Uncovering Performance Opportunities by Relaxing Program Semantics of GPGPU Kernels

Jhe-Yu Liou
Arizona State University
Tempe, AZ
jhe-yu.liou@asu.edu

Stephanie Forrest
Arizona State University
Tempe, AZ
Stephanie.Forrest@asu.edu

Carole-Jean Wu
Arizona State University
Tempe, AZ
carole-jean.wu@asu.edu

1 INTRODUCTION

Approximate computing trades off computation precision for desirable properties such as improved runtime or energy efficiency, typically through methods such as precision scaling or task skipping. These methods are particularly appropriate for error-tolerant applications such as image processing, e.g., [2]. Here we focus on Machine Learning (ML) kernels, in order to expose additional performance-accuracy optimization opportunities [4]. By relaxing the requirement to preserve semantics, our wacky idea involves pushing error tolerance in ML kernels down to the code level in exchange for finding performance tuning opportunities on GPUs.

Over the past decade, stochastic search algorithms have been applied to a variety of software engineering problems with increasing success, despite the fact that they do not necessarily enforce exact semantic equivalence, relying instead on test suites to encode the required behavior of the program [3]. Earlier work used evolutionary computation to evolve neural network architectures [7], but this is the first work we know of to evolve the code that implements the ML kernels. There have been few attempts to apply stochastic search to the LLVM intermediate representation (LLVM-IR), in part because the IR has many data dependencies and requires careful implementation of code modification operations. Despite this challenge, LLVM-IR is the only post-compiler-optimized representation for CUDA GPU program with ease of manipulation from the LLVM compiler framework, while the other 2 possible representations, PTX and SASS, are not supported by reliable and open-sourced compiler tools.

We propose GEVO-approx (Gpu EVOlution for approximate computing), a post-compilation performance tuning approach, to discover optimized GPGPU kernel implementations using Genetic Programming (GP). GEVO-approx encodes desired optimization objectives as the fitness function and implements a set of mutation and recombination operators for GPU kernel transformations at the LLVM-IR granularity. GEVO evolves kernel implementations based on the fitness of individuals in the population. We show how GEVO-approx can simultaneously tune code to meet two independent objectives—runtime and accuracy—using GPGPU kernels on NVIDIA Tesla P1000 GPUs. GEVO-approx improves kernel runtime performance from 12% to 393%, and our analysis reveals interesting optimizations that cannot be realized using traditional compiler optimization techniques. We find architecture-, application-, and dataset-specific performance tuning opportunities

2 THE PROPOSED DESIGN: GEVO-APPROX

GEVO-approx, takes as input a GPGPU program, a comprehensive set of test cases that specify required program functionality, and a multi-dimensional fitness function for optimization. GEVO-approx attempts to maximize the fitness function by evolving and evaluating mutated kernel variants in an iterative population-based search. GEVO's basic design follows earlier work applying Genetic Programming (GP) to software [5, 6], but many details have been redesigned to accommodate the LLVM-IR. Kernels in a GPGPU program are first compiled into LLVM-IR with the clang compiler. GEVO-approx then takes kernels in the LLVM-IR format as inputs and uses GP to discover improved kernel implementations as defined by the fitness function. Figure 1 depicts the execution flow, highlighting the key operations.

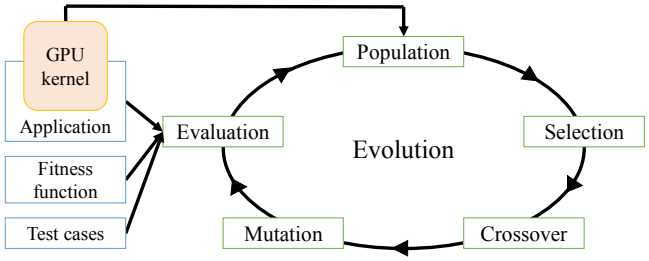

Figure 1: GEVO-approx Execution Flow.

GEVO-approx searches for high-fitness program variants using three operators: **Mutation**, **Crossover**, and **Selection**. *Mutation* introduces a random change to a single instruction in the program. *Crossover* hybridizes two individual program variants, by exchanging subsections of their code, and *Selection* chooses program variants to be retained in the search according to fitness. Since our fitness function is two-dimensional, the algorithm returns a Pareto set of options, performing multi-objective optimization.

3 EXPERIMENTAL SETUP

Experimental Systems We instrumented the LLVM-8 compiler with redesigned C++ genetic operators. For each GPGPU application, GEVO-approx is given a 48-hour budget to search for optimizations. All GEVO experiments were conducted with population size of 256, crossover rate of 80% (i.e., 80% of individuals in population are selected for crossover), and a mutation rate of 30% (i.e., every individual has 30% chance to get one mutation).

Applications and Test Suites: To assess GEVO-approx, we use error-tolerant GPGPU applications, Particle Filter (`pf`), Needleman-Wunsch (`nw`), and Hotspot (`hs`) [1], and ThunderSVM, an open-source support vector machine library for GPUs [8]. We constructed two workloads for the SVM using two different datasets: handwriting recognition (MNIST) and income prediction (a9a).

To validate kernel variants, we use all the default test suites input, and we generate many additional test using application input generators, generating from tens of thousands to millions of input values. For ML workloads, we evaluate on the training dataset using 2-fold cross validation, rejecting kernel variants that exceed

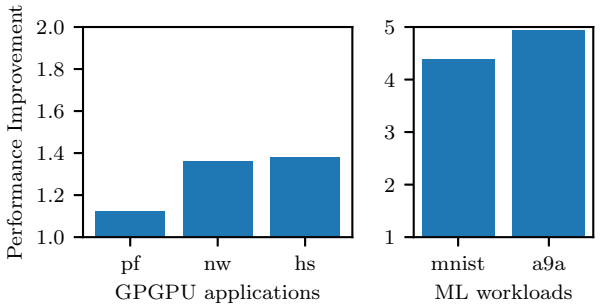

Figure 2: The best performance improvement over default baseline under 1% error tolerance.

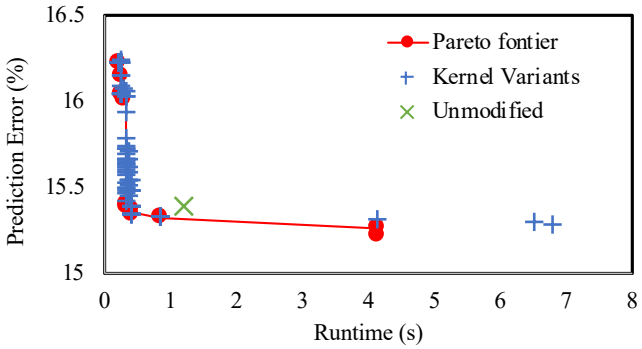

Figure 3: The Pareto-frontier of kernel variants for a9a.

1% output error or 1% additional training error. After training, we evaluate on the default testing dataset, and then test for generality on a much larger training dataset.

4 EARLY RESULTS

Figure 2 presents the performance improvement opportunities uncovered by GEVO-approx over the default baseline with full compilation optimization. With 1% output accuracy relaxation, GEVO-approx improves runtime performance from 12% for `pf` to 38% for `hs`. GEVO-approx uncovers significant performance tuning opportunities for the ML workloads, leading to 4.38X and 4.93X training time reduction for `mnist` and `a9a`. We used our mnist-optimized kernel to train on a supplied large dataset instead of the training dataset (8,000,000 vs 60,000 image samples) training time was reduced further from 1181 to 121 minutes, 9.76X reduction.

Figure 3 depicts the Pareto frontier of kernel variants found for `a9a`. The best kernel of `a9a` is the leftmost data point, which incurs 0.84% accuracy penalty for large runtime improvement. ML application engineers can navigate the Pareto-frontier generated by GEVO-approx to identify a best-performing kernel variant that meets the desirable error rate. Sometimes it is possible to find improvements in both dimensions, e.g., the bottom left data point might has better accuracy than the baseline and achieves 2.93X training time reduction.

We manually inspected the the *fittest* kernel variants to analyze what changes led to the improvements. Although some improvements came from single mutation operations, others involved combinations of mutations, which could not be achieved independently. Here are the most common optimizations we found:

- Removing synchronization primitives (in `hs`, `nw`): Although generally risky, some `syncthread()` calls in CUDA can be removed because the thread scheduler in the GPU hardware provides redundant synchronizations under particular memory access patterns.
- Removing conditional execution (in `hs`, `pf`): GEVO eliminates code blocks from the conditional path when the input space does not touch that portion of the kernel.
- Loop perforation (in `hs`): GEVO discovers loop perforations, for example, when loops have been unrolled heavily post-compilation. GEVO then removes some part(s) of the unrolled loop.
- Memoization (in `hs`): GEVO identifies memoization opportunities by eliminating unneeded instructions and using stored results directly. `hs` performs some pre-processing based on the physical dimension of the processor chip. Since the shape of simulated chips is the same across all loop iterations, GEVO discovers opportunities to reuse the preprocessing results of the x-dimension for the y-dimension.
- Convergence relaxation (ML workloads): GEVO-approx modified the behavior of the ML training kernel, by indirectly relaxing its convergence bound and reducing training time significantly. In SVM, the regularization parameter is a user-adjustable variable that determines the balance between generalization and model accuracy, controlling the convergence condition.

5 SUMMARY

GEVO explores performance tuning opportunities for approximate computing by relaxing program semantics of GPGPU kernels. GEVO-approx finds kernels with 12-38% performance speedup with 1% error tolerance for `pf`, `nw`, `hs`, and 4.38/4.93 times training time reduction for the ML models. GEVO-approx can effectively exploit test-case based semantics to tune LLVM-IR code with application-specific, architecture-specific, and dataset-specific optimizations. Although preliminary, these results suggest that significant performance gains can be achieved by relaxing semantic correctness and carefully tuning LLVM-IR codes to their expected operating conditions. For applications, like deep learning, where architectures are designed for specific kinds of datasets, GEVO can potentially accelerate training times significantly with minimal accuracy penalty.

REFERENCES

[1] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing.

[2] Sparsh Mittal. 2016. A Survey of Techniques for Approximate Computing. *ACM Comput. Surv.* (2016).

[3] Justyna Petke, Saemundur O Haraldsson, Mark Harman, William B Langdon, David R White, and John R Woodward. 2018. Genetic Improvement of Software: A Comprehensive Survey. *IEEE Transactions on Evolutionary Computation* (2018).

[4] Adrian Sampson, Werner Dietl, Emily Fortuna, Danushen Gnanapragasam, Luis Ceze, and Dan Grossman. 2011. EnerJ: Approximate Data Types for Safe and General Low-power Computation. In *Proc. of the ACM SIGPLAN Conf. on Programming Language Design and Implementation.*

[5] Eric Schulte, Jonathan DiLorenzo, Westley Weimer, and Stephanie Forrest. 2013. Automated Repair of Binary and Assembly Programs for Cooperating Embedded Devices. In *Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems.*

[6] Eric Schulte, Jonathan Dorn, Stephen Harding, Stephanie Forrest, and Westley Weimer. 2014. Post-compiler Software Optimization for Reducing Energy. In *Proc. of the Intl. Conf. on Architectural Support for Programming Languages and Operating Systems.*

[7] Kenneth O. Stanley and Risto Miikkulainen. 2002. Evolving Neural Networks Through Augmenting Topologies. *Evol. Comput.* (2002).

[8] Zeyi Wen, Jiashuai Shi, Qinbin Li, Bingsheng He, and Jian Chen. 2018. ThunderSVM: A Fast SVM Library on GPUs and CPUs. *Journal of Machine Learning Research* (2018).