Work-in-Progress: Toward a Robust, Reconfigurable Hardware Accelerator for Tree-Based Genetic Programming

Christopher Crary, Wesley Piard, Britton Chesley, Greg Stitt
Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL 32603, USA
{ccrary, wespiard, bchesley97, gstitt}@ufl.edu

***Abstract*—Genetic programming (GP) is a general, broadly effective procedure by which computable solutions are constructed from high-level objectives. As with other machine-learning endeavors, one continual trend for GP is to exploit ever-larger amounts of parallelism. In this paper, we explore the possibility of accelerating GP by way of modern field-programmable gate arrays (FPGAs), which is motivated by the fact that FPGAs can sometimes leverage larger amounts of both function and data parallelism—common characteristics of GP—when compared to CPUs and GPUs. As a first step towards more general acceleration, we present a preliminary accelerator for the evaluation phase of "tree-based GP"—the original, and still popular, flavor of GP—for which the FPGA dynamically compiles programs of varying shapes and sizes onto a reconfigurable function tree pipeline. Overall, when compared to a recent open-source GPU solution implemented on a modern 8nm process node, our accelerator implemented on an older 20nm FPGA achieves an average speedup of 9.7×. Although our accelerator is 7.9× slower than most examples of a state-of-the-art CPU solution implemented on a recent 7nm process node, we describe future extensions that can make FPGA acceleration provide attractive Pareto-optimal tradeoffs.**

***Index Terms*—genetic programming, reconfigurable computing, FPGA devices**

I. Summary

In general, the parallelism opportunities implicit to tree-based genetic programming (GP) are not fully realized by general-purpose CPU/GPU systems [1], [2], [3]. Specifically, although CPU/GPU systems can be made to exploit both data and function parallelism—e.g., by evaluating multiple data points, multiple operations, or multiple candidate solutions in parallel [3], [4], [5]—these efforts are ultimately limited by general-purpose execution/memory models or by frequent, dynamic changes in control flow [2], [4], [5]. In this paper, we improve upon such limitations of CPU/GPU systems with a hardware accelerator specialized to the evaluation phase of tree-based GP, implemented by way of a modern, mid-range FPGA device.

As shown in Fig. 1, our accelerator leverages a specialized, full tree of generic computing resources that can compute any program consisting of functions supported by the generic resources, as long as the depth of the program is not larger than the depth of the tree, the latter of which is defined by the user. By then pipelining the generic resources, the accelerator can generate program outputs *every clock cycle* after some initial latency. To further increase throughput, the accelerator also dynamically compiles programs for the tree while evaluating, so that the tree may switch between programs *within a single clock cycle*. Importantly, such forms of parallelism have not been achieved via general-purpose CPU/GPU architectures.

Average FPGA Speedup				
Fitness Case Threshold (≤)	Tool			
	DEAP	TensorGP (CPU)	TensorGP (GPU)	Operon
10	427×	663×	931×	0.227×
100	415×	622×	860×	0.215×
1,000	270×	286×	388×	0.097×
10,000	187×	60.2×	73.0×	0.127×
100,000	146×	18.8×	9.7×	0.127×

TABLE I: Average NEPS speedups for various fitness case thresholds. For a given threshold value, the average is calculated from all results regarding thresholds less than or equal to this value. The last row represents an overall average.

We compare the performance of our architecture with the evaluation engines given by three GP software tools: *DEAP*, *TensorGP*, and *Operon*.[1] From each tool, we use the evaluation engine—and no evolution engine—to execute a large set of randomly generated programs for various amounts of *fitness cases* (i.e., sample points), and we estimate evaluation performance in terms of *node evaluations per second* (*NEPS*). For each software-based tool, we utilize a 4.8 GHz, 12-core 7nm AMD Ryzen 5900X CPU, and, additionally, an 8nm Nvidia RTX 3080 GPU for TensorGP. To implement our hardware accelerator, we utilize a 20nm mid-range Intel Arria 10 10AX115N2F40E2LG GX FPGA provided by an Intel Programmable Acceleration Card (PAC) through the Intel FPGA DevCloud service. We compile the accelerator by way of Quartus Pro 19.2.0, Build 57.

Table I presents average speedups for the FPGA in terms of numbers of fitness cases. Overall, compared to DEAP,

This material is based upon work supported by the National Science Foundation under Grant Nos. CNS-1718033 and CCF-1909244.

[1]Experiments: https://github.com/christophercrary/conference-cases-2022.

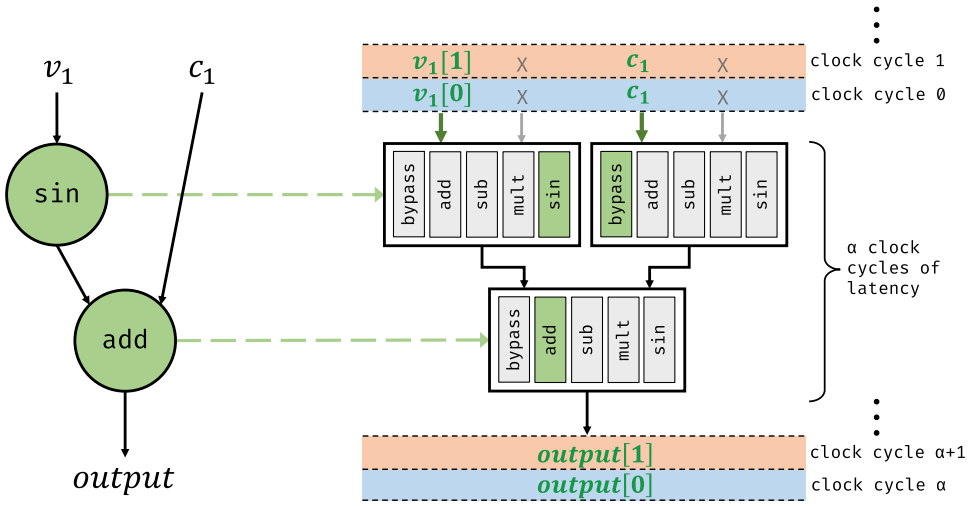

Fig. 1: An illustration of how the proposed GP accelerator can parallelize evaluation of different solutions and/or different training samples *every cycle* via a reconfigurable tree pipeline. Each node of the tree pipeline provides an implementation of every function within the GP primitive set, in addition to a bypass, which allows for arbitrary program shapes.

a baseline GP software tool that we ran parallelized across all cores of the chosen CPU, our architecture achieved an average speedup of 146×. Compared to TensorGP, a recent open-source GP software tool targeting both CPU and GPU systems, our architecture achieved an average speedup of 18.8× in regard to CPU execution and 9.7× in regard to GPU execution. Finally, compared to Operon, a recent state-of-the-art GP software tool targeting CPU systems, our current architecture executed about 7.9× slower on average.

Despite not achieving an average speedup over Operon, our preliminary architecture demonstrates potential for significant future improvements. First and foremost, we are currently comparing an older 20nm FPGA (released in 2013) to a newer 7nm/8nm CPU/GPU (released in 2020); notably, our chosen FPGA has 8× fewer floating-point DSP resources—our main constraint—and slower clock speeds than some newer FPGAs (e.g., [6]). Secondly, some GP applications can have additional computational bottlenecks, such as non-linear least squares for symbolic regression [5], [7], for which FPGAs have been shown to have significant advantages [8]. In fact, the notable solution quality frequently demonstrated by Operon—which is not evaluated within this work—generally relies on non-linear least squares [5], [9], which often significantly lowers runtime performance [7]. Lastly, it has been widely shown that FPGAs generally have power and energy advantages when compared to CPUs/GPUs [10], [11], sometimes with improvements of up to several orders of magnitude (e.g., [11]). Although we do not evaluate power and energy in this paper, if advancements to our architecture can achieve performance comparable to Operon, the power/energy advantages would likely be significant. Such advantages, in turn, could enable more energy-efficient (and, thus, potentially more cost-effective) multi-computer GP systems, as well as new embedded GP use cases.

Therefore, although this paper is a preliminary first step, we expect future extensions of our architecture to constitute a state-of-the-art, energy-efficient accelerator for many GP applications.

References

[1] R. Poli, W. B. Langdon, and N. F. McPhee, *A Field Guide to Genetic Programming*. Lulu Enterprises, UK Ltd, 2008.

[2] D. M. Chitty, "Fast parallel genetic programming: multi-core cpu versus many-core gpu," *Soft Computing*, vol. 16, pp. 1795–1814, Oct 2012.

[3] D. M. Chitty, "Faster gpu-based genetic programming using a two-dimensional stack," *Soft Computing*, vol. 21, pp. 3859–3878, Jul 2017.

[4] D. Robilliard, V. Marion-Poty, and C. Fonlupt, "Genetic programming on graphics processing units," *Genetic Programming and Evolvable Machines*, vol. 10, p. 447–471, Dec 2009.

[5] B. Burlacu, G. Kronberger, and M. Kommenda, *Operon C++: An Efficient Genetic Programming Framework for Symbolic Regression*, p. 1562–1570. New York, NY, USA: Association for Computing Machinery, 2020.

[6] Intel. Intel Agilex™ M-Series FPGA and SoC FPGA Product Table, 2015 [Online].

[7] M. Kommenda, B. Burlacu, G. Kronberger, and M. Affenzeller, "Parameter identification for symbolic regression using nonlinear least squares," *Genetic Programming and Evolvable Machines*, vol. 21, pp. 471–501, Sep 2020.

[8] J. Shawash and D. R. Selviah, "Real-time nonlinear parameter estimation using the levenberg–marquardt algorithm on field programmable gate arrays," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 1, pp. 170–176, 2013.

[9] W. La Cava, P. Orzechowski, B. Burlacu, F. de Franca, M. Virgolin, Y. Jin, M. Kommenda, and J. Moore, "Contemporary symbolic regression methods and their relative performance," in *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks* (J. Vanschoren and S. Yeung, eds.), vol. 1, 2021.

[10] Nurvitadhi *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '17, (New York, NY, USA), p. 5–14, Association for Computing Machinery, 2017.

[11] G. Stitt, A. Gupta, M. N. Emas, D. Wilson, and A. Baylis, "Scalable window generation for the intel broadwell+arria 10 and high-bandwidth fpga systems," in *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '18, (New York, NY, USA), p. 173–182, Association for Computing Machinery, 2018.