Bespoke: an ONNX to FPGA compiler for Rapid Development of ML Stream Processors

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

***Abstract*—*Bespoke* is a system for synthesizing FPGA stream processors from 8-bit quantized neural network specifications. For edge ML applications, power efficiency and speed are paramount. The system described presents an alternative to running intensive ML routines on microcontrollers, allowing for edge systems to use cheaper and more power efficient microcontrollers alongside a *bespoke* FPGA that can be dynamically reconfigured to suit any ML stream-processing task. The model could even be modified or tuned on the edge by the microcontroller. Suitable applications include sensor data processing, state estimation and control, forecasting, and decision making. Repository available at https://github.com/theloni-monk/Bespoke**

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

I. Introduction

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 difficulties. A key concern with edge machine learning is the computation and power requirements of ML routines. Common general purpose microcontrollers can often not 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 microcontroller 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, I propose purpose-built accelerators that only run a particular network. 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 8bit formats. Many so called "8-bit quantization" implementations in popular neural network frameworks like Tensorflow[Mar+15], PyTorch[Pas+19], and ONNX[dev21] fail to create a fully 8-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.

III. Machine Learning Composable Components

As a generic architecture all modules will follow a similar format. Each module will be connected to its previous and next nodes via register-based FIFOs 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 flag. In a future implementation, the FIFOs could use BRAM and have masking to acheive the varied read and write widths. This would significantly reduce the LUT usage in the synthesized design.

A. Static Matrix Vector Product

Perhaps the most important operation of any ML system is the matrix-vector product. The MVProd 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 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. The pipelined adder tree

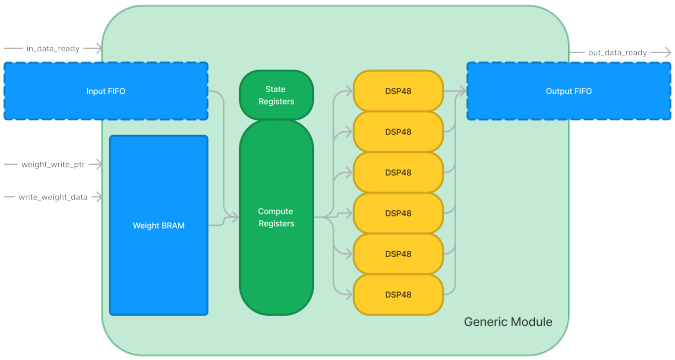

Fig. 1. Generic module architecture which is specified by each ML component

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 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. As it stands the MVProd module requires

$$OutVecLength * \lceil \log_2(WorkingRegs) \rceil + \frac{InVecLength * OutVecLength}{WorkingRegs}$$

cycles to complete an operation.

B. Static Vector Vector Addition

Bias, or static vector-vector addition is another common ML operation. It is implemented with a BRAM instance for its biases and a FIFO for the input vector. It is implemented as a Mealy FSM performing *WorkingRegs* parallel additions per cycle.

C. Dynamic Vector Vector Addition

Dynamic Vector-Vector addition takes vectors from two FIFOs and outputs their sum to a third FIFO. This operation is important because it allow for branching computations and recurrence.

D. Dynamic Vector Vector Multiplication

Dynamic Vector-Vector multiplication is the multiplicative analog of the dynamic vector-vector addition module. This operation is common for masking or attention models.

E. Rectified Linear Unit (ReLU)

The Rectified Linear Unit or ReLU is a common activation function which simply zeros out all elements below zero. It allows for strong nonlinearity and masking. It is implemented as a Mealy FSM, performing conditional evaluation of *WorkingRegs* bytes at a time.

IV. Dynamic Model Hardware Generation

The second aspect of my project is the model parsing and Verilog 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.

B. BRAM Allocation and .mem Generation

In this stage weights from the integer matrix initializer fields of the ONNX model will be converted to .mem format in this stage and the compiler will error if it runs out of space.

C. 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 mutlipliers, thus minimizing LUT usage. Here the gross latency is computed and reported to the user as well. The latency of a ”square” network in this scheme is

$$\frac{VectorDimension^2}{\#DSPsPerMatMul} \frac{3 * NetworkDepth}{\#MatMuls}$$

But this formula becomes more complicated for relatively prime vector dimensions and # DSP blocks

D. 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.

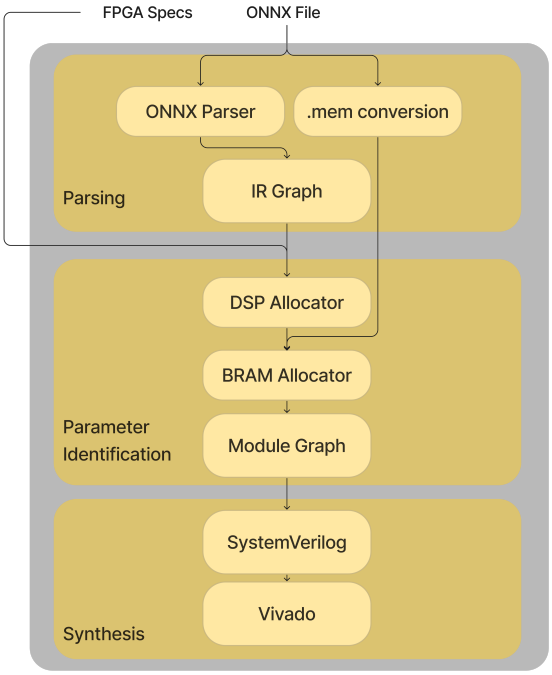

Fig. 2. ONNX-to-FPGA compiler architecture diagram

V. Results and Performance Analysis

A. Verification Methodology

Bespoke's models have been verified by generating ONNX models with random integer weight and bias matrices that are then compiled to a model and simulated. This test was performed for 15 models of various sizes and the simulated FPGA produced identical results to the pPthon ONNX Reference Evaluator. In order to perform this verification a script was written that generates onnx specifications for 8 bit neural networks of specified dimension.

B. Timing and Utilization Analysis

Analysis was performed on 3-layer fully connected square networks optimized according to the Spartan-7 specification. It

Width	Params	Slice LUT	Slice Reg	BRAM	DSP
5	90	2.32	1.19	0	15
10	330	5.31	1.89	6	25
20	1260	6.98	3.56	10	50
40	4920	12.69	8.18	18	100
50	7650	34.86	10.34	12	62.5
75	17100	50.46	13.78	14	62.5
80	19440	38.37	16.16	20	100
100	30300	61.05	18.49	14	62.5

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

is of note that all of the utilization categories are significantly sublinear with respect to number of parameters. 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

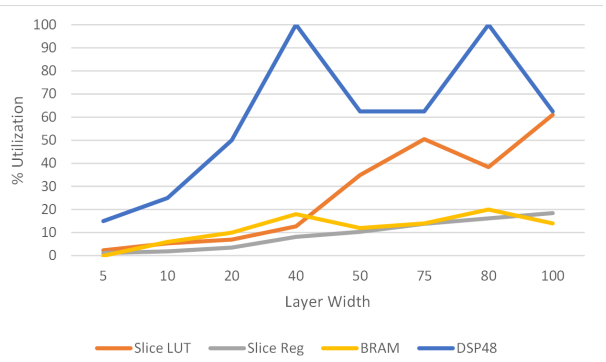

Fig. 3. Utilization of FPGA resources vs Layer Width

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.

VI. Discussion and Applications

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 (the fpga dark side). 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.

VII. Conclusion and Future Work

Although ML accelerators are far from new, there don't exist low-cost methods for creating cheap stream processors for any particular ML application. So I ask the question, why pay for generality when its not needed? Bespoke will deliver 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.

If I were to have more time with this project, I would reimplement the FIFOs in 2-cycle BRAM with byte masks for varied read and write widths. This would significantly improve the potential ability to scale and could allow for the processing of very high dimensional information such as images. I would also implement several more modules, such as arbitrary convolution via FFT, MaxPooling for downscalling in large CNNs, and other activation functions. This would expand the number of networks realizable in the hardware framework.

I have thoroughly enjoyed working on this project and would be happy to continue on it as I think it is something I could imagine using in several interesting embedded applications. I have enjoyed the way that digital hardware development makes you think. Things can be very complicated but their fundemental operation is so directly tied to what you've typed that you (mostly) have yourself to blame when things go wrong. I find this paradigm satisfying. I am interested in pursuing systems science with embedded applications for ubiquitous signal processors.

ACKNOWLEDGMENT

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

REFERENCES

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

[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.

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

[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.

[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.

[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.

[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 3rd year undergraduate at MIT studying course 6-1 (Electrical Engineering) with a focus in hardware and systems science.