Closing the Co-Design Gap with *Bespoke* Heterogeneous Acceleration

Thelonious Cooper
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA, USA
theloni@mit.edu

***Abstract*—In both edge and datacenter ML applications, power efficiency and speed are paramount. To address the growing need for efficiency in deep neural computation systems, much work has gone into co-design of hardware and software for ML operations. Field-Programmable-Gate-Arrays are a natural target for co-design due to their balance of flexibility and efficiency. *Bespoke* is a system for synthesizing FPGA stream processors from ONNX neural network specifications. This presents users with the ability to focus on designing new network architectures with the peace of mind that the target hardware will be well-suited to their tasks. *Bespoke* dramatically shortens the co-design development cycle, enabling for rapid prototyping, verification, and deployment. Edge and datacenter SoCs operating alongside a *Bespoke* FPGA core can be dynamically reconfigured to suit any ML stream-processing task. The model could even be modified or tuned on the edge. Suitable applications include ultra-low latency multimodal data fusion, control, forecasting, and decision making.**

***Index Terms*—Field Programmable Gate Array, Stream Processing, Machine Learning Hardware.**

I. Introduction

IN the landscape of high-performance computing, Field-Programmable-Gate-Arrays hold a unique position.

Edge machine learning applications have been gaining significant attention lately due to their promise of complex operations without the need for slow server-side processing. Many *IoT* devices are glorified relay clients, with all significant computation occurring offsite. While in transit the data they are relaying is subject to latency, corruption, unwanted inspection[Dub+21], spoofing, and all manner of other challenges. Performing computations on-device remedies all of these problems, but presents its own challenges.

A key concern with edge machine learning is the computation and power requirements of ML routines. Common general purpose microcontrollers are not designed to handle the intensive linear algebraic operations common in machine learning in a time or power efficient manner. One approach to address this concern is packaging the micro controller with a TPU[Zha+23] or other generic matrix operation accelerator. These accelerators often support many operations not needed for a particular use case, compromising footprint and power consumption for generality. Instead, *Bespoke* purpose-built accelerators only run a particular network or intensive subroutine. It is uncommon for edge ML applications to be running many different models requiring a diverse set of supported operations. As such, *Bespoke* offers a superior power and time efficient solution for edge computation.

II. Related Work

Most focus of research on edge ML processors is concerned with generic tensor accelerators that can support many operations. These architectures are limited by their need for shared memory access which is subject to data leakage concerns and bandwidth limitations. Some work of interest is in the quantization of large networks to few-bit formats. Many so called "n-bit quantization" implementations in popular neural network frameworks like Tensorflow[Mar+15], PyTorch[Pas+19], and ONNX[dev21] fail to create a fully low-bit implementation. Instead they find zero points and scales for 32-bit floating point numbers which they go between when passing between integer matrix multiplications. There has been some work to generate entirely 8 bit integer networks that have the potential to abuse things like over and underflow [Wan+20]. But these networks remain on generalist hardware such as gpu and cpu architectures.

There have been other attempts at High-Level-Synthesis of ONNX models. Noteably there are projects out of CERN [Dua+18] and Xilinx [Umu+17]. Unfortunately, both of these rely on existing HLS toolchains for C++ implementations of relevant operations, which tend to have poor performance as compared to hand-crafted RTL.

III. Parametric Dataflow Module Design

As a generic architecture all modules will follow a similar format. The architecture is a typical dataflow fused-layer implementation a la [Gil+24]. Each module will be connected to its previous and next nodes via BlockRAM (BRAM)-based First-In-First-Out (FIFO)s Queues and will be parameterized by the number of concurrent multiplication or logic registers (*WorkingRegs*) they have. Each module will grab as many bytes from the FIFO as they have working registers at a time. Working registers will each be allowed a single DSP48 slice so DSP48 and BRAM usage will be known at link time as opposed to synthesis time. The FIFOs used have different read and write sizes, allowing modules with different bandwidths to communicate seamlessly via a ready-valid handshake. All weights and inputs are quantized to Q4.8 (4 twos-complement integer bits, 8 fraction bits) fixed-point format.

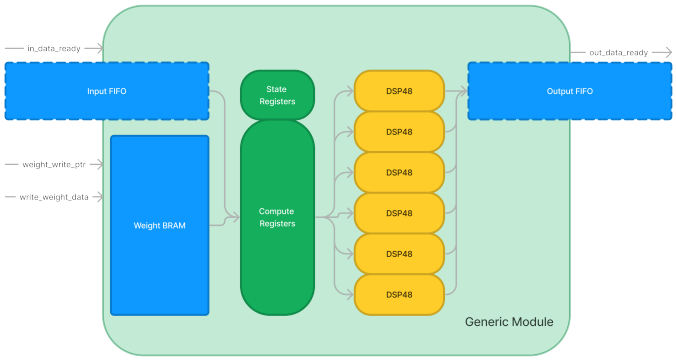

Fig. 1. Generic module architecture which is specified by each ML component

A. Gemm

Perhaps the most important operation of any ML system is the Generalized Matrix-Multiply-Accumulate. The Gemm module is parameterized in terms of *WorkingRegs*: the number of parallel multipliers allocated to the module, *InVecLength*: the dimension of the input vector, *OutVecLength*: the dimension of the output vector, and *WeightFile*: a string referencing the .mem file for the weight and bias BRAM. I have implemented the matrix as a contiguous row-major BRAM instance which is defined at link time by a weight file parameter. The vector is received in chunks of *WorkingRegs* bytes from the input FIFO. The dot product is implemented with parallel multiplications fed into a pipelined adder tree with double the fixed-point bit width at its accumulation output. The pipelined adder tree first stores is constructed via a recursive generate block that pushes its input into registers and then combinationally adds them for its output. Pipelining the inputs as opposed to the outputs is necessary to prevent the synthesizer from inferring multiply-accumulate blocks which have higher latency than pure multiplications. The chunks are requested with a single logic that tells the FIFO to increment the read pointer and provide the information. A separate signal tells the FIFO to reset its read pointer back to the start of the vector block. Currently the module performs a single parallel block multiply-accumulate operation per cycle, and thus produces only one output vector element at a time. In a future implementation this could be further parallelized by computing rows independently instead of sequentially.

B. Vector Multiply-Accumulate

Vector-Multiply-Accumulate is another common ML operation, commonly implemented as BatchNorm. It is implemented with a Block RAM (BRAM) instance for its weights and biases and a FIFO for the input vector. It is implemented as a Finite-State-Machine performing *WorkingRegs* parallel multiply-accumulate per cycle.

C. Concatenation

Unfortunately I was not able to complete the implementation and validation of a concatenation module, rendering the compiler unable to generate the model I had originally envisioned.

D. LeakyReLU Activation

The Rectified Linear Unit or ReLU is a common activation function which simply zeros out all elements below zero. LeakyReLU improves training performance by maintaining gradients below zero with a small factor. It allows for strong nonlinearity and masking. It is implemented as a FSM, performing conditional evaluation of *WorkingRegs* bytes at a time.

IV. Dynamic Model Hardware Generation

The second aspect of the project is model parsing and SystemVerilog generation based on the model specification. This script will have several components.

A. Model Specification Parsing

To begin, parses the Open Neural Network Exchange (ONNX) data file, a model specification designed by Microsoft for edge applications. It then identifiers the variables and the flow of information into an intermediate graph representation. The system iterates through the nodes of the ONNX graph and creates modules with FIFOs in between them serving as skid-buffers.

B. Computation Graph Optimizations and Pipeline-Aware Scheduling

Because the cycles-needed to complete an operation for each module are known at compile-time, discrete optimization techniques can be applied to maximize throughput and utilization.

C. BRAM Allocation and .mem Generation

In this stage weights from the 32 floating-point matrix initializer fields of the ONNX model will be converted Q4.8 fixed point format and dumped to .mem format. The compiler will error if it runs out of BRAM space during the allocation process.

D. Register and DSP Allocation

In this stage the compiler starts by splitting the available DSP blocks among the matrix multiplication modules. Each matrix multiplier gets the highest factor of the input vector that is less than the total number DSPs divided by the number of matrix multipliers. This ensures that the DSP utilization will always be within the constraints. Then, to reduce idleness, the compiler calculates how many cycles it will take to complete the necessary matrix multiplications and will allocate working registers to the remaining modules as to match the number of cycles in the multipliers, thus minimizing LUT usage.

E. Code Generation

Finally, the modules will be assembled into SystemVerilog and linked up according to the graph. This SystemVerilog will then be synthesized by Vivado and flashed to the FPGA. Modules were developed as classes which inherit from a common *MLModule* abstract base class that describes a node in a DAG with variables it owns and variables it references.

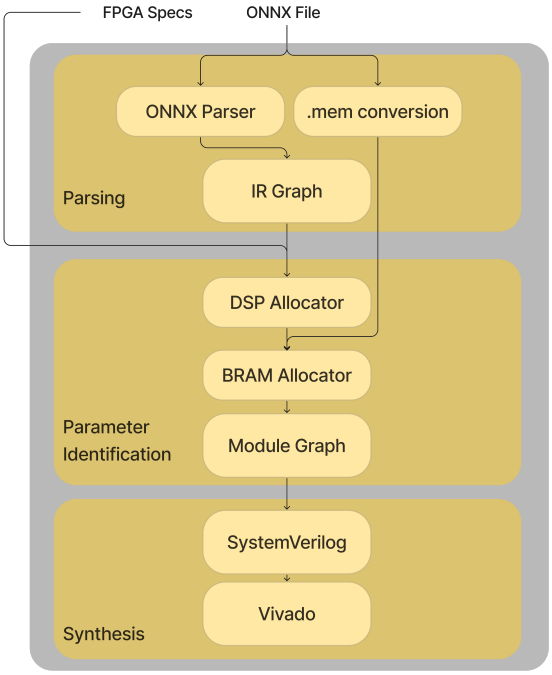

Fig. 2. ONNX-to-FPGA compiler architecture diagram

V. End-to-end Testing and Verification with Respect to ONNX Model

A. Verification Methodology

Bespoke's models have been verified by generating ONNX models with random weight and bias matrices that are then compiled and simulated via CocoTB on random inputs. In all tests of the 12bit fixed-point system against the ONNX CPU Reference Evaluator, all modules produced outputs within a $\epsilon = 0.05$ tolerance. A CocoTB test library complete with monitors, scoreboards, drivers, and abstract test classes was written to facilitate easy testing.

VI. Performance and Utilization Scaling Analysis

Analysis was performed on 3-layer fully connected square networks optimized according to the Xilinx Spartan-7 specification. It is of note that all of the utilization categories are

Width	Params	Slice LUT %	Slice Reg %	BRAM %	DSPs
5	90	2.32	1.19	0	18
10	330	5.31	1.89	6	30
20	1260	6.98	3.56	10	60
40	4920	12.69	8.18	18	120
50	7650	34.86	10.34	12	75
75	17100	50.46	13.78	14	75
80	19440	38.37	16.16	20	120
100	30300	61.05	18.49	14	75

TABLE I
% Utilization of various resources on the Spartan-7

significantly sublinear with respect to number of parameters.

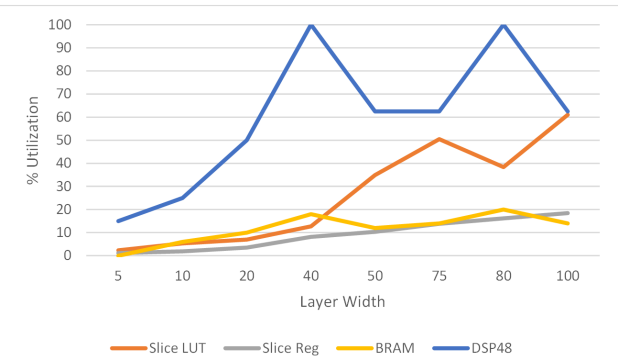

Fig. 3. Utilization of FPGA resources vs Layer Width

The LUT usage seems to be the limiting factor. After performing hierarchical utilization report analysis, I identified the LUT usage culprit as my register-based FIFO. I implemented the FIFO in registers because having variable read and write widths allows for modules that have different operating widths to interface to each other seamlessly and without having to keep track of their own indexing.

VII. Hardware-Aware ML Model Development

A. Case Study: Principled Design of a Novel Neural Audio Processing Scheme

1) Know your Data: An important aspect of model development is understanding the salient features of your data. In the case of audio, I took ques from the MP3 standard which employs a 32-band Cosine-Modulated Psuedo-Quadrature Mirror Filter Bank to decimate the signal. I instead use an 8-band PQMF Bank as a preprocessor as I found it to be sufficient.

2) Know your primitives: The success of the Spectrogram U-Net[Sto+19] indicates that audio is well represented by the relationships between frequencies over time. In order to achieve this without a Short-Time-Fourier-Transform (STFT), I used a reservoir computer with a spectral radius of $\rho = 0.1$ and a clipping nonlinearity to prevent state explosion while using only one comparison per cycle.

3) Have Clear Performance Targets: I decided at the start of the project that I wanted 80% accuracy on validation of 11 classes of Free-Spoken Digit classification (0-9, Noise)

The architecture I landed on is displayed in figure 4. It takes in an 8band decimated audio signal and a feature state that is initalized to zero at the start of inference. There are 4 signal paths which serve different functions. The first is a skip connection from the inputs to the penultimate linear-relu classifier. The second is a batchnormalized version of the inputs, which effectively whitens the signal. The 3rd is the nonlinear feed-forward path which is implemented as a 2-layer Multilayer-Perceptron(MLP). The last path is that through the Reservoir computer, which maintains a past state and loops back on itself during inference, connecting x.3 to fin.

The model was quite difficult to optimize. Because it only has 1100 parameters, small changes can dramatically impact the performance. For this reason I used a hybrid Bayesian

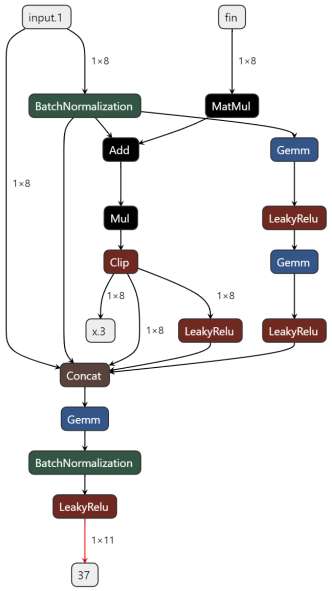

Fig. 4. Architecture of novel recurrent unit

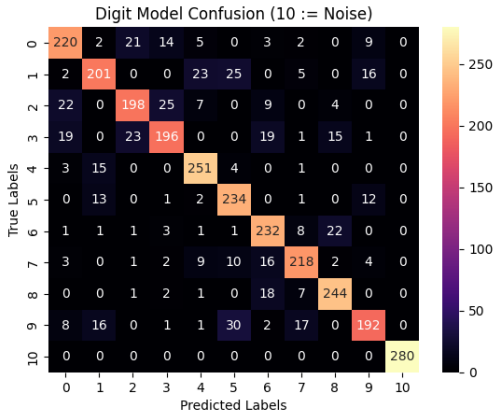

Fig. 5. Confusion matrix for 1193 parameter model; Comparable results in literature use 30,000 parameters

optimization approach known as an Ensemble Kalman Filter to train the network.

B. Model Results

I am quite proud that I was able to achieve 80% validation accuracy on floating point and 73% validation accuracy in an INT8 ONNX quantized version. The class confusion matrix is shown in figure 5

C. Limitations

Unfortunately I was not able to get the compiler to work for the recurrent network. The loopy ONNX computational graph traversal proved quite difficult to implement. If given more time I would improve the compiler so that I could simulate and deploy the developed model.

VIII. Deployment on Heterogeneous SoCs: The Benefits of Co-Existence

Although provocative statements on the future of AI hardware are exciting, the most likely future has no monolithic paradigm. Many SoCs are composed of many chiplets which are interconnected [Li+20]. FPGAs hold an exciting place in these heterogeneous architectures because they are the most flexible components. For maximum performance, it will be most efficient to allow every component to do what its good at. With a set of ASIC accelerators and processors an FPGA can pick up the slack of the most expensive or unsuitable operations on the other cores.

IX. Future Optimizations and Scaling

The current bottleneck in scaling the implementation is the use of BRAM in FIFOs between components. This can be remedied by expanding the scope of the project to include a custom BRAM cache with DRAM parameters. A key advantange of the *Bespoke* approach is that the memory access pattern is known exactly at compile time. This allows for the deployment of custom cache rules that could ensure near-zero miss rates, providing potentially dramatic speedups compared to random-access or LRU schemes.

X. Near-Term Markets

Bespoke opens the door to efficient realizations of all manner of realtime data analysis and decision making from fused multimodal information. Realizing connections between multimodal time series data is a challenging task, but advancements using ML techniques such as [TZ22] have proven effective. Fast decision making from multimodal information is critical in fields such as interventional medicine, defense, and finance. Machine learning models such as 1d convolutional neural networks have proven very effective for anomaly detection [TT23] and classification [Kir+19]. These models can be effectively realized in the Bespoke architecture.

As for nonlinear signal processing, I have talked to a startup spun off from the Princeton Intelligent Robotic Motion Lab (IRoM) that is developing MEMs sensors with very nonlinear voltage responses to the target variables, who use simple ML models to perform nonlinear regression. They are interested in using this technology to move the burden of this computation from the microcontroller to the sensorboard itself allow for modular use. They develop these sensors for realtime wind sensing and gust rejection for advanced quadcopter autopilots as discussed further in [Sim+23].

On the topic of control, Bespoke is of interest for designing controllers and observers for complex autopilot systems. As certain classes of dynamical systems can be exactly represented by neural networks [TLR21], monte carlo methods for control can be constructed with Bespoke generating many forward passes for a model predictive control scheme [YR23].

XI. Long-Shot Markets

In the long term, *Bespoke* systems could be deployed in the cloud as inference endpoints or in networked settings for onsite mid-frequency trading.

XII. Conclusion

Although ML accelerators are far from new, there don't exist low-cost methods for creating cheap stream processors for any particular ML application. Why pay for generality when its not needed? Bespoke delivers the greatest value to those interested in achieving efficient ML computation without having to invest in expensive and complex microprocessors or SoCs with embedded GPUs. Something like an Arduino Micro could theoretically be used as the brain of even a complex system if its role is simply to ingest data from sensors or the web and feed it to a stream processor for any intensive computation.

Acknowledgment

The author would like to thank Joe Steinmeyer of the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology for his advice on direction of the project.

References

[dev21] ONNX Runtime developers. *ONNX Runtime*. https://onnxruntime.ai/. Version: x.y.z. 2021.

[Dua+18] Javier Duarte et al. "Fast inference of deep neural networks in FPGAs for particle physics". In: *JINST* 13.07 (2018), P07027. DOI: 10.1088/1748-0221/13/07/P07027. arXiv: 1804.06913 [physics.ins-det].

[Dub+21] Anuj Dubey et al. "Guarding Machine Learning Hardware Against Physical Side-Channel Attacks". In: *arXiv:2109.00187* (2021). URL: https://arxiv.org/abs/2109.00187.

[Gil+24] Michael Gilbert et al. "LoopTree: Exploring the Fused-layer Dataflow Accelerator Design Space". In: *IEEE Transactions on Circuits and Systems for Artificial Intelligence* (2024).

[Kir+19] Serkan Kiranyaz et al. "1D Convolutional Neural Networks and Applications: A Survey". In: *arXiv preprint arXiv:1905.03554* (2019).

[Li+20] Tao Li et al. "Chiplet Heterogeneous Integration Technology—Status and Challenges". In: *Electronics* 9.4 (2020). ISSN: 2079-9292. DOI: 10.3390/electronics9040670. URL: https://www.mdpi.com/2079-9292/9/4/670.

[Mar+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Sim+23] Nathaniel Simon et al. "FlowDrone: Wind Estimation and Gust Rejection on UAVs Using Fast-Response Hot-Wire Flow Sensors". In: *arXiv preprint arXiv:2210.05857* (2023). URL: https://arxiv.org/abs/2210.05857.

[Sto+19] Daniel Stoller et al. *Seq-U-Net: A One-Dimensional Causal U-Net for Efficient Sequence Modelling*. 2019. arXiv: 1911.06393 [cs.LG]. URL: https://arxiv.org/abs/1911.06393.

[TLR21] Margaret Trautner, Ziwei Li, and Sai Ravela. "Learn Like The Pro: Norms from Theory to Size Neural Computation". In: *arXiv preprint arXiv:2106.11409* (2021). URL: https://arxiv.org/abs/2106.11409.

[TT23] VL. Vo Tran and Nguyen TC. "One-dimensional convolutional neural network for damage detection". In: *Springer* (2023).

[TZ22] Peiwang Tang and Xianchao Zhang. "Features Fusion Framework for Multimodal Irregular Time-series Events". In: *Springer*. 2022. URL: %5Curl%7Bhttps://link.springer.com/chapter/10.1007/978-3-031-20862-1_27%7D.

[Umu+17] Yaman Umuroglu et al. "FINN: A Framework for Fast, Scalable Binarized Neural Network Inference". In: *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. FPGA '17. ACM, 2017, pp. 65–74.

[Wan+20] Maolin Wang et al. "NITI: Training Integer Neural Networks Using Integer-only Arithmetic". In: *arXiv preprint arXiv:2009.13108* (2020). URL: https://arxiv.org/abs/2009.13108.

[YR23] Erina Yamaguchi and Sai Ravela. *Multirotor Ensemble Model Predictive Control I: Simulation Experiments*. 2023. arXiv: 2305.12625 [eess.SY]. URL: https://arxiv.org/abs/2305.12625.

[Zha+23] Zhuoyi Zhang et al. "Exploring the Potential of Flexible 8-bit Format: Design and Algorithm". In: *arXiv preprint arXiv:2310.13513* (2023).

Thelonious Cooper is a senior undergraduate at MIT studying course Electrical Science & Engineering with a focus in digital hardware and systems science.