SAND2016-3549C

Asynchronous Parallel Cartesian Genetic Programming

Adam Harter[1], Alex R. Bertels[1], Daniel R. Tauritz[1], and William M. Siever[2]

[1] Missouri University of Science and Technology, Rolla, MO, U.S.A.
[2] Western Illinois University, Macomb, IL, U.S.A

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 under the right circumstances can reduce the time incurred by fitness evaluation proportional to the number of parallel processing units. The most naïve approach maintains the synchrony of evolution as employed by the vast majority of EAs, but this requires an entire generation to be evaluated before progressing to the next generation; heterogeneity in the evaluation times will degrade the performance gain as parallel processing units will have to idle until the longest evaluation has completed. Asynchronous parallel evolution mitigates this bottleneck and techniques such as Cartesian GP (CGP) which experience high heterogeneity in evaluation times are therefore a prime candidate for asynchrony. 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.

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

1 Introduction

Cartesian Genetic Programming (CGP) arranges problem-specific operations as function nodes on a two-dimensional grid [6]. Dissimilar to the genotypes in most forms of GP, these grids remain a static size which 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. CGP evaluations can be performed in parallel like most traditional Evolutionary Algorithms (EAs) as the evaluations are independent of each other. 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 [7, 5]. 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 in regards to wall-time than synchronous parallel CGP (SPCGP)
- 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 [2]. 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.

Those that have specifically addressed heterogeneous populations, note that APEAs are biased toward individuals with shorter evaluation times [1, 8, 9, 5]. 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 [8]. 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 [7].

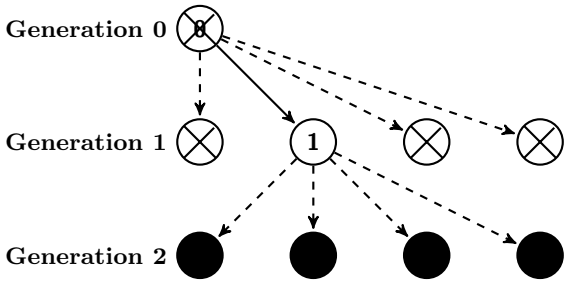
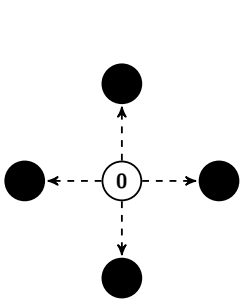
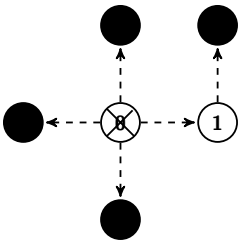


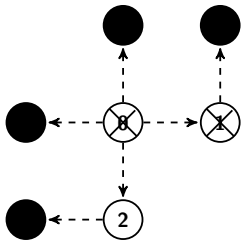
Fig. 1: Exploration of search space in Synchronous CGP. The best individual of the parent and its four children is used for producing the next generation.



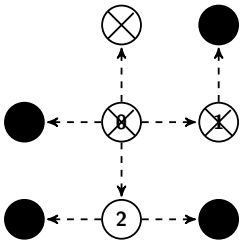
(a) The initial state – Node 0 produces four children.



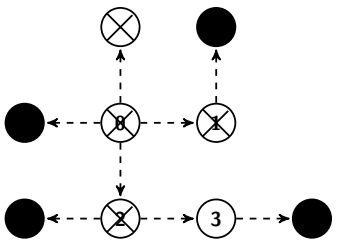
(b) Node 1 returns and is better than or equal to Node 0, Node 1 replaces Node 0 and produces a child.



(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.

Fig. 2: Exploration of search space in Asynchronous Parallel CGP

3 Asynchronous Parallel CGP

Synchronous CGP, both serial and parallel, were implemented using the Standard CGP model, as defined by Miller [6], 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 [6], 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.

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 [3]. This was chosen as it has a known solution, providing easy termination criteria. Although more computationally complex problems would benefit more from parallelization, CGP suffers from high variation [3, 4], 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 [6]. 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 {`nand`, `and`, `nor`, `or`} and thus the maximum parity, 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 problem complexity. 2-bit and 3-bit parity problems were

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

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, for an overhead of 200, the asynchronous parallel and synchronous parallel models have very similar run-time performance, but significantly better run time averages than the synchronous serial equivalent. 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 with an α of 0.05 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.

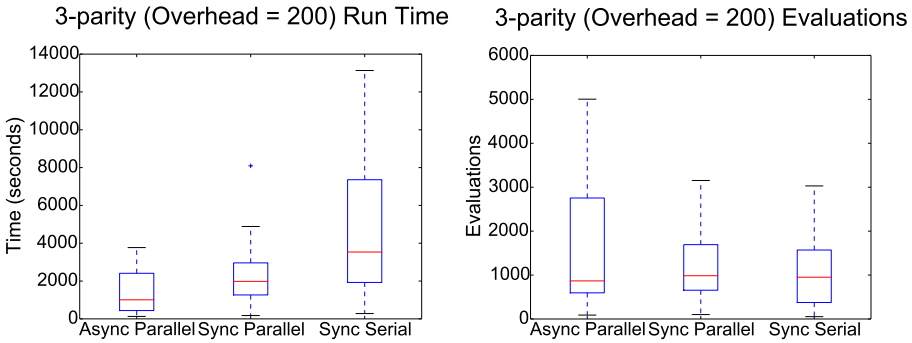


Fig. 3: Results for 3-parity with an overhead of 200 (lower is better)

As shown in Figure 4 and confirmed in Table 3, there is not strong statistical evidence that the runtime or the number of evaluations differ between APCGP

	Time (seconds)		Evaluations	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	1386.8667	2271.8667	1598.0667	1197.1333
Variance	1300173.1540	2405185.9126	1943444.8230	547167.4299
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

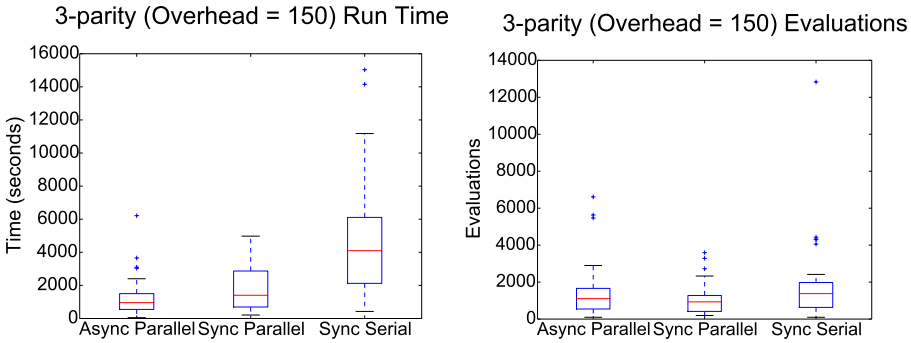


Fig. 4: Results for 3-parity with an overhead of 150 (lower is better)

	Time (seconds)		Evaluations	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	1291.3333	1843.3333	1566.6333	1107.0000
Variance	1686790.6437	1970160.6437	2708884.1713	797132.9655
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

and SPCGP when employing an overhead of 150. As may be expected, further lowering the overhead to 100, results in the same result that neither the runtime nor the number of evaluations differ between APCGP and SPCGP; this is shown in Figure 5 with statistical analysis shown in Table 4.

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 par-

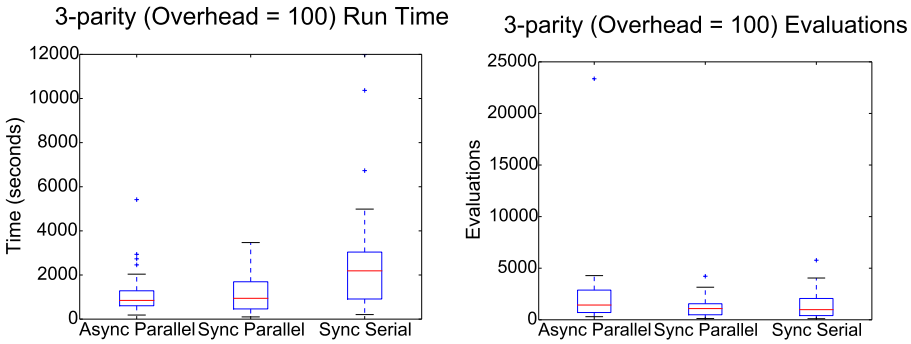
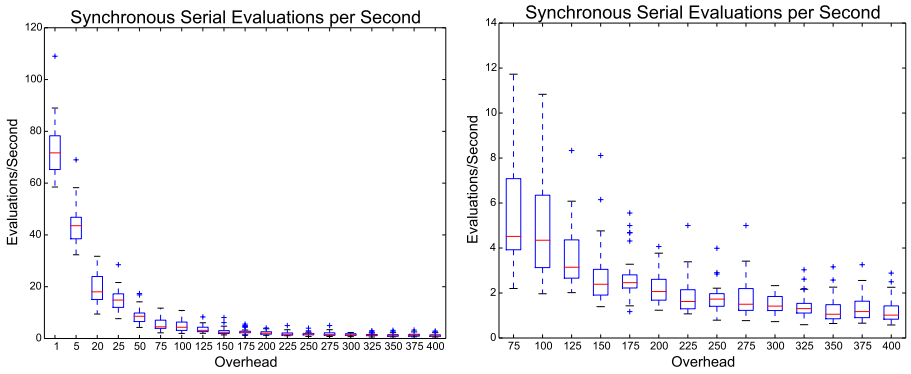


Fig. 5: Results for 3-parity with an overhead of 100 (lower is better)

	Time (seconds)		**Evaluations**	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	1180.1333	1216.6667	2492.8000	1224.0667
Variance	1124180.9471	808805.7471	16920736.5793	919078.2713
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



(a) Overhead ranging from 1 to 400 (b) Overhead ranging from 75 to 400

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

allel than in synchronous serial. Furthermore, as demonstrated by the statistical analysis with an overhead of 175, shown in Table 6, there is statistical evidence with an α of 0.05 that APCGP is faster than SPCGP. This evidence is only

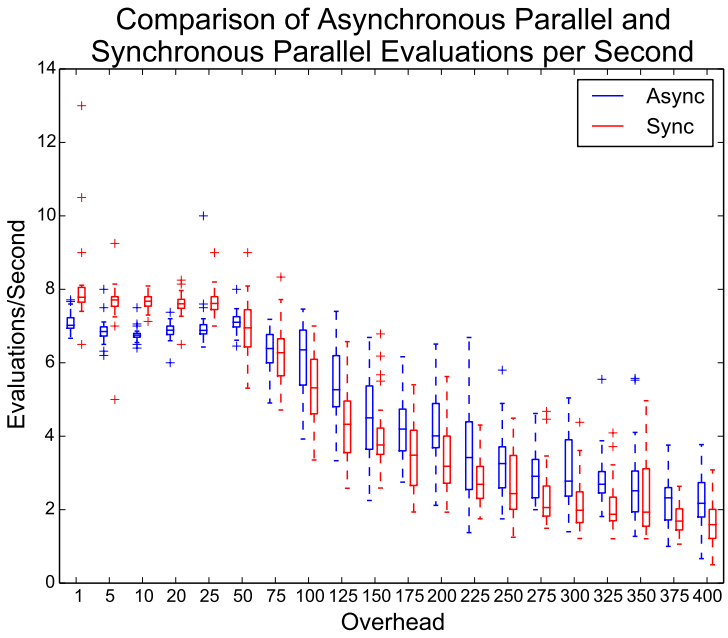


Fig. 7: Evaluations per second of asynchronous parallel and synchronous parallel with a variety of overheads for 2-bit parity (higher is better)

	Time (seconds)		**Evaluations**	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	79.2667	228.2667	186.7333	261.1333
Variance	3837.9954	29632.0644	27792.7540	56633.9126
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

strengthened as the overhead increases, demonstrated by the statistical analysis with an overhead of 400, showing almost statistic certainty that ASCGP is faster than SPCGP.

	Time (seconds)		Evaluations	
	Async Parallel	Sync Parallel	Async Parallel	Sync Parallel
Mean	57.4667	156.2333	240.9333	392.8667
Variance	2101.2230	31019.1506	37326.8920	193517.7747
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

6 Conclusion

Statistical evidence has been presented showing that APCGP outperforms SPCGP for computationally expensive tasks, while both outperform synchronous serial CGP. 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 heavy, and when performed, an asynchronous model should be preferred.

7 Future Work

More advanced versions of CGP exist, which often have superior performance to standard CGP; 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 EA's, such as co-evolutionary EAs or multi-objective EAs. Although this study used the traditional (1 + 4) population model for parallel synchronous, changing the number of offspring could potentially result in further improvements over synchronous serial.

References

1. 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)
2. 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)
3. 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)

4. 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
5. 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
6. Miller, J.: Cartesian Genetic Programming. Natural Computing Series, Springer-Verlag, Heidelberg, Berlin (2000)
7. 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)
8. 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)
9. 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)