TELKOMNIKA Telecommunication, Computing, Electronics and Control
Vol. 19, No. 1, February 2021, pp. 182∼191
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.v19i1.15026

Fine-grained or coarse-grained? Strategies for implementing parallel genetic algorithms in a programmable neuromorphic platform

Indar Sugiarto[1], **Steve Furber**[2]
[1]Department of Electrical Engineering, Petra Christian University, Indonesia
[2]School of Computer Science, University of Manchester, United Kingdom

Article Info

Article history:

Received Dec 19, 2019
Revised Apr 10, 2020
Accepted Sep 24, 2020

Keywords:

Genetic algorithm
Network on chip
Neuromorphic computing
Parallel computing
SpiNNaker

ABSTRACT

Genetic algorithm (GA) is one of popular heuristic-based optimization methods that attracts engineers and scientists for many years. With the advancement of multi- and many-core technologies, GAs are transformed into more powerful tools by parallelising their core processes. This paper describes a feasibility study of implementing parallel GAs (pGAs) on a SpiNNaker. As a many-core neuromorphic platform, SpiNNaker offers a possibility to scale-up a parallelised algorithm, such as a pGA, whilst offering low power consumption on its processing and communication overhead. However, due to its small packets distribution mechanism and constrained processing resources, parallelising processes of a GA in SpiNNaker is challenging. In this paper we show how a pGA can be implemented on SpiNNaker and analyse its performance. Due to inherently numerous parameter and classification of pGAs, we evaluate only the most common aspects of a pGA and use some artificial benchmarking test functions. The experiments produced some promising results that may lead to further developments of massively parallel GAs on SpiNNaker.

Corresponding Author:

Indar Sugiarto
Department of Electrical Engineering
Petra Christian University
Jl. Siwalankerto 121-131, Surabaya, 60236, Indonesia
Email: indi@petra.ac.id

1. INTRODUCTION

The advent of multi- and many-core platform makes the parallel and distributed computing more feasible and attractive. The same is also a valid case for algorithms that are inherently feasible for parallelism such as a genetic algorithm (GA). However, the real implementation of parallelism for such algorithms may be challenged by the fact that many current computing platforms inherent serial processing mechanism of von-Neumann architecture [1, 2]. This makes the implementation of parallel information processing complicated and expensive [3, 4]. In the case of GAs, the parallelism concept has been applied in recent years in order to achieve a higher performance throughput. One common method to achieve such acceleration is by utilizing accelerator such as graphics processing units (GPUs) [5–8], it is well-known that single instruction multiple data (SIMD) mechanism in a GPU plays an important role to achieve such very high computation throughput [9, 10].

GA belongs to the larger class of evolutionary algorithms (EA) and inherently works as a sequential process [11, 12]. Even though the inner processes in a GA are sequential, the data on which the processes take place are partitionable. This allows a GA to be parallelized [13]. A parallel GA (pGA) usually exploits the abundant processing elements in a parallel/distributed system, and can be roughly classified into two categories: coarse-grained parallel GA (cpGA) and fine-grained parallel GA (fpGA). The most distinctive features of the two can be inferred from the GA's population granularity and the communication overhead. The most popular example of the cpGA is the island model, whereas in fpGA cellular GA is the most common model.

In this paper we present our preliminary study of pGAs implementation on a novel many-core neuromorphic platform called SpiNNaker. SpiNNaker (spiking neural network architecture) is a many-core neuromorphic platform that is designed to emulate brain's processing and connectivity in neuronal level [14–16]. Due to enormous features of pGAs [17, 18], we cannot explore and present all features of both pGA categories in this paper. Thus, this paper serves as the starting point for future exploration of pGAs in SpiNNaker. To our knowledge, this is the first paper that describes methods for implementing pGAs on a neuromorphic platform. Thus, benchmarking the performance evaluation of pGAs on SpiNNaker with other available neuromorphic platforms is not possible. Rather, the emphasis is on testing the SpiNNaker platform as a viable device to implement a massively parallel GA. The results of this paper can be used for future exploration on a collaboration between extensive optimization and deep learning [19, 20]. The paper is organized as follows. After the introduction in this section, related works on pGAs and SpiNNaker for conventional non-spiking applications are described. In section 3, the detailed implementation of pGAs on SpiNNaker is described. Section 4 presents the experiment setup and the collected results, followed by their analysis and discussion. Finally, this paper is closed by conclusions in section 5.

2. RELATED WORKS

GA is a nature (biology) inspired algorithm mostly popular to be used to solve complex optimization problems [21, 22]. It is a metaheuristic method inspired by the process of natural selection that involves bio-inspired operations such as gene mutation and crossover. There are different families of GAs, and in the recent decade, they have been showing to converge [23, 24]. Conceptually, inner working principles in a GA are sequential processes. To speed them up, parallelism strategies are applied on those processes. At some extent, the parallelism also involves distributed processes. In this paper, we refer a parallel distributed GA to as simply a parallel GA. This is conceptually a straightforward method, since GA normally works on regular data in a partitioned solution space. However, in real implementation, there is a complication issue due to various options that could be traded off in order to match between the hardware platform and the GA-encoded problem description.

In general, a parallel process/algorithm can work in a fine-grained or coarse-grained fashion. A fine-grained parallel GA (fpGA) commonly takes a form in a cellular model; whereas a coarse-grained parallel GA (cpGA) utilizes a concept of group separation, such as in island models. The granularity of pGA can also be described in a finer continuum, creating a broader range of classification; in this paradigm, any new pGA model can fall in between fine- and coarse-grained classes. Furthermore, they can also be stacked hierarchically for creating a hybrid class from any possible combination of existing models.

Comprehensive lists of pGA models have been presented in various recent review papers, for example [25–30]. The morphology, genetic operation, and communication principle between elements on those various pGA models have also been described on those papers. Hence, we will not cover detailed aspects of a pGA in this paper. Instead, we present a general concept of pGAs only for these simplified classes: fine- and coarse-grained models. Fine-grained pGAs are strongly associated with the massively parallel machines in the form of multi-core computers. On the other hand, coarse-grained pGAs are commonly implemented in a distributed platform and closely related to the concept of migration islands. Due to numerous tuning properties of a pGA as well as a tight coupling between pGA design and the problem it tries to solve, it is almost impossible to create a generic pGA framework that unifies both classes. Fortunately, the SpiNNaker platform is generic enough to be used as both parallel and distributed computing. In this paper, we use some parallelism strategies and tools developed in our previous work [31].

3. RESEARCH METHOD

3.1. Brief overview of SpiNNaker

SpiNNaker (spiking neural network architecture) is a neuromorphic platform originally designed to simulate brain processing in a biologically plausible mechanism [32]. On hardware level, SpiNNaker is a field programmable neuromorphic platform that is built on many-core concept using massively connected multi-core chips [33]. SpiNNaker working mechanism are based on small packet transaction between nodes/chips. These packets are very fast to be distributed intra- and inter-chips to mimic biological realm of the brain where many small spikes are interchanged among neurons in the brain. This way, the SpiNNaker allows brain-like parallel, asynchronous computation, and in real time with time-steps at about $\sim$ 1 ms. In terms of power efficiency, SpiNNaker has demonstrated superiority over supercomputing-clusters when simulating large-scale biological networks [34]. Currently, the full SpiNNaker machine construction is underway and it will host 50K SpiNNaker chips; hence, it will have one million processors with 7 terabytes of memory running in parallel. It is expected that the machine will consume at most 90kW of electrical power [35]. Figure 1 shows the structure of the SpiNNaker machine.

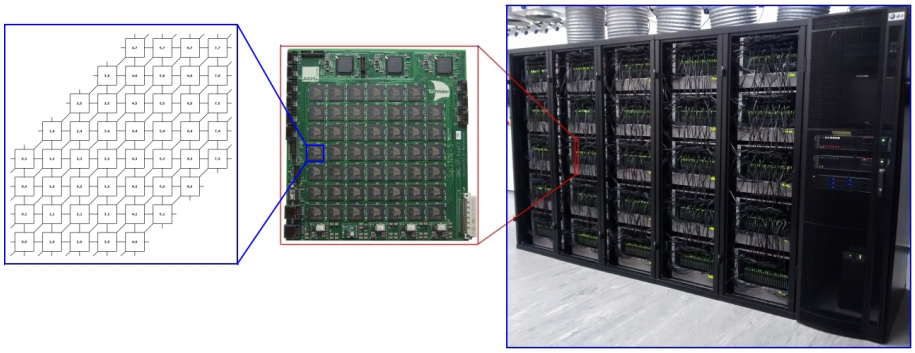

Figure 1. The SpiNNaker machine is composed of many SpiNNaker chips assembled on a board. Each SpiNNaker board contains 48 chips, arranged in 2D triangular mesh

3.2. fpGA implementation

3.2.1. Basic cellular model

An fpGA is implemented usually as a cellular model. In this model, population is spatially distributed across a grid population in which genetic interactions may only take place in a small neighborhood of each individual. In a basic cellular model, a node/processor is responsible for processing only one (or very limited number) GA chromosome and performs the GA operations in a reproductive cycle together or after interaction with its adjacent neighbor. In this model, a 2D grid of processors is used where the neighborhood is defined using the Manhattan distance from one point to others in the grid.

Figure 2 shows examples of fpGA networks. The neighborhood refers to the set of potential mates of an individual. Consequently, each node in the grid has a neighborhood that overlaps the neighborhoods of nearby nodes. The geometrical simplification is used in the sense that all the neighborhoods have the same size and identical shapes. There are various configurations in which a cellular pGA can be formed. The two most commonly used are L5, also called Von Neumann or NEWS (North, East, West and South), and C9, also known as Moore neighborhood.

3.2.2. Realisation in SpiNNaker

Implementing the pGA basic cellular model on SpiNNaker is straight forward. However, there are several considerations for implementation. First, SpiNNaker uses triangular mesh layout, whereas the common pGA cellular models use rectangular mesh topology. In our work, we create a modification to the SpiNNaker routing such that only four links are used to mimic the standard rectangular cellular model: north, west, south and east links are used for pGA interaction. However, we keep the other two links (northeast and southwest) alive for the purpose of fault tolerance; i.e., they only be used as backups if the main pGA links fail to work, either permanently or intermittently.

Second, the SpiNNaker machine can be used per board basis or per group of board basis. In order to create a continuous torus, at least three SpiNNaker boards are needed. In this three board setup, a closed grid of 12x12 nodes can be accomplished and a normal cellular pGA model can operate properly. In some circumstances, however, a square grid cannot be closed since the torus is discontinued; for example, when only one SpiNNaker board is used. For this particular situation, we define a maximum neighborhood distance as the maximum squared odd-number of chips available on the system. In this paper, we use three board system but they are not connected in a torus. The grid structure of this three board system is shown in Figure 3. Here we use 8x15 grid for the model. In such a configuration, the maximum neighborhood distance allows to create either L13 or C49 model.

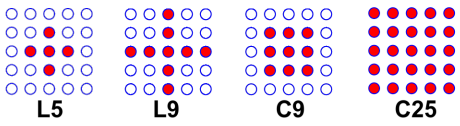

Figure 2. Example of fpGAs as cellular models. Colored nodes represent a neighborhood. Here, L stands for linear while C stands for Compact. It is possible to arrange nodes in the grid differently, such as using a diamond-like layout

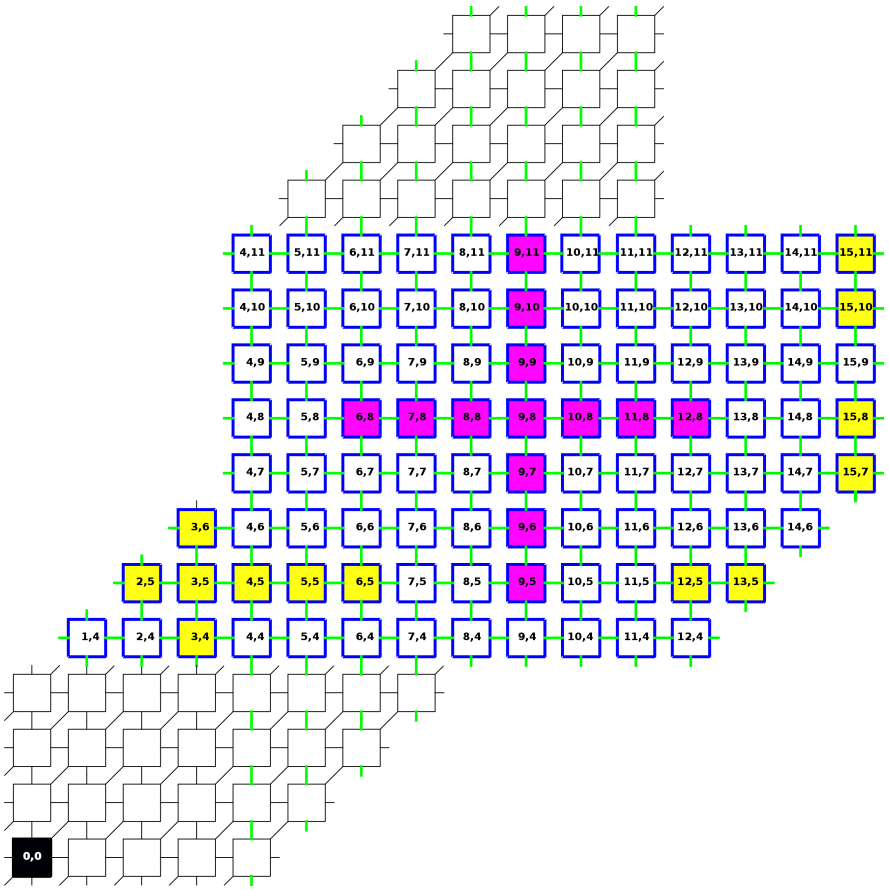

Figure 3. The proposed layout for a pGA cellular model on a three SpiNNaker board

Our method for implementing fpGA based on the idea of run-time improvement; here, the multi-core feature of SpiNNaker chips provides a redundancy that can be exploit for amelioration. Here, we propose to use the similar approach presented in our previous work (see [36]), where we use a spawning task mechanism on several cores for improving performance through local parallelism (i.e., those cores will be loaded with the

same GA code and work in synchrony) and the rest of the cores will serve as the backup cores for providing higher reliability through a fault tolerance mechanism.

As a summary to our implementation strategy, Figure 3 shows the proposed layout to implement an fpGA in a cellular model on SpiNNaker. The torus is created on the 8x15 grid, in which each of the blue-colored nodes processes one GA chromosome/individual. In the diagram, just two GA chromosomes are depicted to show how the neighborhood in L13 is formed: one with purple color, and the other with yellow color. The number inside the square indicates the coordinate of the node used for routing by the SpiNNaker routing mechanism.

3.3. cpGA implementation

3.3.1. Basic island model

In contrast to the fpGA models, cpGA models try to achieve higher computation performance whilst reducing the communication overhead. In this paradigm, the proponent of these models argue that it is better to localise the intense GA operations into a group of processing units. In reality, this concept is commonly implemented as a network of processing islands; hence comes the name of island model. By distributing and localising the GA processors, the solution space is expected to be partitioned and each island can specialize itself to a certain solution region. In order to improve the convergence, an additional operation called migration is introduced in these models. The conceptual island model is presented in Figure 4.

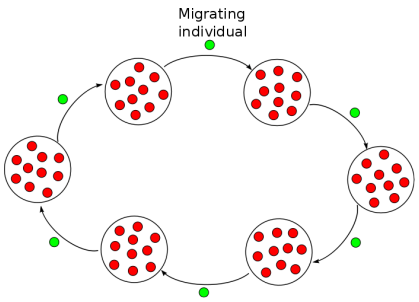

Figure 4. Example of cpGA in an island model

3.3.2. Realisation in SpiNNaker

In contrast to the approach for implementing fgGA on SpiNNaker, implementing island models on SpiNNaker is not straight forward. There are some constraints that need to be considered. For example, how an island is defined in terms of hardware resources of SpiNNaker; i.e., how many cores or chips are used to contain an island? This corresponds to the common problem in this island model: specifying/tuning the island size and numbers. This is closely related with the mapping strategy that will be employed by the SpiNNaker tool-chain.

There are two options for this: static mapping and dynamic mapping. In static mapping, the number of cores or chips that will contain an island and the routing table for those cores/chips are predefined in compile-time; i.e., the designer must specify them explicitly (hardcoded) in the SpiNNaker program code. The second option involves a dynamic mapping approach in the sense that the number of cores/chips along with the routing among them can be obtained automatically, even though not during run-time. In this paper, we use the static mapping approach since the problem formulated for our evaluation is small. Regarding the island implementation, all cores in a chip are orchestrated to handle one population that consists of 120 individual. Since there are only 16 cores available for computation in each chip, each core is responsible for handling up to 8 individuals (some of them handle just 7 individual). In this paper, we design an island to be contained in a SpiNNaker chip.

4. RESULTS AND ANALYSIS

4.1. Experiments setup

For evaluating the performance of pGA on SpiNNaker, we use two standard benchmarking functions: Rastrigin and the second Schaffer function. They are expressed in the following formula:

$$f(x_1 \cdots x_n) = 10n + \sum_{i=1}^{n} (x_i^2 - 10cos(2\pi x_i)) \quad (1)$$

where

$$-5.12 \leq x_i \leq 5.12$$

$$f(x,y) = 0.5 + \frac{sin^2(\sqrt{x^2+y^2}) - 0.5}{[1 + 0.001 \cdot (x^2+y^2)]^2} \quad (2)$$

where

$$-100 \leq x_i \leq 100$$

Both functions have minimum value at (0,0). The Rastrigin function in (1) is an example of non-convex and non-linear multimodal function. Finding the minimum of this function is a fairly difficult problem due to its large search space and its large number of local minima. The second Schaffer function in (2) is also non-convex and non-linear multimodal function, but it has a broader search space than the Rastrigin function. It is used here along with the Rastrigin function so that we can evaluate the extensibility of our pGA framework for solving a similar but bigger solution space.

In this paper we use only three SpiNNaker boards. For the fpGA models, we use L13 and C49 model as described in section 3.2.2.. For the cpGA models, we use varied number of islands from 2 to 21. An example of cpGA model implementation with 21 islands is depicted in Figure 5. Note that the position of islands do not give any effect on the pGA performance, but only on the creation of the routing table for SpiNNaker packets. For the sake of simplicity, the mapping was static one as described in section 3.3.2.; hence, the routing for SpiNNaker packets were defined manually. If the islands position in Figure 5 are changed, then a new routing table must be defined. We are aware of this limitation, and we plan to use a dynamic mapping in our future work. To make a fair comparison between fpGA and cpGA, we use the same GA parameters as shown in Table 1. Regarding the population generation in fpGAs, there are two mechanism that can be employed: synchronous and asynchronous. In this paper, we use the synchronous approach as the parallel update policy, in which all nodes are updated simultaneously in each operation.

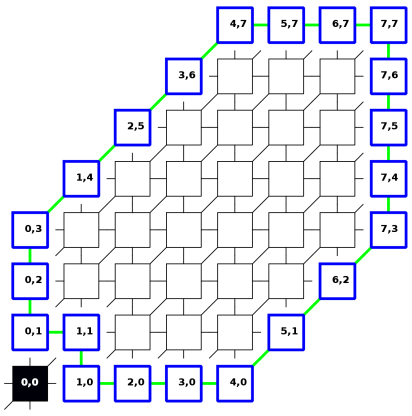

Figure 5. Example of an fpGA model with 21 islands using only a single SpiNNaker board. Here the green-colored links are predefined as the route for migration

Table 1. Common GA parameters used for both fpGA and cpGA in the experiments

Parameter	Value
GA encoding	binary 16-bit
Elitism	none
Crossover Type	single point
Crossover Rate	0.1
Mutation Rate	0.02
Migration Rate	0.05
Population Size	120

4.2. Experiment results

We run the fpGA as a cellular model for the Rastrigin and Schaffer functions on SpiNNaker using L13 and C49 configurations. The network was executed for maximum 100 iterations, which corresponds to generating 100 generations in the GA process. The result is shown in Figure 6. Just for convenience, the plot

shows the objective value instead of fitness value; plotting the fitness value will show the similar pattern. We also run the cpGA version using an island model with various number of islands. We use the model to find the global minima for both the Rastrigin and the second Schaffer test functions. Similar to the previous experiment on fpGA, we run the model for 100 generations. The result is shown in Figure 7 (a) for Rastrigin, and Figure 7 (b) for the Schaffer function.

4.3. Evaluation and discussion

We have run initial experiments using two standard optimization test functions and we found that our implementation of both fpGA and cpGA produced satisfying results. Both models found the global optimum very quickly, which is measured as number of generations needed to achieve the convergence result. Particular evaluations of each model are presented as follows.

4.3.1. Performance on fpGA models

We use two cellular models for implementing fpGA on SpiNNaker: L13 and C49. In both cases, multi-core feature of SpiNNaker chips are employed for amelioration and reliability purposes; hence, the number of individual in the population is the same as the size of the grid, which is 120. As we can see in Figure 6, both L13 and C49 models work properly and quite fast to converge into the correct solution. In general, we can observe that solving the second Schaffer function is a bit faster than the Rastrigin function. This is interesting because the Schaffer function has a larger search space than the Rastrigin function; one would expect that the searching for the solution should be longer. However, Rastrigin function has much more local minima than the Schaffer function. We argue that this explains the convergence speed behaviour in our models.

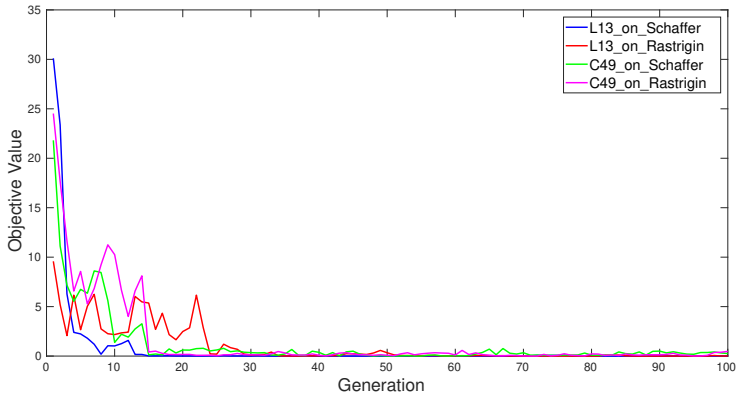

Figure 6. The result of running cellular models using L13 and C49 configurations to find the global minima for Rastrigin and Schaffer functions

Also, we notice that C49 model runs in a more steady fashion than the L13 model. As we can see from the plot, C49 model needs almost the same number of generations to converge for both test functions, which happens at 15^{th} generation. However for L13 model, even though it reach the convergence for the Schaffer function at generation 13^{th}, the convergence for the Rastrigin function starts imminently after the 25^{th} generation. We admit that we could not give further insight why this happens, and we plan to carry further evaluation using some more complex functions in our future work.

4.3.2. Performance on cpGA models

The similar test mechanism was also applied for our island model as a representation of cpGA class. Here, we use the number of islands as the indicator for performance. As we can see in Figure 7, higher number of islands means higher convergence rate. In both test cases, the convergence has been achieved before the 10^{th} generation when using 20 islands (in the Schaffer test, the convergence starts at the sixth generation). As in the fpGA case, we also notice that the network performance for solving the Schaffer test problem is a bit faster than for the Rastrigin test function.

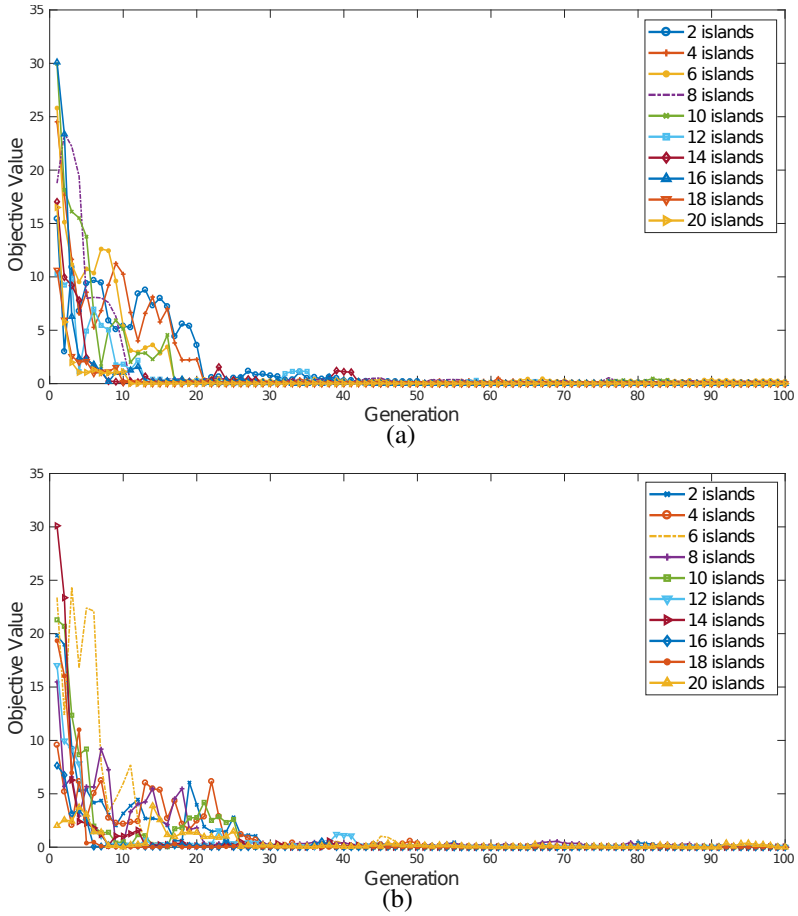

Figure 7. cpGA results for: (a) rastrigin, and (b) schaffer function

4.3.3. Further improvement ideas

Comparing the island model and the cellular model, our experiments show that the island model converges faster than the cellular model at higher number of islands. However, when the number of islands is small, the island model does not show better performance than the cellular model. We argue that increasing the number of islands means that the number of individuals searching for the solution is also increased. We also observe that in the cellular models, similar individuals tend to cluster creating niches, and these groups operate as if they were separate sub-populations similar to islands in the island model. For practical implementation consideration, we show here two observations:

1. For solving small problems, in general, implementing the "constrained" island models on SpiNNaker is easier than the cellular models. In this case, the island models require less SpiNNaker resources in terms of the number of chips and the packet routing mechanism. However, this does not scale-up with problem size.
2. For solving bigger problems which require longer GA strings for encoding, the cellular models are preferred because we can achieve higher computation performance by applying local parallelism inside the chip. This cannot be achieved when using island models, because all cores in the chip have been used already and leave no space for performance improvement.

There are several other features that still need to be explored in future. For example, in the fpGA models, what if the neighborhood takes non-canonical forms as in a diamond or even in a trapezoid shape? Also, how the synchronous and asynchronous updates affected the performance on the models? For the cpGA models, there are also many features that need to be explored. For example, what are the effects of different migration rates as well as the direction of migration? Also, how to merge islands and what is the effect of dynamic immigrant size? Nevertheless, this paper serves as an initial starting point to explore the vast potential of pGA applications on the SpiNNaker platform.

5. CONCLUSIONS

This paper presents a preliminary work on the feasibility for implementing parallel genetic algorithms (pGA) with some degrees of flexibility on a neuromorphic platform. SpiNNaker, a many-core programmable neuromorphic platform, is used to implement two classes of pGA: fine-grained pGA (fpGA) and coarse-grained pGA (cpGA). Due to numerous variability of these models, not all possible configurations are presented. However, the experiment results show the promising flexibility that potentially can be used to implement any model configuration from both fgGA and cpGA domains. Some considerations are presented regarding the expected performance of the implemented pGA with regard to hardware constraint. The trade-off between performance and scalability is also presented. We argue that once the hurdle with the mapping complexity has been overcome, SpiNNaker will become a superior platform for implementing a massive parallel genetic algorithm framework to solve various multi-objective optimisation problems. This will be a precious added value for SpiNNaker as a many-core neuromorphic platform, not only for performing brain-style information processing, but also as a generic high performance computing system.

ACKNOWLEDGMENT

This work was supported partially by the United Kingdom Engineering and Physical Sciences Research Council (EPSRC) through the Graceful project (grant EP/L000563/1). The design and construction of the SpiNNaker machine was supported by EPSRC (the UK Engineering and Physical Sciences Research Council) under grant nos. EP/D07908X/1 and EP/G015740/1, in collaboration with the universities of Southampton, Cambridge and Sheffield and with industry partners ARM Ltd, Silistix Ltd and Thales. Ongoing development of the software is supported by the EU ICT Flagship Human Brain Project (H2020 785907 and 945539), in collaboration with many university and industry partners across the EU and beyond. The development of the pGA test cases on SpiNNaker was supported by Petra Christian University through the Special Grant (grant 009/SP2H/LT-MULTI/LPPM-UKP/III/2020 and 016/SP2H/LT-MULTI/LPPM-UKP/III/2020).

REFERENCES

[1] A. O. Moraes, J. F. Mitre, P. L. Lage, and A. R. Secchi, "A robust parallel algorithm of the particle swarm optimization method for large dimensional engineering problems," *Applied Mathematical Modelling*, vol. 39, no. 14, pp. 4223–4241, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0307904X14007094
[2] J. Domínguez and E. Alba, "Dealing with hardware heterogeneity: a new parallel search model," *Natural Computing*, vol. 12, no. 2, pp. 179–193, 2013.
[3] K. Asanovic, R. Bodik, J. Demmel, T. Keaveny, K. Keutzer, J. Kubiatowicz, N. Morgan, D. Patterson, K. Sen, J. Wawrzynek, D. Wessel, and K. Yelick, "A view of the parallel computing landscape," *Communication of the ACM*, vol. 52, no. 10, pp. 56–67, 2009. [Online]. Available: http://doi.acm.org/10.1145/1562764.1562783
[4] D. Lim, Y.-S. Ong, Y. Jin, B. Sendhoff, and B.-S. Lee, "Efficient hierarchical parallel genetic algorithms using grid computing," *Future Generation Computer Systems*, vol. 23, no. 4, pp. 658–670, 2007.
[5] J. Luo, S. Fujimura, D. El Baz, and B. Plazolles, "GPU based parallel genetic algorithm for solving an energy efficient dynamic flexible flow shop scheduling problem," *Journal of Parallel and Distributed Computing*, vol. 133, pp. 244–257, 2019. [Online]. Available: http://doi.org/10.1016/j.jpdc.2018.07.022
[6] N. Hou, F. He, Y. Zhou, Y. Chen, and X. Yan, "A parallel genetic algorithm with dispersion correction for hw/sw partitioning on multi-core cpu and many-core GPU," *IEEE Access*, vol. 6, pp. 883–898, 2018.
[7] S. Kang, S.-S. Kim, J. Won, and Y. M. Kang, "GPU-based parallel genetic approach to large-scale travelling salesman problem," *The Journal of Supercomputing*, vol. 72, no. 11, pp. 4399–4414, 2016.
[8] A. Ferigo and G. Iacca, "A gpu-enabled compact genetic algorithm for very large-scale optimization problems," *Mathematics*, vol. 8, no. 5, pp. 758 (1–26), 2020.
[9] B. Plazolles, D. El Baz, M. Spel, V. Rivola, and P. Gegout, "SIMD monte-carlo numerical simulations accelerated on GPU and xeon phi," *International Journal of Parallel Programming*, vol. 46, no. 3, pp. 584–606, 2018.
[10] B. Ren, S. Balakrishna, Y. Jo, S. Krishnamoorthy, K. Agrawal, and M. Kulkarni, "Extracting SIMD parallelism from recursive task-parallel programs," *ACM Trans. Parallel Comput.*, vol. 6, no. 4, pp. 1–37, Dec. 2019.
[11] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing, $2^{nd}ed.$ $Springer$*, 2015.
[12] J. Pérez Heredia, "A computational view on natural evolution: On the rigorous analysis of the speed of adaptation," Ph.D. dissertation, University of Sheffield, 2017.
[13] Y. Y. Liu and S. Wang, "A scalable parallel genetic algorithm for the generalized assignment problem," *Parallel computing*, vol. 46, pp. 98–119, 2015.
[14] S. Furber, "Large-scale neuromorphic computing systems," *Journal of neural engineering*, vol. 13, no. 5, pp. 1–14, 2016.
[15] M. Davies, N. Srinivasa, T. Lin, G. Chinya, Y. Cao, S. H. Choday, and *et al.*, "Loihi: A neuromorphic manycore processor with on-chip learning," *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
[16] Y. van De Burgt, A. Melianas, S. T. Keene, G. Malliaras, and A. Salleo, "Organic electronics for neuromorphic computing," *Nature Electronics*, vol. 1, no. 7, pp. 386–397, 2018.
[17] Y. Y. Wong, K. H. Lee, K. S. Leung, and C.-W. Ho, "A novel approach in parameter adaptation and diversity maintenance for genetic algorithms," *Soft Computing*, vol. 7, no. 8, pp. 506–515, 2003.

[18] G. Karafotias, M. Hoogendoorn, and A. E. Eiben, "Parameter control in evolutionary algorithms: Trends and challenges," *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 2, pp. 167–187, 2015.
[19] F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune, "Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning," *CoRR*, vol. abs/1712.06567, 2017. [Online]. Available: http://arxiv.org/abs/1712.06567
[20] A. AbuZekry, I. Sobh, E.-S. Hemayed, M. Hadhoud, and M. Fayek, "Comparative study of neuro-evolution algorithms in reinforcement learning for self-driving cars," *European Journal of Engineering Science and Technology*, vol. 2, no. 4, pp. 60–71, 2019.
[21] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, no. 2, pp. 65–85, 1994.
[22] M. Srinivas and L. M. Patnaik, "Genetic algorithms: a survey," *Computer*, vol. 27, no. 6, pp. 17–26, June 1994.
[23] E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, Oct 2002.
[24] J. R. Cheng and M. Gen, "Accelerating genetic algorithms with GPU computing: A selective overview," *Computers & Industrial Engineering*, vol. 128, pp. 514–525, 2019.
[25] E. Cantú-Paz, "A survey of parallel genetic algorithms," *Calculateurs paralleles, reseaux et systems repartis*, vol. 10, no. 2, pp. 141–171, 1998.
[26] E. Alba and J. M. Troya, "A survey of parallel distributed genetic algorithms," *Complexity*, vol. 4, no. 4, pp. 31–52, 1999.
[27] D. Knysh and V. Kureichik, "Parallel genetic algorithms: a survey and problem state of the art," *Journal of Computer and Systems Sciences International*, vol. 49, no. 4, pp. 579–589, 2010.
[28] B. Johnson and Y. Qu, "Towards an autonomously configured parallel genetic algorithm: Deme size and deme number," in *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*. IEEE, 2016, pp. 278–288.
[29] H. Ma, S. Shen, M. Yu, Z. Yang, M. Fei, and H. Zhou, "Multi-population techniques in nature inspired optimization algorithms: A comprehensive survey," *Swarm and Evolutionary Computation*, vol. 44, pp. 365–387, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2210650217306363
[30] Y.-J. Gong, W.-N. Chen, Z.-H. Zhan, J. Zhang, Y. Li, Q. Zhang, and J.-J. Li, "Distributed evolutionary algorithms and their models: A survey of the state-of-the-art," *Applied Soft Computing*, vol. 34, pp. 286–300, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1568494615002987
[31] I. Sugiarto, G. Liu, S. Davidson, L. A. Plana, and S. B. Furber, "High performance computing on SpiNNaker neuromorphic platform: A case study for energy efficient image processing," in *2016 IEEE 35th International Performance Computing and Communications Conference (IPCCC)*. IEEE, 2016, pp. 1–8.
[32] B. Sen-Bhattacharya, S. James, O. Rhodes, I. Sugiarto, A. Rowley, A. B. Stokes, K. Gurney, and S. B. Furber, "Building a spiking neural network model of the basal ganglia on SpiNNaker," *IEEE Transactions on Cognitive and Developmental Systems*, vol. 10, no. 3, pp. 823–836, 2018.
[33] A. Yousefzadeh, L. A. Plana, S. Temple, T. Serrano-Gotarredona, S. B. Furber, and B. Linares-Barranco, "Fast predictive handshaking in synchronous FPGAs for fully asynchronous multisymbol chip links: Application to SpiNNaker 2-of-7 links," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 8, pp. 763–767, 2016.
[34] T. Sharp, F. Galluppi, A. Rast, and S. Furber, "Power-efficient simulation of detailed cortical microcircuits on SpiNNaker," *Journal of Neuroscience Methods*, vol. 210, no. 1, pp. 110–118, 2012.
[35] A. G. D. Rowley, C. Brenninkmeijer, S. Davidson, D. Fellows, A. Gait, D. Lester, L. A. Plana, O. Rhodes, A. Stokes, and S. B. Furber, "Spinntools: the execution engine for the SpiNNaker platform," *Frontiers in Neuroscience*, vol. 13, pp. 231–251, 2019.
[36] I. Sugiarto, P. Campos, N. Dahir, G. Tempesti, and S. Furber, "Optimized task graph mapping on a many-core neuromorphic supercomputer," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE, 2017, pp. 1–6.

BIOGRAPHIES OF AUTHORS

Indar Sugiarto was a postdoctoral researcher at the Advanced Processor Technology of the University of Manchester. He obtained PhD Degree from Technische Universität München in 2015. His researches are in fields of parallel computing, artificial intelligence, machine learning, and robotics. In his postdoctoral work, he developed a fault-tolerance and optimization method for the SpiNNaker neuromorphic supercomputer. He is affiliated with IEEE and ACM as an active member. He serves as an officer for the Indonesian Chapter of IEEE Computer Society. He is also the Deputy of Artificial Intelligence Research Center at Petra Christian University.

Steve Furber received the B.A. degree in mathematics and Ph.D. degree in aerodynamics from the University of Cambridge, Cambridge, U.K., in 1974 and 1980, respectively. He was with the RD Department, Acorn Computer Ltd. from 1981 to 1990 and was a Principal Designer with BBC Micro and the ARM 32-bit RISC microprocessor. He moved to his current position as ICL Professor of Computer Engineering with the University of Manchester, Manchester, U.K., in 1990. Prof. Furber is a Fellow of the Royal Society, the Royal Academy of Engineering, the BCS, and the IET.