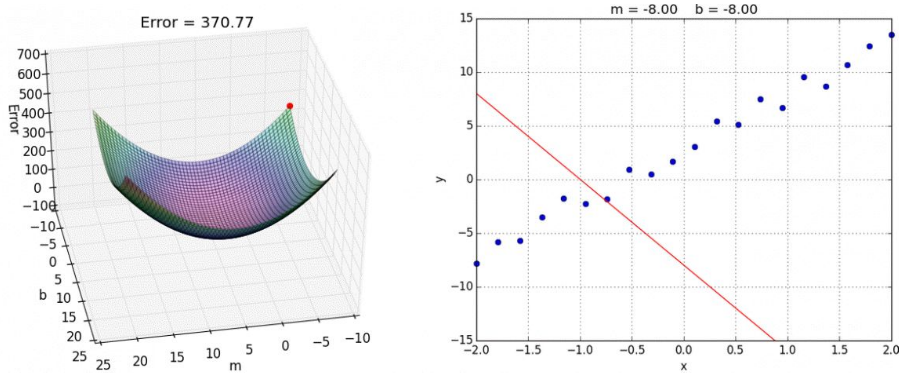
ISTOCK/ANNA LURYE

CAT

* Unless otherwise noted, all figures from Mahajan, Divya, et al. "Tabla: A unified template-based framework for accelerating statistical machine learning." High Performance Computer Architecture (HPCA), 2016 IEEE International Symposium on. IEEE, 2016.

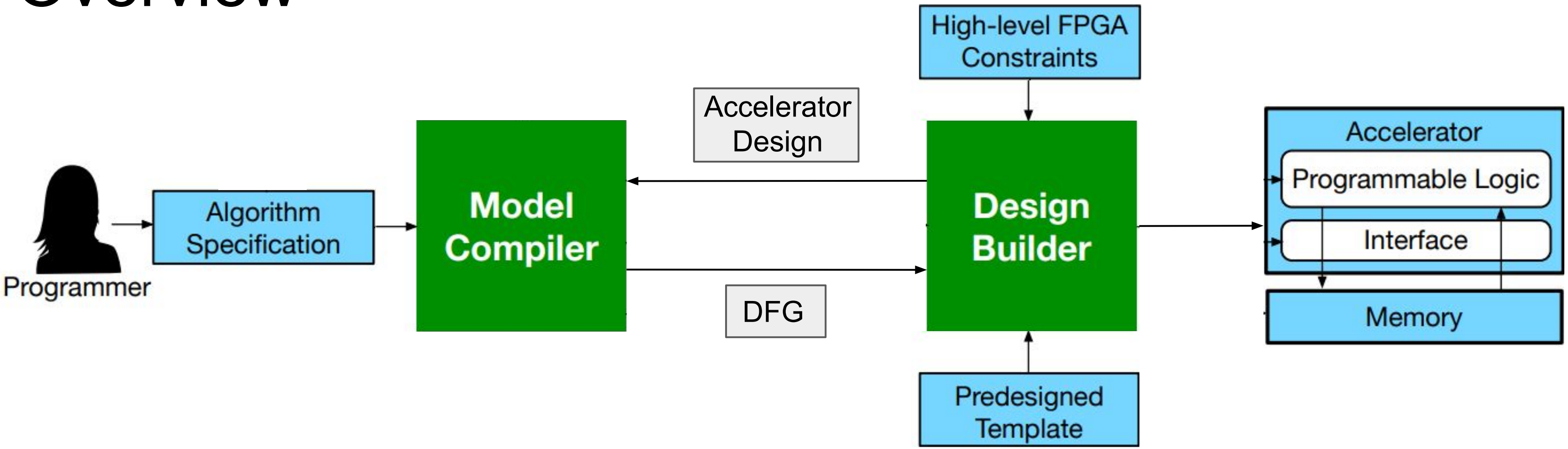
Stochastic Gradient Descent

- Machine learning uses objective (cost) functions
- Ex. linear regression
  - objective: $\sum_i 1/2(w^T x_i - y_i)^2 + \lambda||w||$
  - gradient: $\sum_i (w^T x_i - y_i)x_i + \lambda||w||$
- Want to find lowest value possible w/ gradient descent
- Can approximate batch update

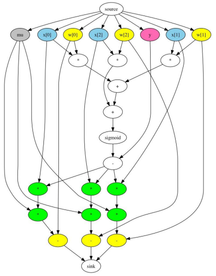


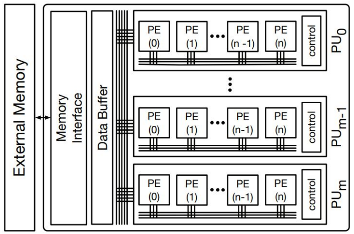
Src: *https://alykhantejani.github.io/a-brief-introduction-to-gradient-descent/*

Overview




```
model_input  x[m]; //model input features
model_output y'[n]; //model outputs
model        w[n][m]; //model parameters
gradient     g[n][m]; //gradient

iterator i[0:m]; //iterator for group operations
iterator j[0:n]; //iterator for group operations

//m parallel multiplications followed by
//an addition tree; repeat n times in parallel
s[j] = sum[i](x[i] * w[j][i]);

y[j] = sigmoid(s[j]); //n parallel sigmoid operations
e[j] = y[j] - y'[j]; //n parallel subtractions
g[j][i] = x[i] * e[j]; //n*m parallel multiplications
rg[j][i] = λ * w[i][j]; //n*m parallel multiplications
g[j][i] = g[j][i] + rg[j][i]; //n*m parallel additions
```





Src: http://act-lab.org/artifacts/tabla/

Programming Interface

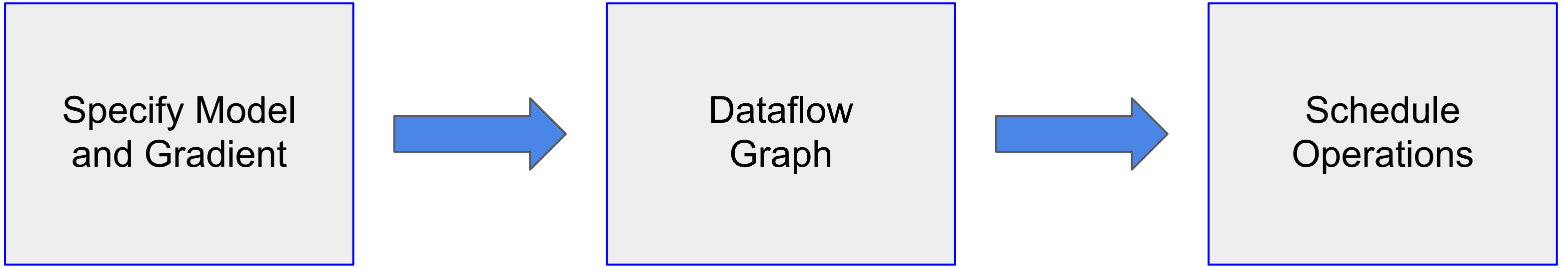
Type	Classification	Language Keywords
Data	Model inputs	model_input
	Model outputs	model_output
	Model Parameters	model
	Gradient of objective function	gradient
	Iterator variable	iterator
Operation	Basic	+,-, <, >, *
	Group	pi, sum, norm
	Non Linear	gaussian, sigmoid, sigmoid_symmteric, log

- Language
  - Close to mathematical expressions
  - Language constructs commonly used in ML algorithms

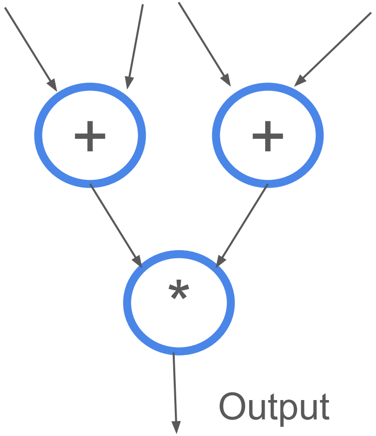
```
model_input   x[m]; //model input features
model_output  y'[n]; //model outputs
model         w[n][m]; //model parameters
gradient      g[n][m]; //gradient

iterator i[0:m]; //iterator for group operations
iterator j[0:n]; //iterator for group operations

//m parallel multiplications followed by
//an addition tree; repeat n times in parallel
s[j] = sum[i](x[i] * w[j][i]);

y[j] = sigmoid(s[j]); //n parallel sigmoid operations
e[j] = y[j] - y'[j]; //n parallel subtractions
g[j][i] = x[i] * e[j]; //n*m parallel multiplications
rg[j][i] = λ * w[i][j]; //n*m parallel multiplications
g[j][i] = g[j][i] + rg[j][i]; //n*m parallel additions
```

- Why not MATLAB/R ?
  - Identifying parallelizable code
  - Conversion to hardware design

Model Compiler



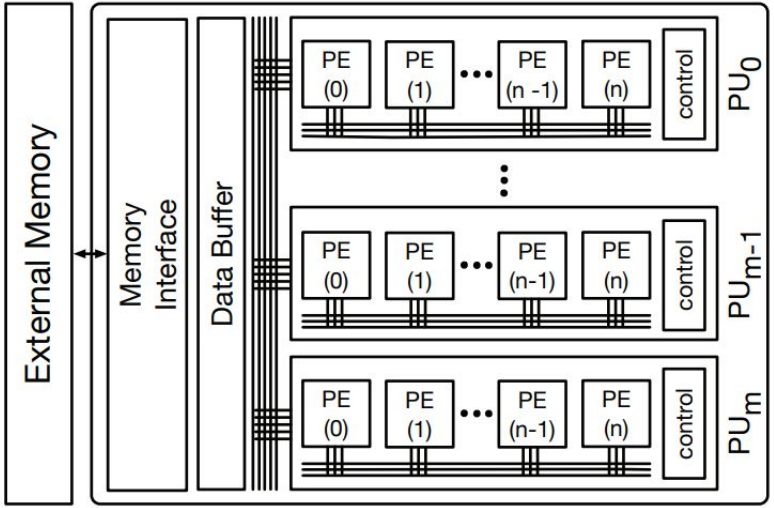
- Model parameters and gradient are both arrays of values
- Gradient function specified using math
- Ex.
  - g[j][i] = u*g[j][i]
  - g[j][i] = w[j][i] - g[j][i]



- Minimum-Latency Resource Constrained Scheduling
- Priority placed on highest distance from sink
- Predecessors scheduled
- Resources available

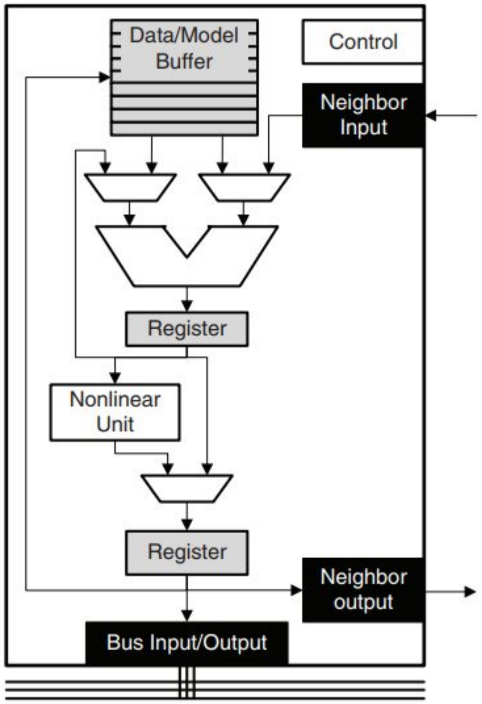
Accelerator Design: Design builder

- Generates Verilog of accelerator from
  - DFG, algorithm schedule, FPGA spec
- Clustered hierarchical architecture
- Determines
  - Number of PEs
  - Number of PEs per PU
- Generate
  - Control units and buses
  - Memory interface unit and access schedule



Accelerator Design: Processing engine

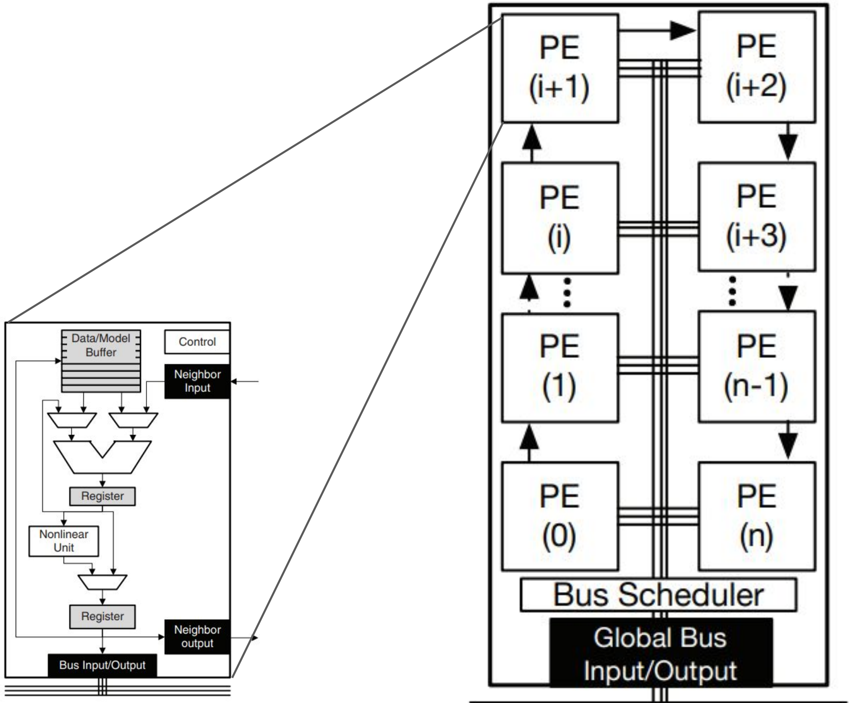
- Basic block
- Fixed components
  - ALU
  - Data/Model buffer
  - Registers
  - Busing logic
- Customizable components
  - Control unit
  - Nonlinear unit
  - Neighbor input/output communication



(b) Processing Engine (PE)

Accelerator Design: Processing unit

- Group of PEs
  - Modular design
  - Data traffic locality within PU
- Scale up as necessary
- Static communication schedule
  - Global bus
  - Memory access



(b) Processing Engine (PE)

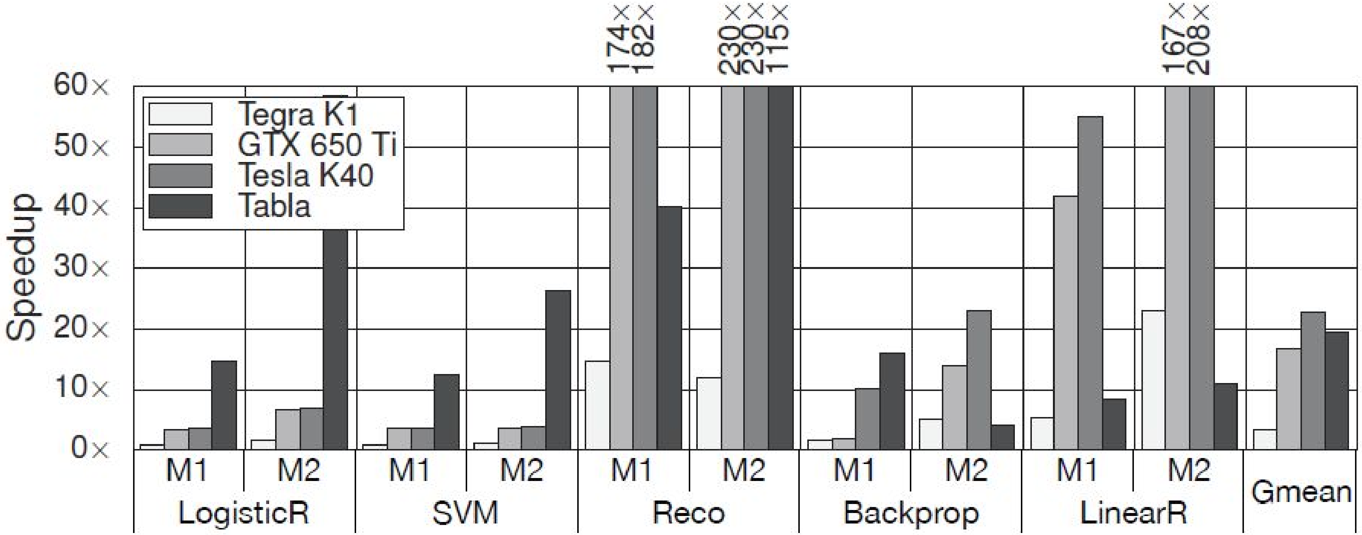
(a) Processing Unit (PU)

Evaluation

Setup

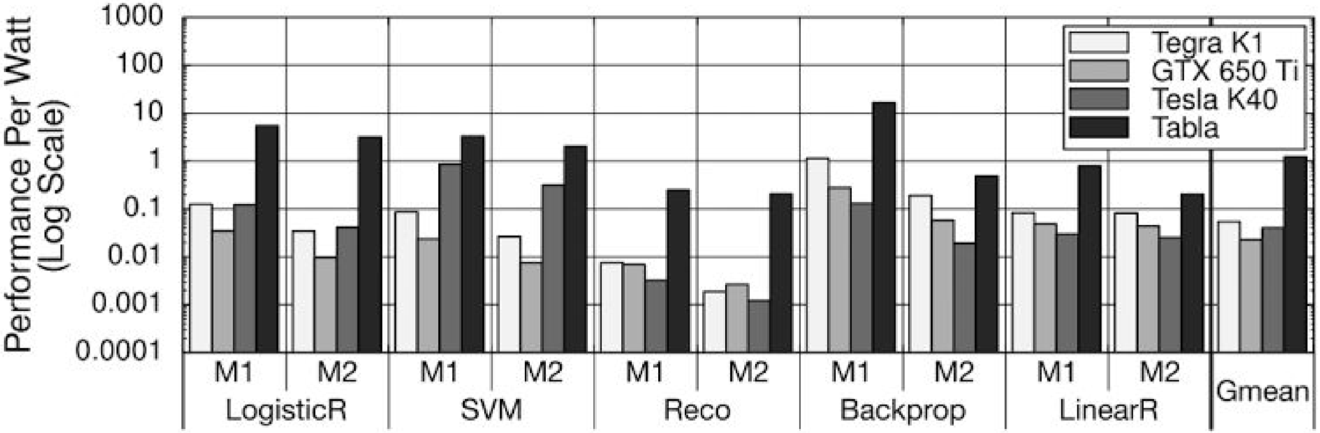
- Implement TABLA using off-the-shelf FPGA platform (Xilinx Zynq ZC702)
- Compare with CPUs and GPUs
- 5 popular ML algorithms
  - Logistic Regression
  - Support Vector Machines
  - Recommender Systems
  - Backpropagation
  - Linear Regression
- Measurements
  - Execution time
  - Power

Performance Comparison



(b) Speedup of GPUs and TABLA design in comparison ARM A15.

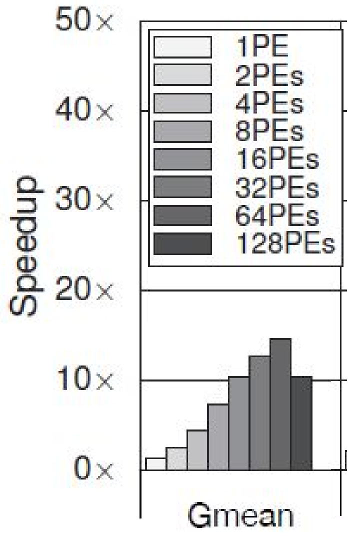
Power Usage



(b) Performance-per-Watt comparison between Tegra, GTX 650 , Tesla and TABLA

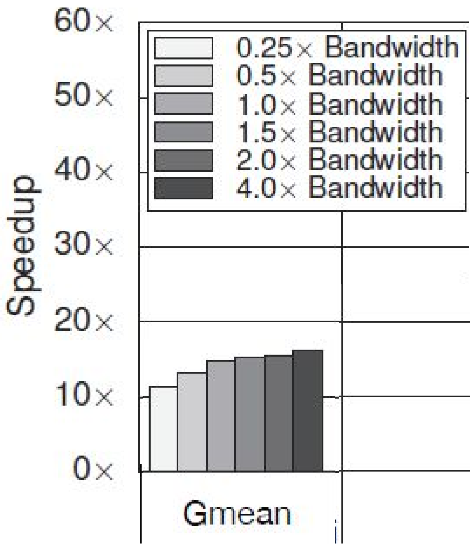
Design Space Exploration

- Number of PEs vs PUs
  - Configuration that provides highest frequency
    - 8 PEs per PU
- Number of PEs
  - Initially linear increase
  - Poor performance after a certain point
- Too many PEs
  - Wider global bus - Reduced frequency



Design Space Exploration

- Bandwidth sensitivity
  - Increase bandwidth between external memory and accelerator
  - Limited improvement
    - Computation dominates execution time
    - Frequently accessed data are kept in PE's local buffers



Conclusion

- Machine learning algorithms popular but compute-intensive
- FPGAs are appealing for accelerating performance
- FPGA design long and expensive
- Automatically generate accelerators for learning algorithms using template-based framework (TABLA)

Discussion Points

- Is this more useful than accelerators specialized for gradient descent?
- Is this solution practical? (Cost, Scalability, Performance)
- Is this idea generalizable to problems other than gradient descent?