Asynchronous Parallel Cartesian Genetic Programming

Adam Harter
Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, MO 65401, USA
athb79@mst.edu

Daniel R. Tauritz
Natural Computation Laboratory
Department of Computer Science
Missouri University of Science and Technology
Rolla, MO 65401, USA
dtauritz@acm.org

William M. Siever
Department of Computer Science and Engineering
Washington University
St. Louis, MO 63130, USA
bsiever@wustl.edu

ABSTRACT

The run-time of evolutionary algorithms (EAs) is typically dominated by fitness evaluation. This is particularly the case when the genotypes are complex, such as in genetic programming (GP). Evaluating multiple offspring in parallel is appropriate in most types of EAs and can reduce the time incurred by fitness evaluation proportional to the number of parallel processing units. The most naive approach maintains the synchrony of evolution as employed by the vast majority of EAs, requiring an entire generation to be evaluated before progressing to the next generation. Heterogeneity in the evaluation times will degrade the performance, as parallel processing units will idle until the longest evaluation has completed. Asynchronous parallel evolution mitigates this bottleneck and techniques which experience high heterogeneity in evaluation times, such as Cartesian GP (CGP), are prime candidates for asynchrony. However, due to CGP's small population size, asynchrony has a significant impact on selection pressure and biases evolution towards genotypes with shorter execution times, resulting in poorer results compared to their synchronous counterparts. This paper: 1) provides a quick introduction to CGP and asynchronous parallel evolution, 2) introduces asynchronous parallel CGP, and 3) shows empirical results demonstrating the potential for asynchronous parallel CGP to outperform synchronous parallel CGP.

CCS CONCEPTS

•Theory of computation → Parallel computing models; Evolutionary algorithms; Genetic programming;

KEYWORDS

Genetic Programming, Asynchronous Parallel Evolution, Cartesian Genetic Programming, Evolutionary Computing

ACM Reference format:
Adam Harter, Daniel R. Tauritz, and William M. Siever. 2017. Asynchronous Parallel Cartesian Genetic Programming. In *Proceedings of GECCO '17 Companion, Berlin, Germany, July 15-19, 2017,* 5 pages.
DOI: http://dx.doi.org/10.1145/3067695.3084210

GECCO '17 Companion, Berlin, Germany

DOI: http://dx.doi.org/10.1145/3067695.3084210

1 INTRODUCTION

Cartesian Genetic Programming (CGP) arranges problem-specific operations as function nodes on a two-dimensional grid [7]. Unlike the genotypes in most forms of GP, these grids remain a static size and may need to be quite large to encapsulate complex solutions. Evaluating the fitness of this structure requires that input be passed to a set of initial nodes that then produces output for other nodes. Inputs are propagated from one function node to the next through the grid; however, not all nodes will necessarily be evaluated. The number of evaluated nodes in the genotype heavily influences the fitness evaluation time, therefore the variation in these times can become significant with large grid sizes. Much like most traditional evolutionary algorithms (EAs), evaluations of individuals are independent of each other in CGP and can be performed in parallel. Classic CGP employs the synchronous model common to the vast majority of EAs, in which all offspring in a generation are evaluated before survival selection is executed. Upon parallelization, the variation of evaluation times can cause classic CGP to excessively idle while waiting for individuals to be evaluated [6, 8]. To combat this problem, we are proposing an asynchronous model, in which survival selection is performed for each offspring individually immediately after evaluation is finished.

The contributions of this paper are as follows:

- Demonstrate statistical evidence that our proposed asynchronous parallel CGP (APCGP) may converge faster than synchronous parallel CGP (SPCGP) in regards to wall-time
- Provide analysis of scalability of APCGP with regards to problem complexity with comparison to SPCGP

2 RELATED WORK

Durillo et al. have shown empirical evidence supporting the significant improvement in terms of various quality metrics when employing asynchronous parallel EA's (APEAs) rather than synchronous parallel EAs for NSGA-II [3]. The APEA master process creates and sends individuals to be evaluated as the slave processors become idle. In the generational version, the population is replaced when enough offspring have been generated. With the steady-state alternative, the offspring are considered as each is received. The researchers employed homogeneous populations as the test cases during experimentation. Bertels and Tauritz performed similar experiments, evolving SAT solvers asynchronously and synchronously, with the asynchronous models outperforming the synchronous ones [1].

APEAs with heterogeneous populations have been found to be biased toward individuals with shorter evaluation times [2, 6, 9–11].

This is a result of the master process receiving those individuals sooner and more often, flooding the population. This potentially reduces the search space that can be reached within a given runtime. Yagoubi and Schoenauer attempt to circumvent this with a duration-based selection on the received offspring [10]. This supposed defect can also be taken advantage of in various situations, one of which is evolving genetic programs, which must use a mechanism such as parsimony pressure or must minimize a size-related objective value to prevent any individual from becoming too large. The bias provided by heterogeneous evaluation times can be used to produce an implicit time pressure; however, in cases with flat fitness landscapes, individuals tend to converge to both long and short evaluation times [8].

3 ASYNCHRONOUS PARALLEL CGP

Synchronous CGP, both serial and parallel, were implemented using the Standard CGP model, as defined by Miller [7], the only difference is that SPCGP evaluates all individuals of a generation simultaneously, while synchronous serial CGP evaluates only one individual at a time. SPCGP and APCGP both have a master node that generates new individuals that are later evaluated by slave nodes. SPCGP waits for all individuals in a generation to be returned, while APCGP acts on each individual as it is returned. In the case of APCGP, using the $(1 + 4)$ survival strategy advocated by Miller [7], the returned individual is compared to the existing best. If the new individual is better than or equal to the current best, it becomes the current best. Following this, a new individual is generated from the current best via mutation and the process continues until termination criteria are met. In this particular implementation, the evolutionary cycle terminates when the best individual has a fitness that exceeds a user-defined threshold. Although APCGP intuitively seems faster than SPCGP, the method by which APCGP explores the search space may lead to more evaluations until convergence. As seen in Figure 1, four individuals from the local search space of the current best individual are evaluated at each generation in SPCGP. In contrast to this, APCGP performs survival selection from only two individuals, and if a high-fitness solution has a long evaluation time, sub-optimal individuals will produce offspring to be evaluated while the high-fitness solution is being evaluated. An example of such an exploration in illustrated by Figure 2.

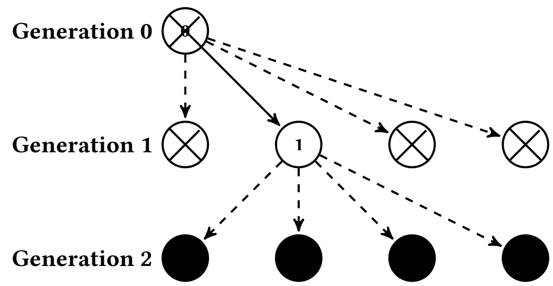
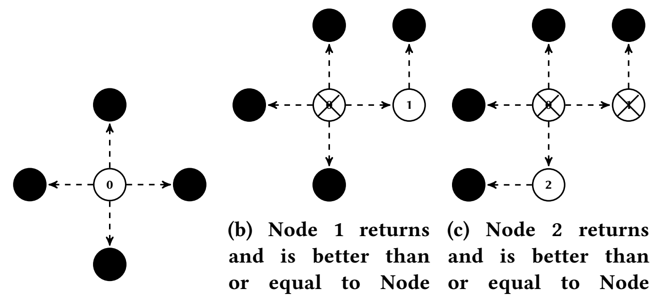


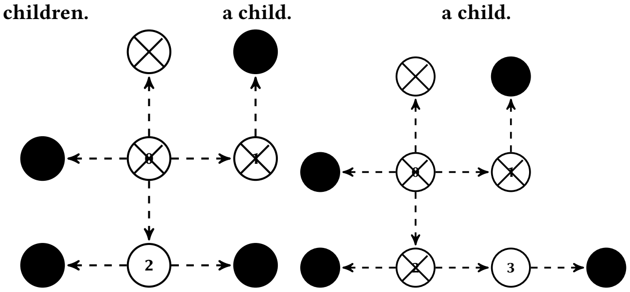
Figure 1: Exploration of search space in Synchronous CGP. The best individual of the parent and its four children is used for producing the next generation.



(a) The initial state – Node 0 produces four children.

(b) Node 1 returns and is better than or equal to Node 0, Node 1 replaces Node 0 and produces a child.

(c) Node 2 returns and is better than or equal to Node 1, Node 2 replaces Node 1 and produces a child.



(d) One of the children from Node 0 finishes evaluating. It is worse than Node 2, so it is discarded and Node 2 produces a child.

(e) Node 3 returns and is better than or equal to Node 2. Node 3 replaces Node 2 and Node 3 produces a child. Note that one of the children from Node 0 is still being evaluated.

Figure 2: Exploration of search space in Asynchronous Parallel CGP

4 EXPERIMENTATION

4.1 Problem

The problem chosen was n-bit parity, a classical digital circuit problem that CGP has been used to solve in the past [4]. This was chosen as it has a known solution, allowing termination once correct. Although more computationally complex problems would benefit more from parallelization, CGP suffers from high variation [4, 5], which becomes more pronounced as the problem complexity increases. Thus, to simulate more computationally complex problems and to reduce the effects of overhead due to parallelization, the fitness evaluation is configured to repeat any number of times.

4.2 Experiment Design

The experiment was run with the parameters shown in Table 1, as recommended by Miller [7]. n_i, the number of inputs, was equivalent to n for the n-bit parity problem trying to be solved (2 or 3). The function set was the bitwise functions {nand, and, nor, or} and thus the maximum parity of the functions, a, was two. The overhead, or the number of times the fitness evaluation was repeated, was varied between 1 and 400 to investigate performance based on

Parameter	Description	Value
n_c	Number of columns	4000
n_r	Number of rows	1
n_0	Number of outputs	1
l	Look back level	4000
μ	Population size	1
λ	Offspring size	4
μ_r	Mutation rate	0.01

Table 1: Parameters used for experimentation

problem complexity. 2-bit and 3-bit parity problems were run using a serial synchronous model, a parallel synchronous model, and an asynchronous parallel model. Each of these experiments was run thirty times. The parallel synchronous and parallel asynchronous models used a master/slave model, with one master thread and four slave threads. The implementation was done in Python, while parallel code was achieved using the multiprocess module.

5 RESULTS

As can be seen in Figure 3, asynchronous parallel and synchronous parallel models clearly have better run time averages than synchronous serial equivalent, while being close to each other in performance. The figure also indicates that asynchronous parallel takes more evaluations than synchronous parallel and synchronous serial, which are nearly identical in the regard. The statistical analysis of the results is shown in Table 2, indicating that there is statistical evidence that asynchronous parallel runs faster than synchronous parallel, while there does not seem to be strong statistical evidence that the number of evaluations differ.

	Time (seconds)		Evaluations	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	1387	2272	1598	1197
Standard Deviation	1140	1551	1394	740
Equal Variance Assumed?	**No**		**No**	
t Stat		-2.5182		1.3915
Two-tailed p-value		0.0148		0.1711

Table 2: Statistical analysis of 3-parity results with an overhead of 200

Using an overhead of 150, shown in Figure 4 with statistical analysis shown in Table 3, there is not strong statistical evidence that the runtime or the number of evaluations differ. When the overhead is lowered to 100, shown in Figure 5 with statistical analysis shown in Table 4, there is no statistical evidence that there is a

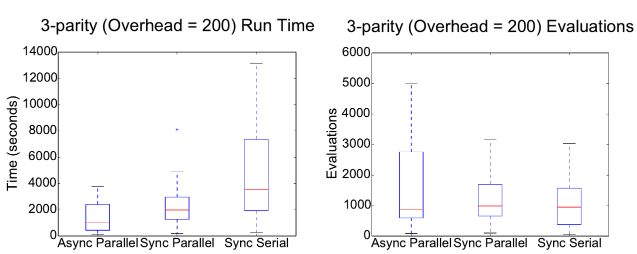


Figure 3: Results for 3-parity with an overhead of 200 (lower is better)

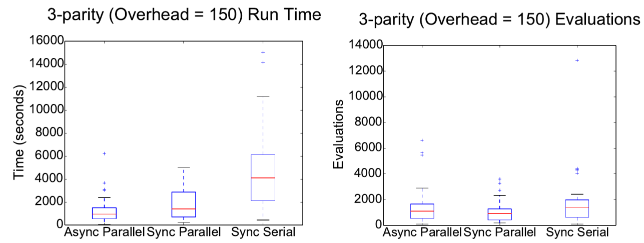


Figure 4: Results for 3-parity with an overhead of 150 (lower is better)

	Time (seconds)		Evaluations	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	1291	1843	1567	1107
Standard Deviation	1299	1404	1646	893
Equal Variance Assumed?	**No**		**No**	
t Stat		-1.5810		1.3445
Two-tailed p-value		0.1193		0.1856

Table 3: Statistical analysis of 3-parity results with an overhead of 150

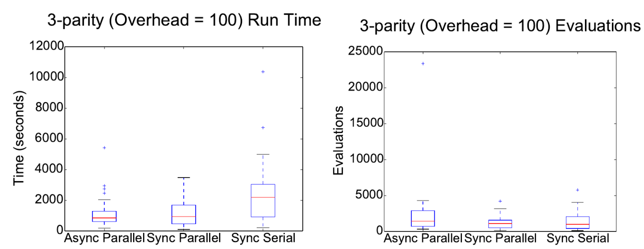


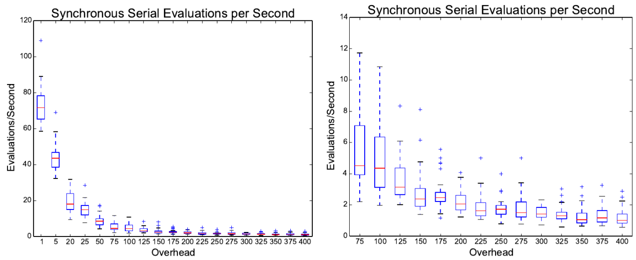
Figure 5: Results for 3-parity with an overhead of 100 (lower is better)

	Time (seconds)		Evaluations	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	1180	1217	2493	1224
Standard Deviation	1060	899	4113	959
Equal Variance Assumed?	**No**		**No**	
t Stat		-0.1439		1.6453
Two-tailed p-value		0.8861		0.1097

Table 4: Statistical analysis of 3-parity results with an overhead of 100

difference between the convergence time of APCGP and SPCGP, while there is still not strong statistical evidence that the number of evaluations differ.

As demonstrated in Figure 6, the synchronous serial model begins with a high evaluations/second rating, which quickly drops as the overhead increases. These results can be compared to those in Figure 7, asynchronous parallel and synchronous parallel both begin with lower evaluations/second, but the rate of decrease is substantially smaller in asynchronous parallel and synchronous parallel than in synchronous serial. Furthermore, as demonstrated by the statistical analysis with an overhead of 175, shown in Table 6,



(a) Overhead ranging from 1 to 400

(b) Overhead ranging from 75 to 400

Figure 6: Evaluations per second of synchronous serial with a variety of overheads for 2-bit parity (higher is better)

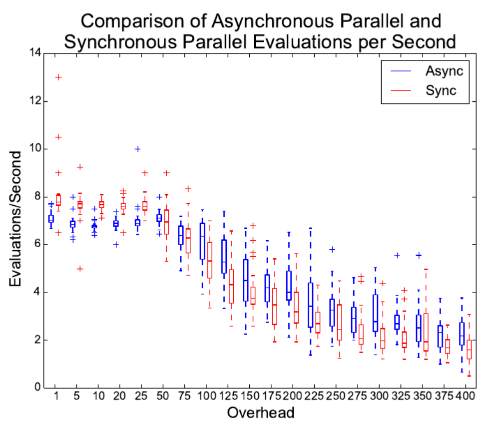


Figure 7: Evaluations per second of asynchronous parallel and synchronous parallel with a variety of overheads for 2-bit parity (higher is better)

	Time (seconds)		Evaluations	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	79	228	187	261
Standard Deviation	62	172	167	238
Equal Variance Assumed?	**No**		**No**	
t Stat		-4.4609		-1.4025
Two-tailed p-value		0.0001		0.1667

Table 5: Statistical analysis of 2-parity results with an overhead of 400

	Time (seconds)		Evaluations	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	57	156	241	393
Standard Deviation	46	176	193	440
Equal Variance Assumed?	**No**		**No**	
t Stat		-2.9725		-1.7320
Two-tailed p-value		0.0055		0.0910

Table 6: Statistical analysis of 2-parity results with an overhead of 175

there is statistical evidence that APCGP is faster than SPCGP. This evidence is only strengthened as the overhead increases, demonstrated by the statistical analysis with an overhead of 400, showing strong statistic evidence that ASCGP is faster than SPCGP.

6 CONCLUSION

This paper has presented statistical evidence showing that APCGP outperforms SPCGP for computationally expensive tasks, while both outperform synchronous serial CGP; we hypothesize that the former is caused by greater heterogeneity in evaluation times. If the task is computationally inexpensive, then APCGP and SPCGP perform similarly, but both are inferior to serial CGP. This provides evidence that parallelization should only be performed if the task is computationally expensive, and when performed, an asynchronous model should be preferred.

7 FUTURE WORK

More advanced versions of CGP exist which exhibit superior performance to standard CGP on various important problems; applying the asynchronous model to them may further increase their performance. Although CGP showed improved performance, there are many forms of GP; these forms may not show the same increase in performance when using the asynchronous model. Additionally, the asynchronous model could be applied to different types of EAs, such as co-evolutionary EAs or multi-objective EAs. Although this study used CGP's traditional (1 + 4) population model for parallel synchronous, changing the number of offspring could potentially result in further improvements over synchronous serial. In order to validate the hypothesis stated in the conclusion, that more computationally expensive tasks cause greater heterogeneity in evaluation times, the range of evaluation times should be diligently recorded and closely analyzed.

REFERENCES

[1] Bertels, A.R., Tauritz, D.R.: Why Asynchronous Parallel Evolution is the Future of Hyper-heuristics: A CDCL SAT Solver Case Study. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion. pp. 1359–1365. GECCO '16 Companion, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2908961.2931729
[2] Churchill, A.W., Husbands, P., Philippides, A.: Tool Sequence Optimization using Synchronous and Asynchronous Parallel Multi-Objective Evolutionary Algorithms with Heterogeneous Evaluations. In: 2013 IEEE Congress on Evolutionary Computation (CEC). pp. 2924–2931. IEEE (2013)
[3] Durillo, J.J., Nebro, A.J., Luna, F., Alba, E.: A Study of Master-Slave Approaches to Parallelize NSGA-II. In: IEEE International Symposium on Parallel and Distributed Processing. pp. 1–8. IEEE (2008)
[4] Goldman, B.W., Punch, W.F.: Analysis of Cartesian Genetic Programming's Evolutionary Mechanisms. IEEE Transactions on Evolutionary Computation 19(3), 359–373 (Jun 2015)
[5] Harding, S.L., Miller, J.F., Banzhaf, W.: Self-modifying Cartesian Genetic Programming. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation. pp. 1021–1028. GECCO '07, ACM, New York, NY, USA (2007), http://doi.acm.org/10.1145/1276958.1277161
[6] Martin, M.A., Bertels, A.R., Tauritz, D.R.: Asynchronous Parallel Evolutionary Algorithms: Leveraging Heterogeneous Fitness Evaluation Times for Scalability and Elitist Parsimony Pressure. In: Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation. pp. 1429–1430. GECCO Companion '15, ACM, New York, NY, USA (Jul 2015), http://doi.acm.org/10.1145/2739482.2764718
[7] Miller, J.: Cartesian Genetic Programming. Natural Computing Series, Springer-Verlag, Heidelberg, Berlin (2000)
[8] Scott, E.O., De Jong, K.A.: Evaluation-Time Bias in Asynchronous Evolutionary Algorithms. In: Proceedings of the Companion Publication of the 2015 on Genetic and Evolutionary Computation Conference. pp. 1209–1212. ACM, New York, NY, USA (Jul 2015)

[9] Scott, E.O., De Jong, K.A.: Evaluation-Time Bias in Quasi-Generational and Steady-State Asynchronous Evolutionary Algorithms. In: Proceedings of the Genetic and Evolutionary Computation Conference 2016. pp. 845–852. GECCO '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2908812.2908934
[10] Yagoubi, M., Schoenauer, M.: Asynchronous Master/Slave MOEAs and Heterogeneous Evaluation Costs. In: Proceedings of the Fourteenth International Conference on Genetic and Evolutionary Computation Conference. pp. 1007–1014. ACM (2012)
[11] Yagoubi, M., Thobois, L., Schoenauer, M.: Asynchronous Evolutionary Multi-Objective Algorithms with Heterogeneous Evaluation Costs. In: 2011 IEEE Congress on Evolutionary Computation (CEC). pp. 21–28. IEEE (2011)