Work-in-Progress: Offloading Cache Configuration Prediction to an FPGA for Hardware Speedup and Overhead Reduction

Ruben Vazquez
Department of Electrical and Computer Engineering
University of Florida
Gainesville, United States
ruben.vazquez@ufl.edu

Ann Gordon-Ross
Department of Electrical and Computer Engineering
University of Florida
Gainesville, United States
anngordonross@ufl.edu

Greg Stitt
Department of Electrical and Computer Engineering
University of Florida
Gainesville, United States
gstitt@ece.ufl.edu

***Abstract*—In this paper, we present our cache configuration prediction methodology offloaded to an FPGA for improved performance and hardware overhead reduction, while maintaining cache configuration predictions within 5% of the *optimal* energy cache configuration for application phases for the instruction and data caches.**

Keywords—cache, configurable, artificial neural network, mobile devices, FPGA, phases, machine learning

I. Introduction

Architectural specialization helps meet the design constraints of embedded systems. Modern embedded systems are expected to have a performance comparable to desktop systems, but with much more stringent design constraints, such as area and energy consumption. Configurable hardware can help to meet those design constraints by specializing the hardware (i.e., processor, cache, etc.) to each specific application. However, exploring different hardware configurations for each application can be costly [6]. The exploration cost is further increased when considering application phases as well. Therefore, an efficient exploration method is required to yield maximum benefits without incurring a large configuration evaluation overhead.

Machine learning techniques can be leveraged to predict *a best* cache configuration. We define a cache configuration as a set of values defining each configurable parameter of the cache. For example, for configurable total size, associativity, and line size, a valid cache configuration can be (2 kB, 2-way, 64B). Machine learning techniques eliminate the configuration evaluation overhead by *predicting* the cache configuration rather than exploring the design space using a set of runtime statistics (i.e., CPI, instruction profile, etc.) collected once during a profiling execution. These statistics serve as input to a machine learning model. Machine learning techniques for architectural specialization have been used in prior works [3][8]. However, these models can prove to be too costly to implement on embedded devices. To alleviate the issue of costly implementation, prior works have also focused on making machine learning models efficient for running on embedded devices, mobile devices, and FPGAs [5][7]. However, writing hardware description language (HDL) code for machine learning models can be costly in terms of designer time and effort. Thus, our goal is to be able to offload our prediction methodology from embedded devices without expending much designer time or effort.

We implement a cache configuration prediction artificial neural network (ANN) on an FPGA to exploit parallelism inherent in neural networks. To reduce designer time and effort, we use LeFlow [2], a framework that creates synthesizable Verilog code from TensorFlow code. LeFlow allows designers to create TensorFlow models very quickly without having to write low-level HDL code. We show that we can create neural networks that predict the cache configuration for different application phases with low energy degradations (within 5% of the optimal cache configuration) with low resource usage based on a Cyclone IV DE2-115 FPGA kit.

II. Cache Configuration Prediction and ANN Implementation

Different applications running on a system can benefit from using different cache configurations. Cache parameter specialization can lead to various benefits, such reduced execution time and reduced energy consumption. However, as the cache configuration space grows, it quickly becomes infeasible to search the space using either exhaustive or heuristic approaches [6]. The cache configuration space can grow by either increasing the number of configurable parameters (e.g., total cache size, associativity, etc.) or by increasing the number of available parameter values (e.g., 2 kB total size, 128 B line size, etc.). To alleviate the overhead of exploration, we propose our cache configuration prediction methodology, where an ANN is created to predict *a best* cache

 CODES/ISSS '19 Companion , October 13–18, 2019, New York, NY, USA © 2019 Association for Computing Machinery. ACM ISBN 978-1-4503-6923-7/19/10…$15.00
https://doi.org/10.1145/3349567.3351722

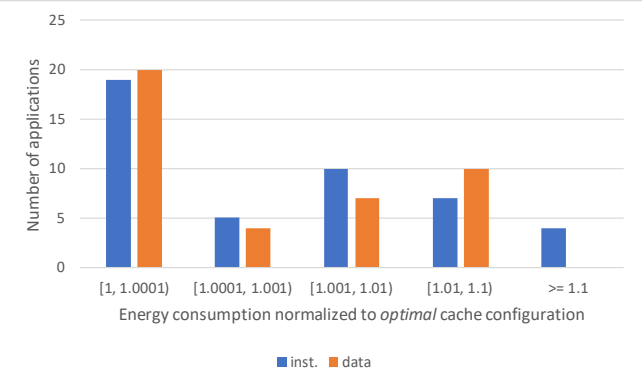

Fig. 1. Energy consumption normalized to the energy consumption of the optimal cache configuration

configuration for each distinct (i.e., unique) application phase based on the runtime statistics gathered for each distinct phase through one profiling run of the phase.

For our methodology, we choose an ANN for predicting the cache configurations over other models for several reasons. The outputs of the ANN are one-hot encoded, meaning each cache configuration is identified by one of the output neurons of the network. Further, decision trees cannot model complexities beyond simple comparisons of the inputs using pre-determined threshold values. Additionally, we can exploit the parallelism of neural networks using FPGAs, which is where we want to deploy our ANN model. However, writing HDL for a neural network can become quite complex and time-consuming, thereby necessitating a quick way to synthesize ANNs on FPGAs.

To reduce the designer time and effort, we use LeFlow [2], a framework that can synthesize Verilog code from a TensorFlow specification [1]. Using LeFlow, we synthesized two ANNs with an input layer of 91 and 175 elements (feature inputs required for the instruction and data caches), a hidden layer of 10 processing elements, and an output layer of 18 processing elements (representing each of the available cache configurations). The network is trained using TensorFlow for 1000 epochs, with early stopping enabled with respect to the validation accuracy. Once training is done, the circuit is synthesized by LeFlow from the TensorFlow file.

III. Experimental Results and Discussion

The TensorFlow code, hardware testing, and synthesis was executed using a virtual machine provided by the LeFlow GitHub repo [2]. The data collection, training, and software testing of the network was done on an Intel® Xeon® CPU E5520 running at 2.27 GHz. We use the EEMBC [4] and MiBench [7] benchmark suites with small and large inputs to represent common embedded system kernels. In this section, we present the performance of the neural networks as well as the number of cycles required to perform the prediction of the cache configurations for all of the benchmarks and discuss various FPGA metrics associated with implementing our ANNs on a Cyclone IV DE2-115 FPGA kit.

This work was supported by the National Science Foundation (CNS-0953447 and CNS-1718033). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

TABLE I
FPGA metrics for implemented ANN

	Instruction cache	Data cache
Cycles	33256	56776
Fmax (MHz)	61.11	66.37
Wall-clock time (ms)	5	9
Logic Elements	13094	13109

Fig. 1 shows the energy consumption of the applications using the predicted cache configurations normalized to the energy consumption of the optimal cache configurations. For almost all of the benchmarks, the energy degradation stays within 10% of the minimum energy consumption for both the instruction and data caches. Exceptions include the stringsearch and qsort benchmarks, which had unusually higher misprediction rates due to unpredictable control flow (i.e., quicksort and variable string search).

TABLE I shows some FPGA metrics concerning the implementations of the two ANNs, where Fmax is the maximum frequency of the implemented circuit. For comparison, we measured the number of software and hardware cycles during the testing phase of our neural network The software cycles, measured using the Linux *perf* [10] tool, was over 13 billion cycles at 2.40 GHz compared to 33256 and 56776 cycles at about 60 MHz, leading to a significant speedup of 800X and 400X for the instruction and data caches, respectively, using only 13000 LEs (10%) of the Cyclone IV DE2-115 hardware resources.

References

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu and X. Zheng, "TensorFlow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016

[2] arXiv:1807.05317 [cs.LG].

[3] B. Dutta, V. Adhinarayanan, and W. Feng. 2018. GPU power prediction via ensemble machine learning for DVFS space exploration. In Proceedings of the 15th ACM International Conference on Computing Frontiers (CF '18). ACM, New York, NY, USA, 240-243.

[4] EEMBC. The Embedded Microprocessor Benchmark Consortium http://www.eembc.org/benchmark/automotive _sl.php, Sept. 2013

[5] V. Gokhale, J. Jin, A. Dundar, B. Martini and E. Culurciello, "A 240 G-ops/s Mobile Coprocessor for Deep Neural Networks," *2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, Columbus, OH, 2014, pp. 696-701.

[6] A. Gordon-Ross, F. Vahid and N. Dutt, "Fast configurable-cache tuning with a unified second-level cache," *ISLPED '05. Proceedings of the 2005 International Symposium on Low Power Electronics and Design, 2005.*, San Diego, CA, 2005, pp. 323-326.

[7] M. R. Guthaus, J. S. Ringenberg, D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown. 2001. MiBench: A free, commercially representative embedded benchmark suite. In Proceedings of the Workload Characterization, 2001. WWC-4. 2001 IEEE International Workshop (WWC '01). IEEE Computer Society, Washington, DC, USA, 3-14.

[8] S. Khakhaeng and C. Chantrapornchai, "On the finding proper cache prediction model using neural network," *2016 8th International Conference on Knowledge and Smart Technology (KST)*, Chiangmai, 2016, pp. 146-151.

[9] N. D. Lane *et al.*, "DeepX: A Software Accelerator for Low-Power Deep Learning Inference on Mobile Devices," *2016 15th ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)*, Vienna, 2016, pp. 1-12.

[10] "Perf events tutorial," http://perf.wiki.kernel.org/, 2012.