Bulletin of Electrical Engineering and Informatics
Vol. 10, No. 1, February 2021, pp. 466~473
ISSN: 2302-9285, DOI: 10.11591/eei.v10i1.2201

Hardware-software partitioning using three-level hybrid algorithm for system-on-chip platform

Tiong Reng Xian, Zaini Abdul Halim, Ching Chia Leong, Tan Jiunn Gim
School of Electrical and Electronic Engineering, Universiti Sains Malaysia, 14300 Niboong Tebal, Malaysia

Article Info

Article history:

Received Feb 7, 2020
Revised May 5, 2020
Accepted Jul 27, 2020

Keywords:

Genetic algorithm
Hardware-software
Hybrid algorithm
Particle swarm optimisation
Partitioning
System on chip

ABSTRACT

This study discusses hardware-software partitioning, which is useful for system-on-chip (SoC) applications. Hardware-software partitioning attempts to obtain the lowest execution time by combining a hardware processor system and a field programmable gate array on the SoC platform in embedded system applications. A three-level hybrid algorithm called GAGAPSO is proposed in this study. The algorithm consists of two successive genetic algorithms (GAs) and one particle swarm optimization (PSO). The drawbacks of these two algorithms are GA has low convergence speed and PSO has premature convergence because of low diversity. These algorithms are combined in this study to achieve high-capacity global convergence and enhanced search efficiency. In this study, three algorithms are developed, namely, GA, GAPSO and GAGAPSO using MATLAB. These algorithms are evaluated on the basis of the number of nodes and the minimum cost that can be achieved. The number of nodes varies from 10 to 1000 nodes. The minimum cost and the number of iterations to achieve the minimum cost are recorded. Results show that GAGAPSO can converge faster than GA and GAPSO. Furthermore, GAGAPSO can achieve the lowest cost for all nodes.

Corresponding Author:

Zaini Abdul Halim,
School of Electrical and Electronic Engineering,
Universiti Sains Malaysia,
Engineering Campus, 14300 Nibong Tebal, Pulau Pinang, Malaysia.
Email: zaini@usm.my

1. INTRODUCTION

System on chip (SoC) is one of the platforms in the current embedded system design. Its features, like low power consumption, light weight and small size, have made it popular in current applications of embedded system design. SoC consists of a field programmable gate array (FPGA) and a hardware processor system (HPS). Instead of using only HPS or FPGA, HPS and FPGA are used together in any application of embedded systems. To use FPGA and HPS, hardware-software partitioning must be implemented to determine which tasks will be implemented in hardware and which task will be implemented in software. Hardware refers to the FPGA that runs as parallel circuits, whereas software refers to the HPS that executes sequential instructions. FPGA tends to improve the execution time through its parallel processing capability but uses many resources [1], which lead to high power consumption. Hardware-software partitioning is proposed for the combination of FPGA and HPS in embedded system applications to require few resources and reduce the power consumption and execution time.

Partitioning is always the key challenge in optimising embedded system efficiency. In the past, partitioning was performed manually by designers based on their experience. However, embedded system

design has increased in complexity over the years, making manual partitioning tedious because of the large number of components with different characteristics that must be considered [2-9]. Therefore, many researches have been conducted to implement the hardware software partitioning automatically [10-15]. The hardware software partitioning can be categorized as exact algorithm and heuristic algorithm. At first, exact algorithm like dynamic programming and branch-and-bound, have been introduced [16-17]. However, the exact algorithm gave poor performance. Hence, the research shifted to heuristic algorithm. Examples of heuristic algorithm are simulated annealing (SA), particle swarm optimisation (PSO) and genetic algorithm (GAs) [17-21]. The heuristic algorithms also have their own drawbacks [22]. For instance, low diversity in PSO causes premature convergence. When the size of the given area is large, the PSO algorithms tends to go back into its local optimum. GA also has its own drawbacks. One of them is, in order to generate a solution, GA needs evolutionary processes like selection, crossover and mutation. As a result, GA suffers from low convergence speed. In addition, it does not guarantee to find a global maximum and it requiresa certain population size and large number of generations in order to obtain a satisfactory result. Hence, GA needs a considerable time to get an optimum solution. Nevertheless, PSO and GA still have some merits, such as PSO has fast convergence speed [23] and GA has capability to easily solve combinatorial optimising problems.

To improve the performance of PSO and GA, hybrid algorithm has been proposed. Example of hybrid algorithms included PSO-GA, GA-PSO, PSO-PSO and GA-GA [2]. Results show that successive GA consumes less time, whereas successive PSO can produce a better graph than GA does and settles at a finite value. The graph provided in [2] indicates that although successive GA did not provide a slope result instead of a step, it used fewer iterations to achive the best cost. PSO uses more iterations to settle the graph. Successive GA can perform fewer iterations to reach the best cost, whereas successive PSO can smoothen the curve. Hence, adding another PSO after successive GA may result in fewer iteration to reach the best cost and a smoother curve. Thus, better results might be obtained in the hardware-software partitioning problem by using the hybrid technique. In the present work, a three-level hybrid model of PSO and GA is constructed to optimise the performance of an embedded system by deciding the implementation of a specific application or to function either in software or hardware. A comparison amongst GA, GAPSO and GAGAPSO is presented. GA is considered a heuristic algorithm. GAPSO is considered a two-level hybrid algorithm and GAGAPSO is considered a three-level hybrid algorithm. The minimum cost versus the number of nodes is also discussed.

2. RESEARCH METHOD

Research methods are divided into three phases, which are the development of GA, PSO and hybrid GAGAPSO. PSO and GA share some similarities. Both begin with a randomised population, and each population has its own fitness value for evaluation. The algorithms update the population and search for the optimum using a random technique. Different from GA, PSO does not have the evolution process, like crossover and mutation. Particles in PSO change themselves through internal velocity, and PSO requires memory to store the parameters. Thus, PSO is faster and simpler than GA [23]. In this implementation, three levels are constructed. The first two levels are successive GAs, followed by a PSO model. Figure 1 shows the implementation architecture. The GA and PSO models are constructed individually before they are combined into a hybrid model.

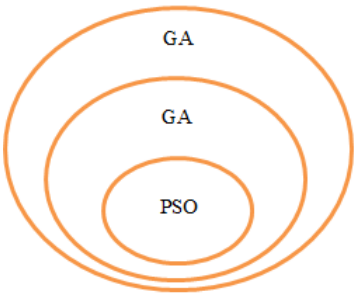

Figure 1. Three-level hybrid model architecture

In GA, natural selection involves three main processes, namely, selection, crossover and mutation. In selection process, the best genes are choosen as parents. These genes are crossed over to produce a better

gene, and a certain probability exists that the gene will mutate. However, the mutation result is not guaranteed. GA imitates the process of natural selection [24-27]. The first step in GA is to initialise all the populations of solutions. The fitness or cost of each solution is evaluated using (1). Once the fitness evaluation is done, the solutions used to perform crossover and mutation will be selected through a selection process. The procedure is followed by the crossover and mutation processes. The fitness for each solution will be re-evaluated after the croosver and mutation processes, which will generate additional solutions. By sorting all the solutions according to the fitness, the additional solutions with the lowest fitness will be eliminated [28-29].

$$ost = 100\left(\frac{HWcost}{All\ HW\ Cost} + \frac{SWcost}{All\ SW\ cost} + \frac{PWcost}{AllPWcost}\right) \quad (1)$$

where

HWcost=hardware implementation cost of particle
SWcost=software implementation cost of particle
PWcost=power implementation cost of particle
All HWcost=total hardware implementation cost of all particles
AllSWcost=total software implementation cost of all particles
AllPWcost=total power implementation cost of all particles in software and hardware

PSO algorithm is inspired by social behaviour of fish schooling or bird flocking. The best way to illustrate PSO is by considering a group of birds searching for food within an area. The birds know how far the food is but do not know where it is. The best strategy to reach the food is to follow the nearest bird to the food. PSO is inspired by this scenario [2]. To demonstrate the PSO model, the population of solutions are first initialised, and the fitness for each solution is evaluated. The fitness function is the cost function as shown in (1). The velocity for each solution is initially set to zero. The velocity and the position are then updated using (2) and (3), respectively [2].

$$v[i] = (W \times v[i] + C\_1r\_1(pBest[i] - x[i]) + C\_2r\_2(gBest[i] - x[i])), \quad (2)$$

$$x[i] = x[i] + v[i], \quad (3)$$

where

$v[i]$=velocity of a particle;
W=damping inertia factor, value from 1 to 0 according to the iteration number (W=W*w_damp);
C_1=self confidence (cognitive) factor;
r_1=random numbers between 0 and 1;
C_2=swarm confidence (social) factor;
r_2=random number between 0 and 1;
$x[i]$=current position of the particle;
$pBest[i]$=position vector of the best solution that this particle achieved thus far;
$gBest[i]$=best position vector obtained thus far by any particle in the population.

The fitness of each particle is evaluated after changing its velocity and position. *gBest* and *pBest* are updated accordingly. The steps are repeated until it reaches the maximum iteration. Figure 2 shows the flowchart of three-level hybrid modelling. PSO is implemented after two successive GAs. The first GA flow is the same as the basic GA flow until excess data are eliminated. After data elimination, the set of data is sent over to the next GA for another round of crossover and mutation. After the data elimination of the second GA, the set of data is passed to the PSO algorithm. In this study, a binary solution is expected by assuming the 1 and 0 values of the hardware and software nodes, respectively. The damping coefficient decreases in each iteration by a factor of *Wdamp*, which is set to 0.97. The cost function as shown in (1) only considers the hardware and software. For simulation purpose, these values are randomly and uniformly generated in the range of 1 to 99.

For a binary problem, the node value is represented by 1 and 0. 1 means the node value will be mapped to hardware and 0 means the node value will be mapped to software. Hard decision rounding (HDR) technique is used to round off the particle. If the node value is larger than 0.5, it will be considered as hardware. Some parameters are used in GA and PSO algorithms. For GA, *Pc*. the crossover probability is set to 0.9, and the *Pm*, the mutation probability is set to 0.1. In PSO, *C*1 and *C*2 for the velocity equation are set

to 2, and *W* is set to 1. In this project, 0.97 is set for the damping value. The number of particles, the population size and the maximum iterations depend on the application [2]; in this study, the values are 512, 60 and 500 respectively.

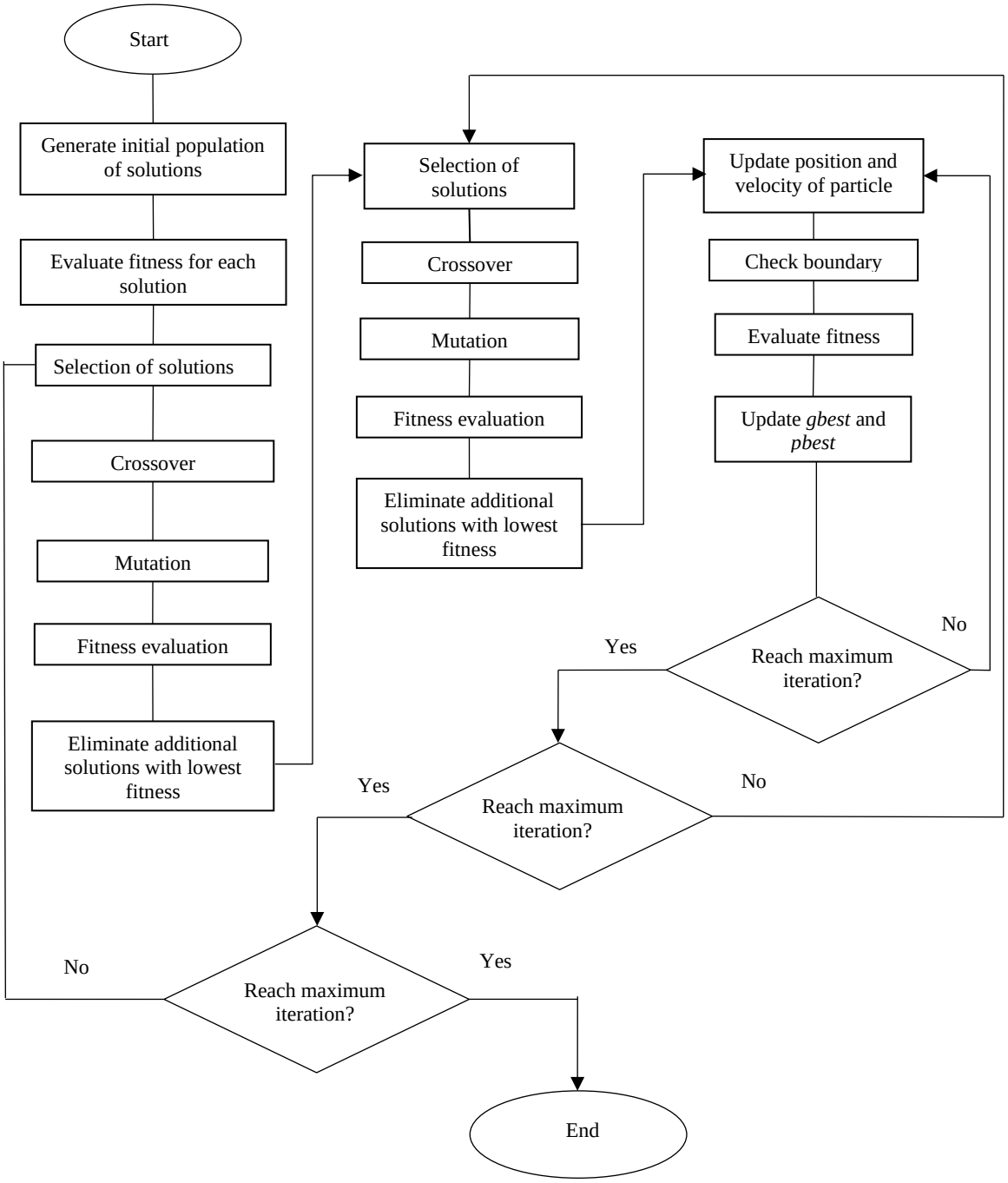

Figure 2. GAGAPSO flowchart

In GA, many approaches can be considered to perform the operations. Three processes are used in GA, namely, selection, crossover and mutation. Each has different methods, and the choice of method will affect the output of the system because GA solely depends on these operators. The selection method in this study is the fitness proportionate selection because it is simple and fast for large numbers of particles. The basic operation of the selection is as follows: the fitness for each particle is normalised and the population is sorted in descending fitness values. Then, the accumulated normalised fitness values are

computed (the accumulated fitness of the last individual should be 1). A random number R between 0 and 1 is selected. The selected individual is the last one which the accumulated normalised value is smaller than R. The next process is the crossover. The method used in this study is heuristic crossover. This method creates one child offspring from two parents. The child gene can be obtained using (4) as follows:

$$O1 = P1 + R(P2 - P1) \quad (4)$$

where

O1=child gene
P1 and P2=parent genes
R=random number between 0 and 1

Lastly, uniform mutation is applied in this algorithm for the mutation process. This process replaces the original value of the selected gene with a uniform value that is randomly generated between the lower and upper boundaries of the gene. The algorithms are programmed using MATLAB and runs on Intel i7, 2.8 GHz processor with an 8GB RAM. In the first testing, the algorithms are analysed based on the random value generated in MATLAB. The number of nodes is set to 500. In the second testing, the algorithms are iterated using values as tabulated in Table 1 for a 12-nodes application. In both testings, the cost is calculated using (1). The best cost is determined as the unchanging cost in 50 consecutive iterations. The graph of cost versus iterations is plotted for 500 iterations.

Table 1. Execution time for each node in hardware and software

Node	Hardware execution time (ms)	Software execution time (ms)
1	18.432	39
2	12.096	18.827
3	12.096	9.610
4	4.032	5.501
5	4.039	3.614
6	4.032	5.653
7	4.039	3.933
8	4.032	3.629
9	4.046	3.623
10	4.032	5.276
11	4.043	3.949
12	4.032	0.004
TOTAL	78.951	102.619

3. RESULTS AND DISCUSSION

In this section, the results of the proposed method are discussed and analysed. The comparisons of the execution times of the three algorithms are also discussed. Figure 3 shows the cost versus iterations graph. The three algorithms, namely, GA, GAPSO and GAGAPSO are plotted in the graph. As expected, the GAGAPSO has a smooth graph with few iterations to achieve the minimum cost. The number of iterations to reach the best cost is approximately 12-31 iterations. This algorithm combines the advantages of successive algorithm into a single model.

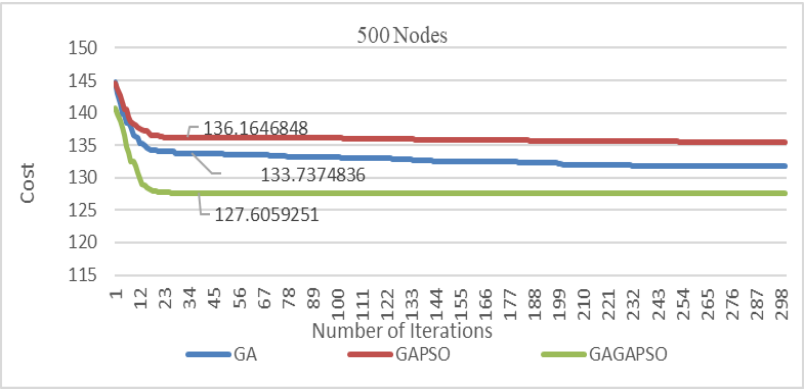

Figure 3. Cost versus iteration for 500 nodes

The graph is used to show the number of iterations required to achieve the best cost. The best cost is determined as the unchanging cost in 50 consecutive iterations. The graph shows that the best cost for GAGAPSO is the lowest amongst the three algorithms. To determine the performance of the algorithms, the time needed to achieve the minimum cost is calculated as shown in (5):

$$t = T_{total} \times \frac{iteration}{500} \quad (5)$$

For each trial or simulation, the total time needed is slightly different because random particles are used in the simulation. Hence, 10 simulation or trials are performed and averaged. The results of the 10 trials are tabulated in Table 2. For the GAPSO and GAGAPSO algorithms, the average time needed to achieve the best cost is 6.1636 s and 0.7830 s respectively. It shows that GAGAPSO is approximately 8 times faster than GAPSO.

Table 2. Result from GAPSO and GAGAPSO algorithms

GAPSO algorithm			GAGAPSO algorithm		
Trial	Number of iterations	Total time needed (s)	Trial	Number of iterations	Total time needed (s)
1	262	5.3649	1	31	1.1368
2	322	5.5162	2	21	0.7100
3	174	3.8544	3	15	0.6149
4	402	7.8660	4	25	0.9702
5	303	5.0621	5	15	0.5203
6	344	7.2371	6	23	0.9236
7	316	6.6388	7	21	0.8727
8	302	6.2603	8	17	0.6755
9	390	8.2670	9	12	0.4960
10	269	5.5694	10	23	0.9102
Average		6.1636	Average		0.7830

Figure 4 shows the cost versus the number of nodes. The minimum cost increases with the number of nodes. GAPSO only performs better than GA when the number of nodes is less than 500. If the number of nodes is more than 500, then GA performs better than GAPSO. The results also show that GAGAPSO performs better than GA for all nodes. A hybrid model must be used to overcome the problem of convergence at the optimum point and achieve the properties of high capacity of global convergence and that the fast, efficient searching. The results indicate that three-level hybrid algorithms can perform better than two-level hybrid models for higher capacity, which is more than 500 nodes. The GA is quite robust but suffers from low convergence speed, which can be improved by using two successive GAs. PSO, which is simpler and faster than GA, can also help the two successive GAs for fast convergence speed. The drawback of PSO, which is premature convergence, is due to low diversity. The problem can be solved using two successive GAs as an input of the PSO.

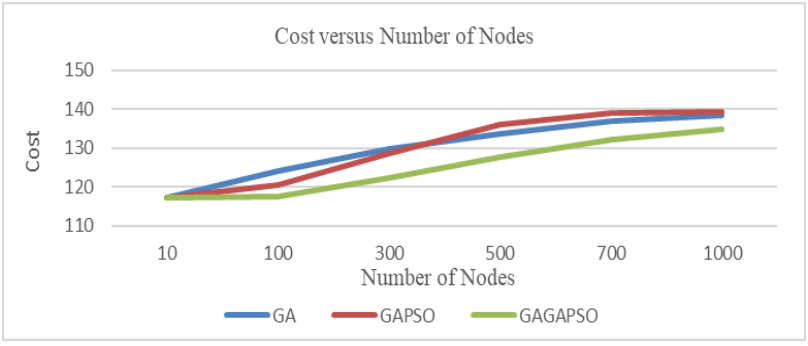

Figure 4. Cost versus number of nodes

Figure 5 presents the percentages of improvement in terms of the minimum cost for GAGAPSO over GAPSO and GAPSO over GA. For GAGAPSO over GAPSO, the maximum improvement is at 500 nodes, with an improvement of 6.3%. After 500 nodes, the improvement of GAGAPSO slightly decreases. When the number of nodes continuously increase, the GAGAPSO algorithm can no longer converge at

the optimum point. The same is true for GAPSO over GA. The maximum improvement achieved by GAPSO over GA is at the 100th node where the improvement is approximately 3%. When the number of nodes continuously increase, the performance of GAPSO decreases, and at 500 nodes, it is worth than GA as shown in Figure 5.

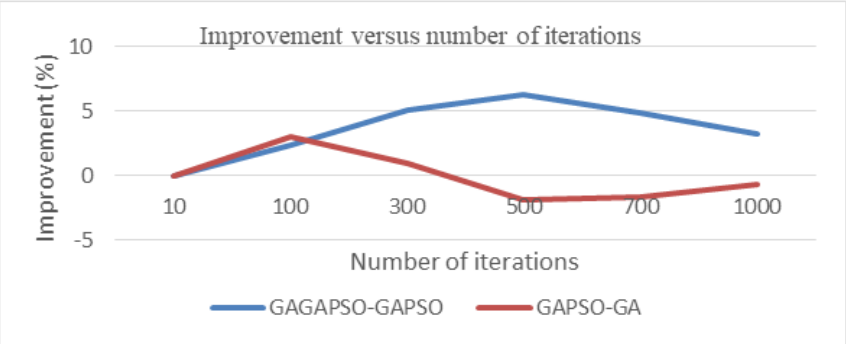

Figure 5. Improvement versus number of nodes

Results from the second simulation show that GAGAPSO and GAPSO algorithms can achieve the minimum cost of 81.63 ms. For a small number of nodes, the minimum cost that can be achieved by GAGAPSO is the same as that of GAPSO. It is also found that the optimum solution for node implementation obtained using GAGAPSO algorithm is 110101000100 as shown in Table 3. The same optimal solution is also obtained using GA and GAPSO algorithm. The hardware node is assigned as 1, whereas the software node is assigned as 0. It means that the Node 1 should be implemented in hardware (FPGA) and node 12 should be implemeneted in software (HPS). By using these combinations, the overall execution time is 81.63 ms as shown in Table 3. If all nodes are implemented in hardware (FPGA) or all nodes are implemented in software (HPS), the impelementation time will be 78.95 ms and 102.62 ms respectively. FPGA can improve the speed but uses more resource that will result in more power consumption whereas the HPS will require more execution time compared to FPGA. Hence hardware software partitioning using GAGAPSO algorithm can help in improving the implementation of embedded system in SoC platform. It will propose an optimal solution for the node implementation in embedded system in order to optimize the power consumptiton and the execution time.

Table 3. Optimal solution obtained using GAGAPSO algorithm

Node Implementation												Hardware execution time (ms)	Software execution time (ms)	Overall execution time (ms)
1	2	3	4	5	6	7	8	9	10	11	12			
1	1	0	1	0	1	0	0	0	1	0	0	42.624	28.362	81.63

4. CONCLUSION

A three-level hybrid GAGAPSO algorithm is developed for hardware-software partitioning using MATLAB. This algorithm exhibits better performance than GAPSO, which is a two-level hybrid algorithm. GAGAPSO can converge eight times faster than GAPSO. Furthermore, it can achieve the lowest minimum cost until 1000 nodes. The best performance it can achieve is 6.3% at 500 nodes. The performance of the hybrid algorithm increases with the number of nodes. However, the performance has its own limitations. For the two-level hybrid algorithm, GAPSO, the data show a maximum limit of 500 nodes. For more than 500 nodes, GAPSO performs worse than GA. GAGAPSO can achieve good performance until 1000 nodes. Future works should be conducted to determine the maximum number of nodes for GAGAPSO algorithm. Other parameters, like crossover rate, mutation rate and population size can also be studied in future works.

ACKNOWLEDGEMENTS

The authors would like to thank the School of Electrical and Electronic Engineering, Universiti Sains Malaysia for supporting this research under Grant No. 1001/PELECT/8014152.

REFERENCES

[1] Haresh Pandya, M. Rangapariya, J, Rajput, "Implement Embedded Controlling using FPGA Chip," *International Journal of Reconfigurable and Embedded Systems*, vol. 8, no. 2, pp. 130-144, 2019.

[2] M. B. Abdelhalim, A. E. Salama and S. E. -. Habib, "Hardware Software Partitioning using Particle Swarm Optimization Technique," *2006 6th International Workshop on System on Chip for Real Time Applications*, Cairo, pp. 189-194, 2006.

[3] S. Ismae, O. Tareq, Y. Taher Qassim, "Hardware/software Co-design for a Parallel Three-dimensional Bresenham's Algorithm," *International Journal of Electrical and Computer Engineering IJECE*, vol. 9, no. 1, pp. 148-156, 2019.

[4] A. Iguider, K. Bousselam, A. En-Nouaary, O. Elissati and M. Chami, "A Novel Approach for Hardware Software Partitioning in Embedded Systems," *2019 International Conference on Wireless Technologies, Embedded and Intelligent Systems (WITS)*, Fez, Morocco, pp. 1-5, 2019.

[5] Andrew Tzer-Yeu Chen, *et al.*, "Accelerating SuperBE with Hardware/Software Co-Design," *Journal of Imaging*, vol. 4, no. 10, pp. 1-17, 2018.

[6] M. Jemai and B. Ouni, "Hardware Software Partitioning of Control Data Flow Graph on System on Programmable Chip," *Microprocessor and Microsystem*, vol. 39, no. 4-5, pp. 259-270, 2015.

[7] M. B. Abdelhalim and S.D. Habib, "An Integrated High-Level Hardware/Software Partitioning Methodology," *Design Automation for Embedded Systems*, vol. 15, pp. 19-50, 2011.

[8] L. An, *et al.*, "Algorithm of Hardware/Software Partitioning Based on Genetic Particle Swarm Optimization Design," *Journal of Computer-Aided Design & Computer Graphics*, vol. 22, pp. 927-942, 2010.

[9] Lanying Li, *et al.*, "Hardware/software Partitioning Based on Hybrid Genetic and Tabu Search in the Dynamically Reconfigurable System," *International Journal of Control and Automation*, vol. 8, No 1, pp. 29-36, 2015.

[10] I. Bahri, *et al.*, "Optimal Hardware/Software Partitioning of a System on Chip FPGA-based Sensorless AC Drive Current Controller," *Mathematics and Computers in Simulation*, vol. 90, pp. 146-161, 2013.

[11] P. Liu, *et al.*, "Hybrid Algorithms for Hardware/Software Partitioning and Scheduling on Reconfigurable Devices," *Mathematical and Computer Modelling*, vol. 58, no. 1-2, pp. 409-420, July 2013.

[12] I. Mhadhbi, *et al.*, "An Efficient Technique for Hardware/Software Partitioning Process in Codesign," *Hindawi*, vol. 2016, no. 6382765, pp. 1-11, July 2016.

[13] Y. Jiang et al., "Uncertain Model and Algorithm for Hardware/Software Partitioning," *2012 IEEE Computer Society Annual Symposium on VLSI, Amherst*, MA, pp. 243-248, 2012.

[14] Yiming Jing, *et al.*, "Application of Improved Simulated Annealing Optimization Algorithms in Hardware/Software Partitioning of the Reconfigurable Systen-on Chip," *International Conference on Parallel Computing in Fluid Dynamics*, vol. 405, pp. 532-540, 2013.

[15] Wenzhong Guo, *et al.*, "A Hybrid Multi-obejective PSO Algorithm with Local Search Strategy for VLSI Partitioning Chip," *Frontiers of Computer Science*, vol. 8, pp. 203-216, 2016.

[16] Imene Mhadhni, *et al.*, "An Efficient Technique for Hardware Software Partitioning Process in Codesign," *Hindawi Publishing Corperation, Scientific Programming*, vol. 2016, no. 6382765, pp. 1-11, 2016.

[17] P.K. Sahu, *et al.*, "Extending Kernighan-Lin Partitioning Heuristic for Application Mapping onto Network-on-Chip," *Journal of Systems Architecure*, vol. 60, no. 7, pp. 562-578, 2014.

[18] A.Kacem, *et al.*, "A Hybrid Algorithm to Size the Hospital Resources in the Case of Massive Influx of Victims," *International Journal of Electrical and Computer Engineering*, vol. 10, no. 1, pp. 1006-1016, 2020.

[19] Palak and P.Gullia, "Hybrid Swarm and GA Based Approach for Software Test Case Selection," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 6, pp. 4898-4903, 2019.

[20] F.F.Yeng, *et al.*, "The Saturation of Population Fitness as a Stopping Criterion in Genetic Algorithm," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 5, pp. 4130-4137, 2019.

[21] A. K.Ariyani, *et al.*, "Hybrid Genetic Algorithm and Simulated Annealing for Multi-trip Vehicle Routing Problem with Time Windows," *International Journal of Electrical and ComputerEngineering*, vol. 8, no. 6, pp. 4713-4723, 2018.

[22] J. Henkel and R. Ernst, "An approach to automated hardware/software partitioning using a flexible granularity that is driven by high-level estimation techniques," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 2, pp. 273-289, April 2001.

[23] Eberhart and Yuhui Shi, "Particle swarm optimization: developments, applications and resources," *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, Seoul, South Korea, vol. 1, pp. 81-86, 2001.

[24] W. Li, *et al.*, "Hardware/Software Partitioning of Cpmbination of Clustering Algorithm and Genetic Algorithm," *Internation Journal of Control and Automation*, vol. 7, no. 1, pp. 347-356, 2014.

[25] E. T. Tan and Z. A. Halim, "Performance evaluation of genetic algorithm to solve hardware-software partitioning design: A factorial design analysis," *TENCON 2017 - 2017 IEEE Region 10 Conference*, Penang, pp. 439-442, 2017.

[26] S. Suresh, *et al.*, "Hybrid Real-coded Genetic Algorithm for Data Partitioning in Multi-Round Load Distribution and Scheduling in Heterogeneous System," *Applied Soft Computing*, vol. 24, pp. 500-510, Nov 2014.

[27] R. Faraji and H.R.Naji, "An Efficient Crossover Architecture for Hardware Parallel Implementation of Genetic Algorithm," *Neurocomputing*, vol. 128, pp. 316-327, March 2014.

[28] Erik D. Goodman, "Introduction to Genetic Algorithm," *GECCO'11 Proceeding of the 13th Annual Conference Companion on Genetic and Evolutionary Computation*, pp. 205-226, July 2014.

[29] P. Kora and P.Yadlapalli, "Crossover Operators in Genetic Algorithms: A Review," *International Journal of Computer Applications*, vol. 162, no. 10, pp. 34-36, 2017.