FPGA-Based Deep Learning Accelerator with TVM Compiler

Date: 11/25/2020

Faculty Mentor: Michael Orshansky

Team H1: Jeffrey Marshall, Zach Sisti, Sergio Chacon, Elgin Allen, Troy Jackson, Michael Pontikes

Project Definition

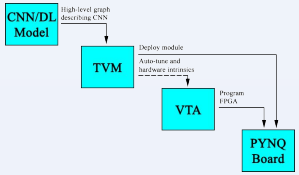
GOAL: This project tests the efficacy of the Tensor Virtual Machine (TVM) compiler [1], the Versatile Tensor Accelerator (VTA) architecture [2], and the PYNQ Field Programmable Gate Array (FPGA) board, to accelerate the inference performance of Convolutional Neural Networks (CNN) used for image classification.

MOTIVATION: Increasing investment into the field of deep learning (DL) has led to a variety of model frameworks and hardware backends. TVM is an open-source compiler stack that strives to compile, optimize, and deploy DL models from any framework to any hardware backend. For this project we are focused on utilizing the parallelism and flexibility of FPGAs as a hardware backend to accelerate CNN inferences.

ENGINEERING REQUIREMENTS: Knowledge in Neural Networks, Deep Learning Compilers, Computer Architecture, Hardware Acceleration, FPGAs, and Python.

System Design

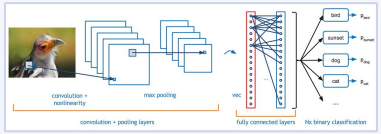
MODULES: CNN Model, TVM Compiler, AutoTVM, VTA, PYNQ Board.



HIGH LEVEL OVERVIEW: A high level graph describing a CNN is imported to TVM, where it is compiled into a deployable module. During compilation high level graph optimizations such as operator fusion and layout transformations occur. TVM then lowers the graph onto the desired target hardware, in this case the PYNQ board. The PYNQ board's FPGA is programmed with VTA and runs the desired inference application. If AutoTVM is used, then the tuned log file will be applied to VTA before VTA is deployed to the PYNQ board.

CNN Model

We imported ResNet-18 and ResNet-50 models from the MXNet DL framework that were pre-trained with the ImageNet database. Different ResNet variations were chosen to emphasize improvements made by the compiler as more convolutional layers are added and optimized.



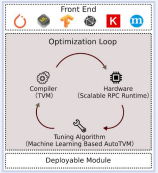
TVM

TVM is a deep learning compiler created by the University of Washington. The compiler uses Relay IR to convert an imported CNN model into a computational graph. Operators are then lowered to TVM's TOPI library by creating a schedule and compute function for each.

To extend TVM to VTA, the CNN model must also be quantized, changed from a float32 data type to int8, and graphpacked to correctly layout data onto the VTA architecture.

AutoTVM

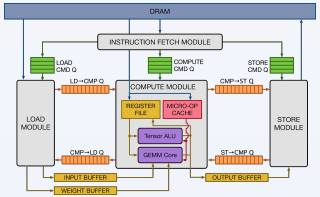
Compiled DL models can be optimized by extracting operators, defining a search space with a tuning algorithm, running trials on each implementation, and recording the best results. The table below shows our evaluations of the different tuning algorithms paired with VTA.



Hardware	Autotuner	Inference Time Average (ms)
SoC + VTA	Default Tuning	1083.29
SoC + VTA	Random	1112.36
SoC + VTA	Grid Search	1152.5
SoC + VTA	Genetic Algorithm	1094.68
SoC + VTA	XGB	1075.3

VTA

VTA is a DL architecture specified in Vivado HLS C++ that utilizes a Task Level Pipeline and a 2-level ISA to run the optimized low-level program. We generated custom bitstreams for VTA at clock periods of 7, 6, and 5 ns, but saw no improvement in inference time with our aggressive pipeline. A breakdown of the VTA architecture can be seen below.



PYNQ Board

For our project we had a PYNQ Z1 and a PYNQ Z2 board. However, both contain the same Zynq 7000 Series SoC which includes a 650 MHz Dual Core ARM A9 Processor (SoC CPU) and Artix-7 family programmable logic:

- -13,300 logic slices with 4 6-input LUTs and 8 flip flops
- -630KB of fast block RAM
- -220 DSP slices

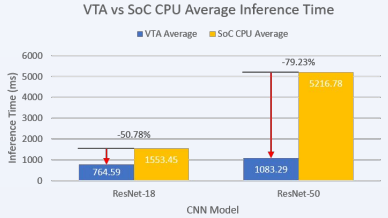
Module	BRAM_18K	DSP48E	FF	LUT
Compute	77	138	14995	30709
Store	4	0	1481	2465
Load	4	0	4241	18146
Fetch	12	0	972	1189
Total	97	138	21689	52509
Available	280	220	106400	53200
Utilization	**34.6%**	**62.7%**	**20.4%**	**98.7%**

The table above shows the utilization summary for our current VTA design. We encountered high LUT utilization as a result of the intensive computations in the GEMM core and Tensor ALU. Given more time, we would've liked to utilize more of the BRAM and efficient DMA usage to speed up memory accesses and decrease inference time.

Testing and Evaluation

METHODS: Evaluated inference time of each successive optimization component of our accelerator against a comparable hardware backend in the PYNQ board's SoC CPU (Dual Core ARM A9 Processor).

RESULTS: Inference time is significantly dropped by our FPGA Based Accelerator. VTA architecture with AutoTVM decreased inference time by a factor of x5 when running ResNet-50 compared against SoC CPU with AutoTVM and a factor of x2 for running ResNet-18.



Conclusion

Our testing confirmed that TVM is a viable DL compiler and that our FPGA-based accelerator significantly improved inference time. Additional testing also confirmed that the transformations done by TVM and VTA did not reduce the model's accuracy when compared with the accuracy rates listed on MxNet. We found that while AutoTVM did lead to decreased inference time, there was little variation in performance when utilizing different tuning algorithms. VTA was also proven to perform the same with a boosted clock frequency. The high LUT and low BRAM usage is an area for further improvement and study of the FPGA hardware accelerator.

References

[1] T. Chen et al, "TVM: An Automated End-to-End Optimizing Compiler for Deep Learning," USENIX Assoc., Carlsbad, CA, Rep. 2018.

[2] T.Moreau et al., "A Hardware-Software Blueprint for Flexible Deep Learning Specialization," Dept. Comp. Science & Eng, Univ. Washington, Seattle, WA, Rep.