International Journal of Engineering & Technology, 7 (3.15) (2018) 179-182

International Journal of Engineering & Technology

Website: www.sciencepubco.com/index.php/IJET

Research paper

Genetic Algorithm (GA) for Multiprocessors Scheduling

Sainuddin Mohd Jufri, *Saiful Izwan Suliman, Mohd Asri Mansor, Yuslinda Wati Mohamad Yusof, Roslina Mohamad, Shahrani Shahbudin, Murizah Kassim

Faculty of Electrical Engineering, Universiti Teknologi MARA
40450 Shah Alam, Selangor, Malaysia
**Corresponding author*
**Corresponding author E-mail: saifulizwan@uitm.edu.my*

Abstract

In a multiprocessors environment, an algorithm is used to effectively delegate tasks to the processors. The algorithm act as the resource distributor and controller to ensure that the processors are equally utilized thus optimizing the usage of the processors. Thus, the objectives of the study are to replicate the Genetic Algorithm (GA) and use the algorithm to shorten the timespan for the parallel task to be completed. The algorithm has the ability to improve and find the optimum solution based on the population needs. Hence, C++ is used as the platform for the simulation. Experimental results validate that the designed program has the ability to replicate the GA in achieving the objectives of the study.

Keywords: *Genetic Algorithm; Multiprocessors; Parallel Processors;*

1. Introduction

Most computers these days run on at least two processors, also known as the parallel processor architecture. It plays a major role on the overall performances of our computers as it enhances the multitasking processes making it faster and more efficient. Thanks to the rapid growth of modern technology, end-users can reap the benefits of endless possibilities.

Yet, even with advanced computer architecture and hardware section, computers still need helps from the software section serving as the brains on managing the architecture itself in order to reach optimum performance. Arguably, one cannot deny the greater importance of the software section. Thus, computing algorithm plays major part in the overall performances of the computer. For instance, algorithm used can be utilized as source allocator, task scheduler and divide task prioritization among others. Furthermore, dividing a task into appropriate size and numbers plays the major role of parallel execution as it holds the balance between a high degree of parallelism and a low communication overhead [1]-[3].

Many previous studies have proven that using algorithm can significantly increase the completion time and efficiency of the processors running in parallel architecture [3], [4], [7], [8], [10], [11].

So, in this paper a Genetic Algorithm (GA) is designed and utilized to optimize the timespan of a Parallel Processor Job (PPJ) in hopes to further increase the overall performance and efficiency of the computer. There are two main objectives of this study, which are

(i) to design an GA specified to perform PPJ

(ii) to optimize the timespan to complete a PPJ using GA.

The study applied a few predetermined controlled variables as to limit the scope of the study as well as to further understand the phenomenon in a more controlled environment. These are (i) the number of processors used which is four processors and (ii) the complexity of the PPJ or the chromosome.

2. Research Method

Genetic Algorithm is an algorithm that has the ability to self-learn, almost like an artificial intelligence, on finding the best optimum solution to any problems. Originally, the GA is inspired from the mother nature herself. Biologically, the nature tends to find the best characteristic to pass down to the next generation in order to give the predecessor a better chance of surviving. Thus, the GA is in the same class as few other algorithms such as the Particle Swarm Algorithm (PSO), Evolution Strategies (ES) and Evolutionary Programming [6], [9].

The GA is very simple yet a very reliable algorithm as it has the capable of finding optimum solution for any given problem. The algorithm works by generating steady improvement (towards the objective function/goal) as the generation grows alongside time as illustrated in Figure 1.

Steps below explain the basic flow of the algorithm [6], and shown in Figure 2:

1. Randomly generate initial population of individuals (first generation).
2. Evaluate the fitness of each individual according to the objective function or goals; distance and time taken.
3. The following steps are repeated until termination criterion is met:
   i. Reproduce the best individuals.

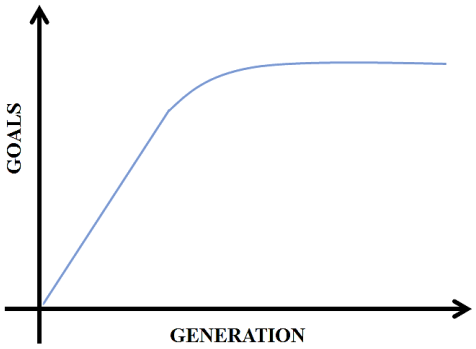

Fig. 1: Graph of generation *vs* goals

ii. Best individuals are put through crossover and mutation phase.
iii. Birth of new offspring/individuals.
iv. Evaluate the new individuals.
v. Replace least-fit individuals.

4. The termination occurs when either:
   i. The goals are achieved, or
   ii. The growth/improvement of the generation stops (iteration limit).

In this study, the GA is simulated by using the C++ language on *CodeBlocks* platform. For every Genetic algorithm, the most important feature is the fitness function of the targeted subject. This is due to the function itself can lead the algorithm towards the best and optimum solution because the fitness function directly reflects on how optimal the solution might look like. Hence, choosing the suitable parameters is so important in making sure that the algorithm can work under the best possible environment [9].

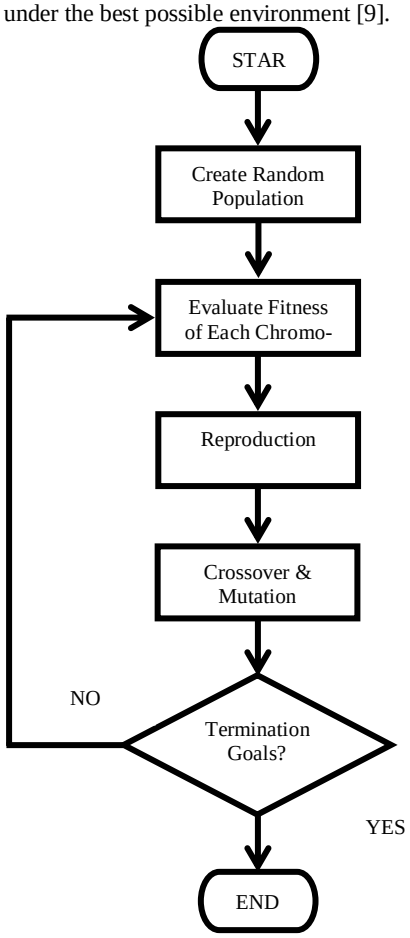

Fig. 2: Steps in Genetic Algorithm

Fast Fourier Transform (FFT) as shown in Figure 3 is a symmetrical graph used to represent the computational time of the processors. Each vertex represents a task with specific completion time. As shown in the figure, there are twenty-eight tasks grouped in five columns. The first column consists of eight tasks with no prerequisite task before them. It means that these eight tasks can be started at any time. For the remaining twenty tasks, they have prerequisite task(s) that must be completed first before their task can be started. For example, task number 9 can only be started if tasks 1 and 2 have been completed. One of the notable characteristic of the graph is that every input vertex has its dedicated output vertex with specific computational time [1],[6].

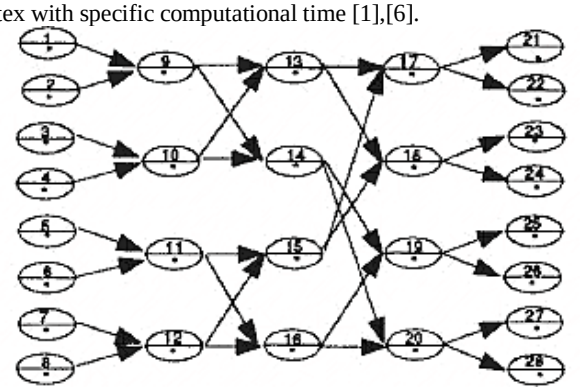

Fig. 3: Fast Fourier Transform (FFT)

As shown in Table 1, every single vertex has its own completion time. As we can see, a big portion of the computation time is saturated in the middle of the FFT section.

Table 1: Completion Time for each Vertex

Vertex	Time
1-8	1
9-12	20
13-16	30
17-20	20
21-28	1

Table 2 shows how data is tabulated by referring to the information from Table 1 and Figure 3. The FFT consists of twenty-eight tasks and each task is connected with other neighbouring tasks to complete the whole process. The data in Table 2 also shows the processing time for each task, its parents and children.

Table 2: Complete Computational Time for Each Vertex

Task	Completion Time	Parent 1	Parent 2	Child 1	Child 2
T1	1	0	0	0	9
T2	1	0	0	0	9
T3	1	0	0	0	10
T4	1	0	0	0	10
T5	1	0	0	0	11
T6	1	0	0	0	11
T7	1	0	0	0	12
T8	1	0	0	0	12
T9	1	0	0	0	14
T10	1	0	0	0	14
T11	20	5	6	15	16
T12	20	7	8	15	16
T13	30	9	10	17	18
T14	30	9	10	17	18
T15	30	11	12	17	18
T16	30	11	12	19	20
T17	20	13	15	21	22
T18	20	13	15	21	22
T19	20	14	16	25	26
T20	20	14	16	27	28
T21	1	0	17	0	0
T22	1	0	17	0	0
T23	1	0	18	0	0
T24	1	0	18	0	0
T25	1	0	19	0	0
T26	1	0	19	0	0
T27	1	0	20	0	0
T28	1	0	20	0	0

2.1. Generation of Initial Population

The GA is a nondeterministic method. The algorithm depends on the randomness of the initial population. In this study, there are four processors utilized to complete the overall tasks. A total of 100 chromosomes were generated randomly before further improved using genetic operators in GA.
Table 3 shows the example of genes for each chromosome. The C(n) is the chromosome number for the population which was set to 100 (n=100). While, the T(I) is the task numbers and 28 tasks were investigated in this study (I=28). Thus, a single chromosome is made up from 28 individual tasks, and the four processors (P1, P2, P3, and P4) randomly assigned to complete those tasks.

Table 3: Sample of Chromosomes

Chromosome	**T_1**	**T_2**	**T_3**	**T_4**	**T_5**	**…**	**$T_{(m)}$**
C_1	P_1	P_2	P_3	P_4	P_1		P_4
C_2	P_2	P_3	P_4	P_1	P_2		P_1
…							
$C_{(n)}$	P_4	P_1	P_2	P_3	P_4		P_3

2.2. Fitness Evaluation

The fitness value of every single chromosome is calculated to evaluate its quality towards the objective function. The fitness value is measured as in unit of time. The evaluation process was conducted by using the following equation :

$$f(i) = \sum_{i=1}^{i=m} T_i + H_i \quad (1)$$

T_i : Completion time for task i
H_i : Communication time for task i

Communication cost is added if different processor is utilized for the subsequent task, in this study the communication cost is set to 25. If the same processor is used, the communication cost will be zero. The chromosome with the lowest fitness value is considered the best as it takes the shortest computational time to complete all tasks.

2.3. Sorting and Selection

The entire population of chromosomes is then sorted and the best 30 chromosomes are sorted out from the population. The selection is inspired from the natural selection in evolution. In which the best chromosome with the best fitness value has the higher probability of surviving compared to those whom have less. These selections were meant towards for a better solution.

2.4. Crossover and Mutation

The crossover is something much similar to the act of sexual reproduction of the living thing. In attempt of pushing for a better solution, the selected chromosomes are then combined and creating a new chromosome or a child. Meanwhile, the new chromosome would have both the characteristics of the parents.

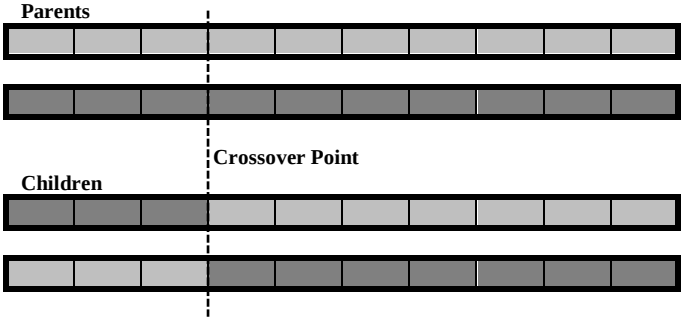

Fig. 4: Process of Crossover

The idea of a crossover is both to collect and merge both qualities of the parents and increase the chances of producing a better offspring. This crossover process is conducted based on the predetermined crossover point. For example, as shown in Figure 4, the crossover process will produce chromosome for offspring 1 which is made of 30% genes from parent 1 and 70% genes of parent 2. The same goes for child 2 where it gets 1 : 7 gene ratio of parent 2 and parent 1 respectively. Then the produced children undergo a mutation process based on predetermined probability of mutation. After preliminary tests conducted in the beginning of this study, the best probability rate was found to be at 0.1 (10%).

2.5. New Population

After the process of crossover and mutation, the new chromosomes (offspring) will be determined for its fitness value. This population of offspring will then be sorted based on the fitness value. A new population is then created; which involves 30 child chromosomes, 30 parent chromosomes and 40 randomly generated chromosomes as shown in Figure 5. This new population will undergo the same processes as mentioned earlier until the predetermined stopping criterion is exhausted.

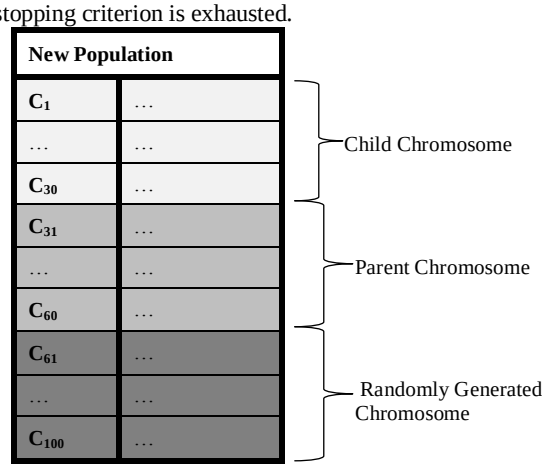

Fig. 5: New Population of Chromosomes

2.6. Termination

The steps in Genetic Algorithm are repeated many times on the possible solutions (chromosomes). The objective is to produce good quality chromosomes that can give the desired outcomes. However, condition(s) must be set to limit these repetitive procedures. After a repeated exposures of the same procedures, the chromosomes might have already reached its optimal solutions (global or local). Therefore, repeating the same processes will not further improve its quality. Thus, its process must be terminated. There are two stopping criteria used in this study which are, (i) the best solution found in the scientific literature and, (ii) number of iteration. If the proposed method manages to get less computational time than the best reported so far, the algorithm will stop its process. If it could not reach the set value, the algorithm will stop once the iteration limit is reached.

3. Results and Analysis

In this study, we attempted to simulate an GA that can perform PPJ and also to help improve the completion time of specified tasks' performances.
The program was designed using the *CodeBlocks* software. As the generation grows in time, the fitness value of the chromosomes reduces as shown in Figure 6. This is perpendicular as one of the main objectives of this study which is to minimize the time span of completing a parallel processors scheduling.

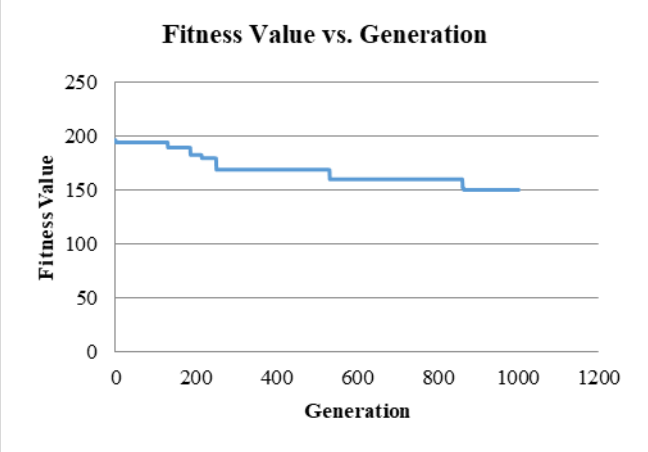

Fig. 6: Fitness Value vs Generation

As illustrated in the graph, the proposed algorithm was unable to produce a significant improvement up until the 147th generation with fitness value of 195. It goes down steadily until iteration 272 with fitness value of 169. This value remains unchanged until iteration 520 before it improves a bit to 160. As seen in the graph, the best solution of 150 was obtained when the iteration reaches 870 and no further improvement observed after that until it reaches the iteration limit of 1000.

Table 4: Comparison of Results

Algorithm	Completion Time	Number of Processor(s)
Serial	296	-
Standard GA	173	3
Proposed method	**150**	**4**
DLS	175	7
MH	175	7
MCP	148	8

The test examples are selected from [1]. The GA is compared with several other algorithms and scheduling such as the standard Genetic Algorithm (GA), Dynamic Level Scheduling (DLS), MH, and Modified Critical Path (MCP) as shown in Table 4.

The GA used in this study outperforms some of the benchmark test function of several examples such as the DLS and MH. Even with fewer processor used, the GA manages to complete the task much faster.

Although, the MCP manages to run the tasks faster, it required four more processors. The difference in terms of the capacity and the capability is quite impressive. Yet, the GA still has several rooms for improvement.

Genetic Algorithm (GA) basically has almost identical features as the Genetic Algorithm (GA). However, the GA manages to perform better compared to the GA even with less number of processors used.

Notably, the results obtained clearly indicate that the GA has the capacity to not only increase the efficiency of the processors but also allow the processors to run the PPJ simultaneously (parallel programming). In conjunction to that, the simulation also strengthens the theory that running parallel processors is better that running it in series. Parallel processing is able to fully utilise the processors capacity and manages to shorten the idle time in between tasks and sub-tasks.

4. Conclusion

Genetic Algorithm theoretically has the ability to significantly improve the overall performances of the processors by fully utilizing its capacity. Not only have that, the GA provided a solution for the processors to run the task simultaneously. Running a multicore processor has several challenges such as tasks division, idle time of the processors and job sequence, however, all of these problems can be addressed by using the GA.

The result obtained from this study clearly supports the theories regarding the ability of the algorithm used. The program designed has managed to replicate the characteristics and qualities of the GA. Not only that, the GA made is capable of simulating and completing the specified PPJ.

As a conclusion, the overall experiment is a success. The results obtained are reliable and significant.

Acknowledgement

The authors would like to express the gratitude to the Ministry of Higher Education, Malaysia and Universiti Teknologi MARA, Selangor, Malaysia for the financial support given for this project. project (Fundamental Research Grant Scheme - FRGS) [File No : 600-RMI/FRGS 5/3 (023/2017)].

References

[1] Caswell, D. J., & Lamont, G. B. (2003, December). Distributed processor allocation for discrete event simulation and digital signal processing using a multiobjective evolutionary algorithm. In Evolutionary Computation, 2003. CEC'03. The 2003 Congress on (Vol. 3, pp. 1803-1810). IEEE.

[2] Bazoobandi, H. A., Khorashadizadeh, M., & Eftekhari, M. (2014, February). Solving task scheduling problem in multi-processors with genetic algorithm and task duplication. In Intelligent Systems (ICIS), 2014 Iranian Conference on (pp. 1-4). IEEE.

[3] McCreary, C. L., Khan, A. A., Thompson, J. J., & McArdle, M. E. (1994, April). A comparison of heuristics for scheduling DAGs on multiprocessors. In Parallel Processing Symposium, 1994. Proceedings., Eighth International (pp. 446-451). IEEE.

[4] Hoseinpour, A., Lahijani, M. J., Hoseinpour, M., & Kazemitabar, J. (2018). Fitness function improvement of evolutionary algorithms used in sensor network optimisations. IET Networks, 7(3), 91-94.

[5] Khan, Z. A., Siddiqui, J., & Samad, A. (2016, March). A novel task scheduling algorithm for parallel system. In Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on (pp. 3983-3986). IEEE.

[6] Albacea, E. A. (1997, April). An optimal parallel algorithm for two-processor scheduling. In High Performance Computing on the Information Superhighway, 1997. HPC Asia'97 (pp. 220-223). IEEE.

[7] Ahmad, I., Kwok, Y. K., & Wu, M. Y. (1996, June). Analysis, evaluation, and comparison of algorithms for scheduling task graphs on parallel processors. In Parallel Architectures, Algorithms, and Networks, 1996. Proceedings., Second International Symposium on (pp. 207-213). IEEE.

[8] Kwok, Y. K., Ahmad, I., & Gu, J. (1996, August). FAST: A low-complexity algorithm for efficient scheduling of DAGs on parallel processors. In Parallel Processing, 1996. Vol. 3. Software., Proceedings of the 1996 International Conference on (Vol. 2, pp. 150-157). IEEE.

[9] Kaur, J., Singh, S., & Singh, S. (2016, February). Parallel Implementation of PSO Algorithm Using GPGPU. In Computational Intelligence & Communication Technology (CICT), 2016 Second International Conference on (pp. 155-159). IEEE.

[10] Faber, Ł., & Boryczko, K. (2016, September). Efficient parallel execution of genetic algorithms on Epiphany manycore processor. In Computer Science and Information Systems (FedCSIS), 2016 Federated Conference on (pp. 865-872). IEEE.

[11] Lassabe, N. (2013, December). Optimized SCC Processor by Using Parallel Genetic Algorithms. In Technologies and Applications of Artificial Intelligence (TAAI), 2013 Conference on (pp. 403-407). IEEE.