International Journal of Engineering

Journal Homepage: www.ije.ir

Efficient Parallelization of a Genetic Algorithm Solution on the Traveling Salesman Problem with Multi-core and Many-core Systems

M. Abbasi*, M. Rafiee

Department of Computer Engineering, Engineering Faculty, Bu-Ali Sina University, Hamedan, Iran

PAPER INFO

Paper history:
Received 10 November 2019
Received in revised form 23 December 2019
Accepted 04 January 2020

Keywords:
Genetic Algorithm
Parallel
Multi-core
Many-core
Traveling Salesman Problem

ABSTRACT

Efficient parallelization of genetic algorithms (GAs) on state-of-the-art multi-threading or many-threading platforms is a challenge due to the difficulty of scheduling hardware resources regarding the concurrency of threads. In this paper, for resolving the problem, a novel method is proposed, which parallelizes the GA by designing three concurrent kernels, each of which are running some dependent effective operators of GA. The proposed method can be straightforwardly adapted to run on many-core and multi-core processors by using Compute Unified Device Architecture (CUDA) and Threading Building Blocks (TBB) platforms. To efficiently use the valuable resources of such computing cores in concurrent execution of the GA, threads that run any of the triple kernels are synchronized by a considerably fast switching technique. The offered method was used for parallelizing a GA-based solution of Traveling Salesman Problem (TSP) over CUDA and TBB platforms with identical settings. The results confirm the superiority of the proposed method to state-of-the-art methods in effective parallelization of GAs on Graphics Processing Units (GPUs) as well as on multi-core Central Processing Units (CPUs). Also, for GA problems with a modest initial population, though the switching time among GPU kernels is negligible, the TBB-based parallel GA exploits the resources more efficiently.

***doi**: 10.5829/ije.2020.33.07a.12*

1. INTRODUCTION

Meta-heuristic optimization algorithms [1-3] like Genetic Algorithms (GAs) have been widely used in science and engineering problems [3-5]. GA is considered as a class of evolutionary algorithms that are used for finding approximate solutions in search [6], optimization problems [7, 8], image processing [9], optimizing artificial neural networks [10], scheduling [11, 12], and rule-based systems [13].

The main difficulty with using GA is the considerable iterations of the genetic algorithm [14]. In such cases, increasing the number of generations would raise the rate of crossovers and mutations. These, in turn, expressively increase the time complexity of the organized algorithm. Therefore, many researchers try to examine parallelization methods for GA on multi-core systems as well as many-core systems [15]. A common approach is to migrate the computation of fitness, mutation, crossover, and selection functions to parallel machines [16]. An interesting point is the efficiency of the deployed parallel kernels in using the computational resources of systems for accelerating GA. Although different approaches with different complexities have been presented for parallel programming on multi-core and many-core systems, a few of them have been carried out to quantitatively assess and compare the efficiency of parallel kernels on multi-core machines with that of many-core systems. Also, due to the intricate design of parallel kernels, none of them have efficiently utilized the computational resources of the parallel systems to accelerate GA.

A pilot study in the domain of efficient parallelization of GA is the research of Zhu et al. [17], which combines the mechanism of Threading Building Blocks (TBB) and Message Passing Interface (MPI) platforms to parallelize a GA-based solution of the Traveling Salesman Problem (TSP). Consistent with the results obtained from

*Corresponding Author Institutional Email: *abbasi@basu.ac.ir* (M. Abbasi) implementation on different datasets in one hundred generations, the achieved acceleration rate on four processing cores in 144 and 1889 cities is 2.1 and 2.55, respectively. Studies directed by Fujimoto et al. [18] and Chen et al. [19] can be considered as preliminary studies, which present two different methods for parallelizing GA computations on the Compute Unified Device Architecture (CUDA) platform. None of those studies can fully exploit the computational resources of the Graphics Processing Units (GPUs). Other studies, like [20, 21] and [22-25], have incoherently executed parallel kernels for GA-based solutions of TSP on GPUs.

The most recently conducted study in this trend is the study conducted by Saxena et al. [26], which compares the efficiency of Open Multi-Processing (OpenMP) and CUDA through running parallel GA-based optimization kernels on multi-core Central Processing Units (CPUs) and GPUs. Unfortunately, this study does not offer a typical experimental setting for all parallel kernels. As a consequence, their shallow results cannot be used for any decision and comparison as to the efficiency of those parallelization platforms.

Our investigation shows that all of the researches in the field of GA parallelization have focused on the unclear design of parallel algorithms. Also, none of the recent studies have inspected the approaches for efficient parallelization of GA operators on multi-core platforms like TBB and CUDA. The different models of parallel processing of the threads and diverse approaches of synchronization of threads in different blocks of GPU have not been exactly studied in any of them.

Motivated by the above mentioned problem, this paper proposes an efficient method for parallelizing the key operators of the genetic algorithm. The proposed parallelization method is based on the structure of multi-core CPUs and many-core GPUs. It shows that by concise use of the parallel resources of a multi-core CPU, the efficiency of multi-core parallelization can be higher than that of a many-core GPU. Also, the presented study inspects the effect of different parameters of GA-based solutions of TSP on the performance of the parallel kernel on both multi-core systems as well as many-core systems.

TSP is an NP-complete problem. The main idea of TSP is to find the shortest path among a set of cities, provided that each city is visited only once, and the source city should be revisited at the end.

The rest of the paper is structured as follows. Section 2 reviews the structure of CUDA and TBB platforms. In addition, the GA solution of the TSP is described in the third section. Then, the offered approach for parallel implementation of a GA is discussed. Experiments and their corresponding analyses are explored in the following section. Section 5 concludes the paper and proposes some future directions.

2. BACKGROUND

This section gives a brief overview of the related concepts, including the architecture of GPU in the CUDA platform and the main ingredients of TBB architecture.

2. 1. CUDA The graphics processing unit is a tool dedicated to display graphic images at workstations, game consoles, or personal computers [27, 28]. CUDA provides features for developers to use the hardware capabilities of Nvidia graphics cards in non-graphical programs and speed up the execution speed of complex algorithms using GPU capabilities. CUDA supports the main factors involved in computing from two different points of view: host and device. The host performs the main program while the machine aids in processing. A typical scenario is that the CPU is considered as the host and the GPU is considered as a help to the processor.

Any program written in CUDA can consist of several kernels. Each kernel is implemented by a grid of several blocks. Each block is made of several threads. These threads are responsible for implementing the program [29].

2. 2. TBB Intel Threading Building Blocks (Intel® TBB) is a common C++ library for writing parallel shared-memory programs. Using this library provides benefits including synchronized containers, scalable memory allocator, work-stealing task scheduler, low-level synchronization primitives. This library is considered as the best tool for task-based parallelism. The details of scheduling by this efficient library can be found in many resources, such as [30, 31].

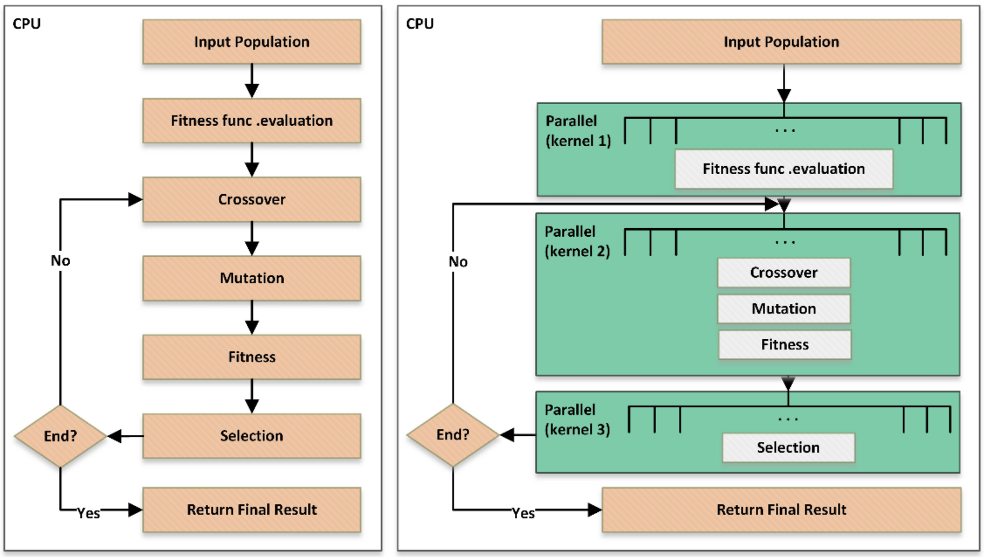
3. THE PROPOSED APPROACH

The common method for solving the TSP with a genetic algorithm is shown by a flowchart in Figure 1(a). This algorithm is used in a lot of applications in different areas of science and engineering [32].

Population: each chromosome contains a fixed number of genes. In this case, each city is represented by a gene, and each chromosome is a permutation of cities.

The fitness function of the initial population: for each chromosome, the function produces a non-negative integer value, which indicates fitness and individual aptitude of each chromosome. In the calculating the fitness of each chromosome in TSP, a matrix containing the coordinates of cities is used. The distance between every pair of cities in a chromosome is computed according to the following equation [33]:

$$f(x) = \left(\sum_{i=2}^{n} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}\right) + \sqrt{(x_1 - x_n)^2 + (y_1 - y_n)^2} \quad (1)$$



(a) Sequential Genetic Algorithm, | (b) Parallel Genetic Algorithm

Figure 1. Sequential and parallel approaches for Genetic Algorithm

In the above equation, x_i and y_i denote the coordinate of the i-th city of a chromosome.

Crossover: this operator exchanges the information between the paired chromosomes and also controls the convergence speed of the genetic algorithm with a probability. This probability value P_c is called the crossover rate. For doing a crossover, a parent and a random position between the parent's genes are considered. Then, all the genes at both sides of the parent chromosome with respect to the specified position are moved to form a new chromosome.

Mutation: This operator produces the new chromosome by randomly changing one of the genes with a low probability. The overall probability of mutation on a chromosome is called mutation rate, which is denoted by P_m.

Calculating the fitness of a generation: the fitness function of the population is calculated using crossover and mutation operators.

Selection: there are diverse methods for selecting the best chromosome and transferring it to the next generation. Commonly, the tournament method is used for selection. In this method, two chromosomes are randomly selected from the population. Then, a random number between zero and one is selected as r. Next, two fitted chromosomes or the ones which are less adapted are selected as the parents. These two chromosomes are then returned to the initial population and again participated in the selection process. Finally, the selected chromosomes are recognized as the next generation and are sent to the next round of algorithm implementation [16]. The statistical analysis of the operators of genetic algorithms was thoroughly investigated in many research studies, including [34-36]. This issue has a substantial effect on designing an efficient GA [37].

The general assembly of the sequential and parallel genetic algorithm is demonstrated in Figure 1(b). In all of the offered parallel kernels, the original population is identical. The aim of the proposed kernels for parallel GA is to assess and compare the efficiency of parallelized codes using CUDA and TBB. In the sequential method, all the operations of the genetic algorithm are performed consecutively. But, in the parallel method, the important operations of the GA are executed in parallel, which decreases the runtime of the GA. These operators are fitness, crossover, mutation, and selection. Figure 1(b) demonstrates the flowchart of the proposed method for parallelizing GA. Regarding the structure of GPUs, only the threads in one CUDA block can be synchronized. The synchronization is a crucial mechanism in the genetic algorithm, where some operators need to be executed in sequence. The most important benefit of our method over recent methods like [19] is offering a technique for solving the problem of synchronizing threads which can synchronize the threads of more than one block. For this purpose, we form three different kernels, each of which corresponds to a different function of GA. These kernels would be duplicated on different CUDA blocks at the same time. By switching between kernels, all threads coincide. The chief challenge in switching among kernels is to minimize the associated cost. Given that objective, the result of the calculations of each kernel is stored in the global memory of the GPU, and no data would be exchanged at each switching step between the host and the device. Consequently, the time required for switching among kernels is negligible. The process of each kernel is described below.

In the proposed method, first, the fitness of the chromosomes in the initial population is calculated concurrently by the first parallel kernel. In this kernel, each thread calculates the fitness of a chromosome. Next, to produce a new generation, the operators of crossover, mutation, and fitness functions are applied concurrently by the second kernel. In this kernel, each thread should form a new child chromosome using dissimilar operators of the GA. The third parallel kernel is responsible for the selection operator. This operator selects the finest chromosomes of the current generation to be passed to the next generation. In this kernel, each selects a chromosome. This cycle is repeated, and at the end of each cycle, the condition of the termination of the generation of the new population is checked. If the condition of the termination of rounds is met, this cycle stops, and the best answer is returned from the populations as a result. Otherwise, the second and third kernels that perform the creation of a new generation, as well as the selections, will continue running to the end of the pre-specified number of iterations. In the next section, we inspect the performance of the proposed method on TSP.

4. IMPLEMENTATION AND PERFORMANCE EVALUATION

In this section, first, the hardware characteristics of the computer system and the values of the diverse parameters of the GA, are described. Then, the performance of the proposed kernels is examined based on several metrics.

4. 1. Conditions and Environments of Implementation The experiments were executed on an Intel (R) Core i5-7600 3.50GHz computer with 8 GB of random-access memory that was equipped with an NVIDIA GeForce GTX 960 graphics card. This GPU has 1024 cores, and its base and boost clocks are 1,127MHz, and 1,178MHz, respectively. The frameworks used in this implementation were created by C++ CUDA 8.0 (V8.0.61) for many-core GPU and TBB version 2018 for multi-core CPU. The number of threads in the CUDA platform and multi-core CPUs was set equal to the number of chromosomes and the number of cores, respectively. Hence, the number of cores in the TBB-based parallel kernel is remarkable. To investigate this effect, we executed the TBB-based parallel kernels on dual-core and quad-core CPUs.

Crossover and mutation probability were set to 0.8 and 0.02, respectively. The probability of selection operator, providing that the operator may select a sample with less fitness, is 0.8. Note that the crossover operator is a single-parent and single-point one, and the mutation operator is of the movement type.

The standard datasets of PKA379, rbx711, and xit1083 from VLSI data were used in implementing the TSP. The datasets consisted of 379, 711, and 1083 geographical regions of cities [38]. TSP was solved with 1024 to 16384 populations by using 100, 200 to 3000 rounds of the GA. The number of offsprings produced in the crossover was between 10%-50% of the size of the preliminary population. In the next section, we analyze the results which were acquired from ten times repeatition of the experiments.

4. 2. Performance Evaluation To assess the proposed methods and compare their performances, we analyze the influence of parameters like population size, number of generations, number of crossover-mutation, and chromosomes size on the performance of each method.

Evaluation metrics are the running time of the GA in the proposed methods and the speedup of the parallel versions of the serial method. First, we examine the cost of switching between the proposed kernels over the CUDA platform in diverse scenarios. Table 1 shows the time required for solving TSP with 370 cities using the proposed set of tertiary kernels with different sizes of population and different numbers of generations. For each case, the time required for switching among kernels, as well as their execution time, is reported. Table 2 shows the switching and execution time of the kernels in 100 generations, with dissimilar numbers of cities and different sizes of the population. In these two tables, the ratio of offsprings is 50% of the population. As explained in Section 3, since there is no data transmission between kernels and generations (from GPU to CPU and vice versa), growing the number of generations has not any effect on the switching time. But by increasing the number of cities and the size of the population, switching time rises due to the transfer cost of the preliminary population from CPU to GPU. However, the switching time is small compared with the kernel execution time. The sum of the switching time and the time of kernel computations represent the total computation time for a parallel kernel in the CUDA platform.

The plots in Figure 2 demonstrate the effect of increasing the size of the initial population and the number of generations on the running time of the serial method, the parallel method on the CUDA platform, and the parallel methods formed by exploiting TBB on dual-core/quad-core CPUs. In this experiment, the size of the new population generated from the crossover operation is 50 % of the initial population. By growing the number of generations, the running time of TSP is grown in all methods.

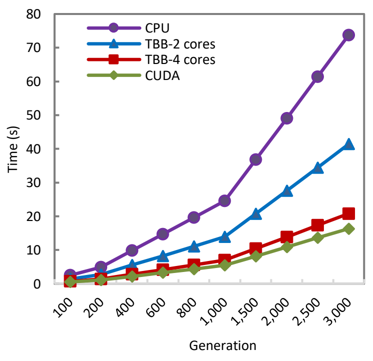
A notable point in this experiment is the effect of the size of the population on the execution time of the CUDA-based parallel code as compared to the TBB-

TABLE 1. Switching and kernel computation time (ms) in CUDA- 379 cities (pka379)

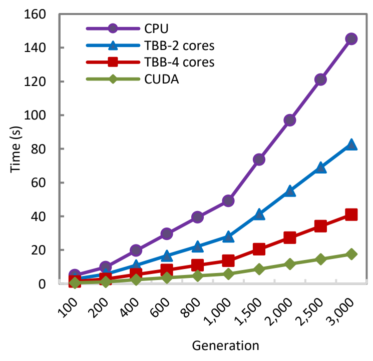
Generation	Population							
	1024		5120		10240		16384	
	Switch	Kernel	Switch	Kernel	Switch	Kernel	Switch	Kernel
100	0.195128	548.545	0.78577	600.4875	1.50617	573.9525	2.3287	672.3125
400	0.18966	2181.9425	0.79366	2388.99	1.52645	2261.02	2.37945	2655
1000	0.196124	5450.35	0.78552	5960.375	1.45712	5638.075	2.29706	6624.425
3000	0.194644	16340.6	0.78178	17871.925	1.48930	16972.8	2.32974	19876.325

TABLE 2. Switching and kernel computation time (ms) in CUDA- 100 generations

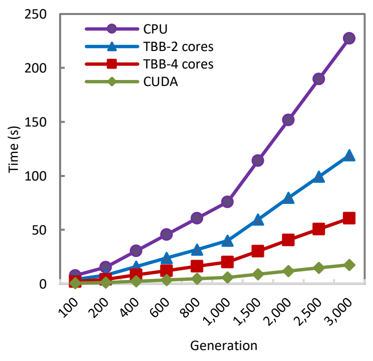
Cities	Population							
	2048		9216		32768		131072	
	Switch	Kernel	Switch	Kernel	Switch	Kernel	Switch	Kernel
379 (pka379)	0.341525	588.4825	1.3636	567.645	5.520225	1756.985	19.6385	6283.225
711 (rbx711)	0.586467	1089.205	2.5615	1164.0825	8.821775	3291.925	34.5882	11756.775
1083(xit1083)	0.866655	1677.325	3.8843	1787.5275	12.68052	5065.65	52.2602	18439.525



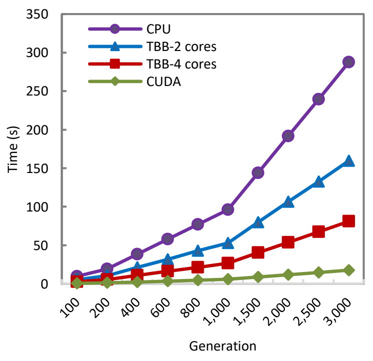
a. 1024 Population



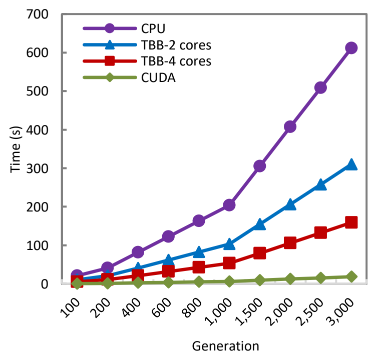
b. 2048 Population



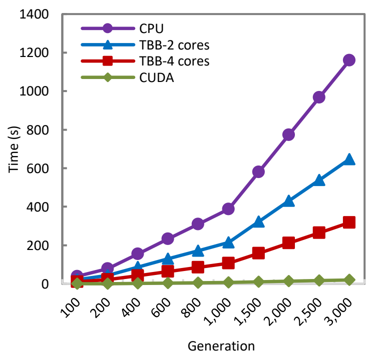
c. 3072 Population



d. 4096 Population



e. 8192 Population



f. 16384 Population

Figure 2. The running time of the algorithm over different numbers of generations and different sizes of population

based parallel codes running on dual-core and quad-core machines. As shown in Figure 2(a), the running time of CUDA-based kernel is worse than TBB-based kernel on dual-core and quad-core CPUs. In this experiment, all resources of CPUs have been completely exploited while in the GPU, only up to 1024 threads have been used.

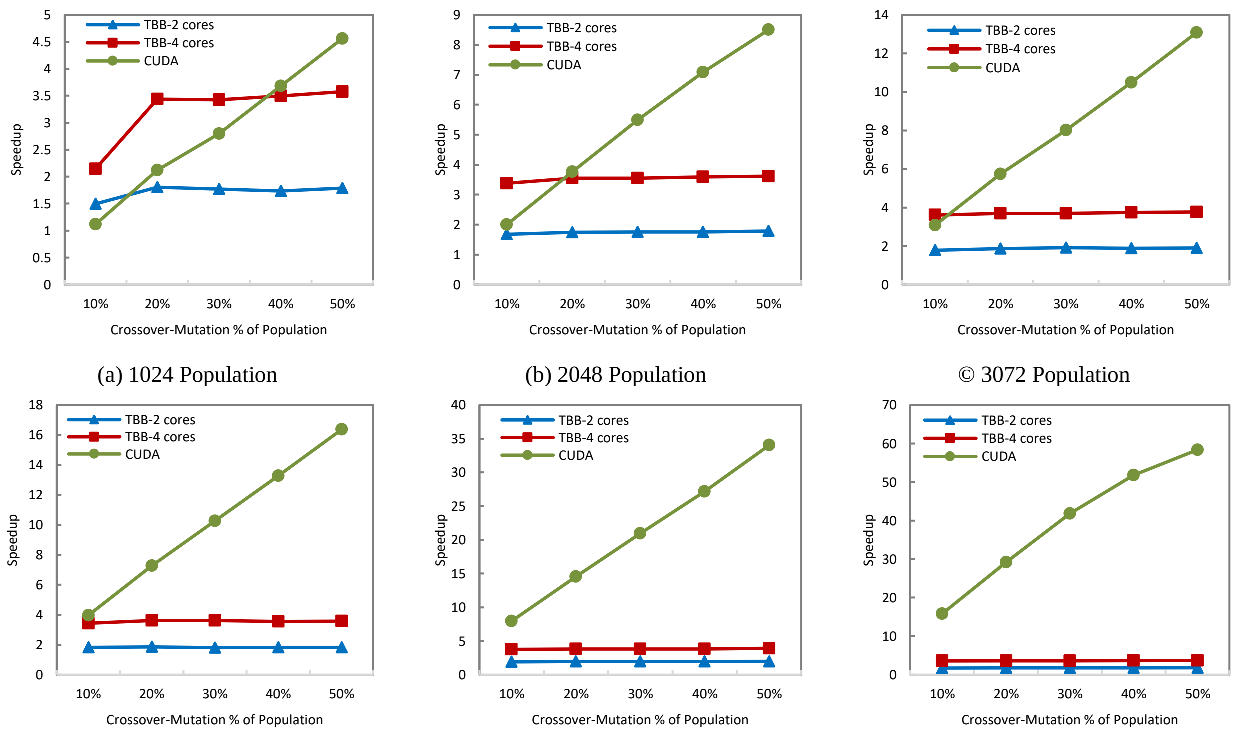
Growing the size of populations increases the concurrency, and improves the performance of CUDA kernels. For example, in the case of having a population with the size of 4096, the CUDA-based parallel TSP code has been better than TBB-based TSP. The degree of this dominance reaches a maximum when the size of the population hits 16384. In this state, the maximum number of GPU resources, or equivalently the maximum number of threads, have been used synchronously.

Figure 3 demonstrates the plots of speedup of three parallelization methods, namely TBB-based TSP kernel on dual-core and Quad-core CPUs and CUDA-based GPU kernel concerning the sequential code in different sizes of population and offsprings created by crossover. As a common setting in this experiment, the upper bound of the number of generations of the GA is set to 100. In all cases, the speedup of TBB on four cores is more than the speedup of TBB on two cores. In addition, the overall rate of the speedup has not been changed with the size of the population and the number of offsprings. This is while in the CUDA-based TSP kernel, the speedup has changed with both of these parameters. The reason is that both parameters affect the utilization of the resources of GPU. The maximum speedup values obtained in the execution of TBB-based TSP on dual-core and quad-core CPUs are 1.98 and 3.92, respectively, while the maximum speedup of CUDA-based TSP on 1024 cores is 58.35.

The effect of changing the number of chromosomes on the execution time of the parallel TSP codes is illustrated in Figure 4. In the corresponding experimentation, the number of generations and the number of offsprings resulting from each crossover process is a constant value while the size of the population is variable. In this experiment, three different datasets with 379, 711, and 1083 cities have been used. The size of offsprings is set to be 50% of the primary population for 100 generations.

As shown in Figure 4, in all cases, the CUDA-based parallel code is the fastest as compared to the other methods. This state hits when CUDA computes the GA solution of TSP with 16,384 population; in this case, each thread computes a chromosome. As the population grows, each thread should compute more than one chromosome, which increases the running time. This is shown in the speedup plots of Figure 4. Another notable point in this experiment is the impotence of changes in the number of genes on a chromosome (number of cities) in the overall execution time of the three parallel kernels of the GA.

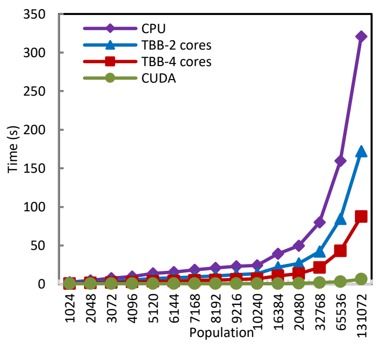
Regarding the ability of CUDA to associate the maximum number of threads for computations of a kernel, the use of CUDA seems to produce higher performance; but, in the following, the results of our experiments show that the TBB-based implementation exploits the computational resources of the system more efficiently.



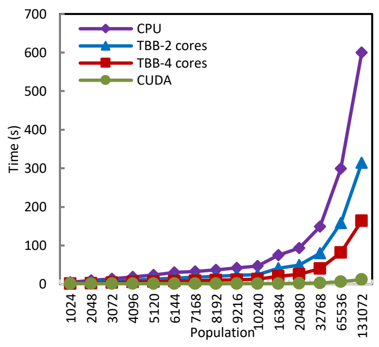
(a) 1024 Population (b) 2048 Population © 3072 Population

(d) 4096 Population (e) 8192 Population (f) 16384 Population

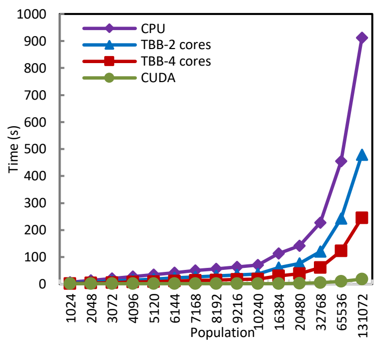
Figure 3. The speedup of the parallel methods for different ratios of offsprings on diverse populations in a TSP with 379 cities



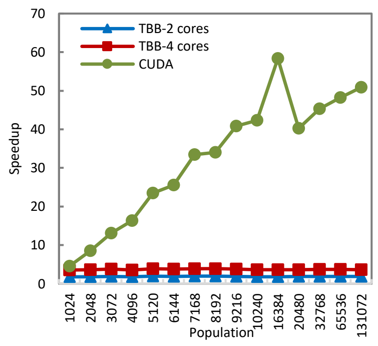
(a) Time, 379 cities (pka379)



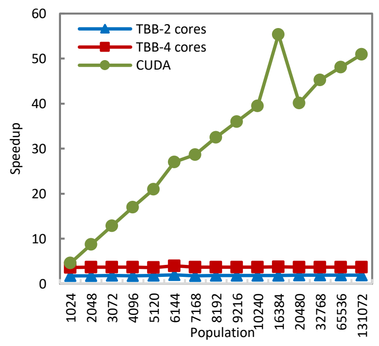
(b) Time, 711 cities (rbx711)



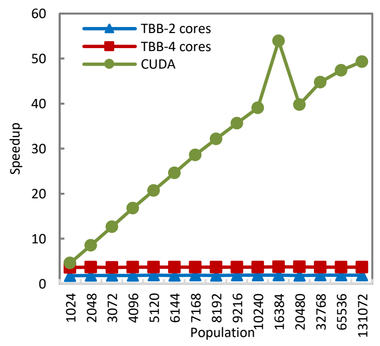
(c) Time, 1083 cities (xit1083)



(d) Speedup, 379 cities (pka379)



(e) Speedup, 711 cities (rbx711)



(f) Speedup, 1083 cities (xit1083)

Figure 4. The impact of population size on the speedup of parallel methods

4. 3. Efficiency Evaluation and Comparison

To intuitively show the effectiveness of the proposed parallelization method, the efficiency of the proposed kernels is computed and compared in Table 3 with the recent advanced methods. As shown in Table 3, the efficiency of the proposed parallel GA on a GPU with 1024 cores is 0.057, which is higher than all opponents. Also, the efficiency of the proposed parallel GA code on a quad-core CPU using the TBB platform is the highest as compared to the other parallel GA solutions of TSP on multi-core CPUs. Also, the efficiency of multi-core parallelization of the GA solution of TSP, using the TBB platform is 0.9975, which is significantly more than that of the CUDA-based parallelization. This result shows that the proposed parallelization method not only outstands any present many-core parallelization of GA solution of TSP, but also confirms that the proposed method could more effectively use the computational resources of the multi-core CPUs and consequently reach higher efficiency.

TABLE 3. Comparing the efficiency of state-of-art parallelizations of GA solution of TSP

Method	Year	Reference	# of Cores	Speed up	Efficiency
Multi-core	2013	Zhu [17]	4	2.55	0.6375
	2019	Saxena [26]	4	2	0.5
	2020	**Proposed Method**	**4**	**3.99**	**0.9975**
Many-core	2011	Fujimoto [18]	240	13.3	0.055
	2011	Chen [19]	448	1.44	0.003
	2016	Kang [21]	2048	6.014	0.003
	2017	Moumen [22]	1280	25.07	0.019
	2019	Saxena [26]	128	5.66	0.044
	2020	**Proposed Method**	**1024**	**58.35**	**0.057**

5. CONCLUSION

In this paper, a novel method for efficient parallelization of genetic algorithms on multi-core and many-core systems was presented. The proposed method efficiently executes parallel kernels on the multi-core and many-core systems, each corresponding to a different operator of GA, by using TBB and CUDA platforms, respectively. The proposed method was tested for parallelizing a GA-based solution of the TSP on CUDA and TBB platforms with the same settings, including the same number of primary population and generations as well as the same ratio of population created by crossover and mutation operators on the same data set. The performance of these two platforms was assessed based on different metrics including the running time and speedup of the parallel GA over each of them.

From the results, we have drawn the following conclusions that crucially represent the real and novel contribution of our work. First, the highest speedup of the parallel algorithm on the GPU, the quad-core, and the dual-core processor are 58.35, 3.99, and 1.99, respectively. Second, the performance of a parallel GA on a GPU-like many-core processor is much higher than that of a multi-core processor, but in a low initial population, parallelization resources in multi-core processors are more efficiently utilized than in the GPU-like many-core systems. Third, the efficiency of the proposed parallelization of the GA solution of TSP on a CUDA-based many-core platform is the highest as compared to state-of-art parallel solutions. Fourth and the most important finding is that the proposed TBB-based parallelization of the GA solution of TSP achieves the highest level of efficiency in exploiting the computational resources of the system for parallel execution of GA.

The CPU/GPU clusters have recently been considered as high-performance accelerators for computation-intensive programs. Therefore, future studies would study how to adapt the proposed parallelization method of GA to best use the resources of the GPU cluster computations.

6. REFERENCES

1. Fathollahi-Fard, A.M., Hajiaghaei-Keshteli, M., and Tavakkoli-Moghaddam, R., 'The Social Engineering Optimizer (Seo)', ***Engineering Applications of Artificial Intelligence***, (2018), Vol. 72, 267-293. https://doi.org/10.1016/j.engappai.2018.04.009
2. Fard, A.F. and Hajiaghaei-Keshteli, M., 'Red Deer Algorithm (Rda); a New Optimization Algorithm Inspired by Red Deers' Mating', in, International Conference on Industrial Engineering, IEEE., (2016). https://doi.org/10.1007/s00500-020-04812-z
3. Mohammadzadeh, H., Sahebjamnia, N., Fathollahi-Fard, A., and Hahiaghaei-Keshteli, M., 'New Approaches in Metaheuristics to Solve the Truck Scheduling Problem in a Cross-Docking Center', ***International Journal of Engineering-Transactions B: Applications***, 2018, Vol. 31, No. 8, 1258-1266. https://doi.org/10.5829/ije.2018.31.08b.14
4. Fathollahi-Fard, A., Hajiaghaei-Keshteli, M., and Tavakkoli-Moghaddam, R., 'A Lagrangian Relaxation-Based Algorithm to Solve a Home Health Care Routing Problem', ***International Journal of Engineering Transactions A: Basics,*** Vol. 31, No. 10, (2018), 1734-1740. https://doi.org/10.5829/ije.2018.31.10a.16
5. Hajiaghaei-Keshteli, M., Abdallah, K., and Fathollahi-Fard, A., 'A Collaborative Stochastic Closed-Loop Supply Chain Network Design for Tire Industry', ***International Journal of Engineering Transactions A: Basics,*** Vol. 31, No. 10, (2018), 1715-1722. https://doi.org/10.5829/ije.2018.31.10a.14
6. López-González, A., Campaña, J.M., Martínez, E.H., and Contro, P.P., 'Multi Robot Distance Based Formation Using Parallel Genetic Algorithm', ***Applied Soft Computing***, (2020), Vol. 86, p. 105929. https://doi.org/10.1016/j.asoc.2019.105929
7. Munroe, S., Sandoval, K., Martens, D.E., Sipkema, D., and Pomponi, S.A., 'Genetic Algorithm as an Optimization Tool for the Development of Sponge Cell Culture Media', ***In Vitro Cellular & Developmental Biology-Animal***, Vol. 55, No. 3, (2019), 149-158. https://doi.org/DOI: 10.1007/s11626-018-00317-0
8. Sin, I.H. and Do Chung, B., 'Bi-Objective Optimization Approach for Energy Aware Scheduling Considering Electricity Cost and Preventive Maintenance Using Genetic Algorithm', ***Journal of Cleaner Production***, Vol. 244, (2020), 118869. https://doi.org/10.1016/j.jclepro.2019.118869
9. Yasmin, S.: 'Linear Colour Image Processing in Hypercomplex Algebra Guided by Genetic Algorithms', University of Essex, 2019
10. Lima, A.A., de Barros, F.K., Yoshizumi, V.H., Spatti, D.H., and Dajer, M.E., 'Optimized Artificial Neural Network for Biosignals Classification Using Genetic Algorithm', J***ournal of Control, Automation and Electrical Systems***, Vol. 30, No. 3, (2019), 371-379. https://doi.org/10.1007/s40313-019-00454-1
11. Rajagopalan, A., Modale, D.R., and Senthilkumar, R., Optimal Scheduling of Tasks in Cloud Computing Using Hybrid Firefly-Genetic Algorithm', Advances in Decision Sciences, Image Processing, Security and Computer Vision, (Springer, 2020) ISBN: 978-3-030-24318-0. https://doi.org/10.1007/978-3-030-24318-0_77
12. Rajesh, K., Visali, N., and Sreenivasulu, N., Optimal Load Scheduling of Thermal Power Plants by Genetic Algorithm', *Emerging Trends in Electrical, Communications, and Information Technologies*, (Springer, 2020) ISBN: 978-981-13-8942-9. https://doi.org/10.1007/978-981-13-8942-9_33
13. Arif, M.H., Li, J., Iqbal, M., and Liu, K., 'Sentiment Analysis and Spam Detection in Short Informal Text Using Learning Classifier Systems', ***Soft Computing***, Vol. 22, No. 21, (2018), 7281-7291. https://doi.org/10.1007/s00500-017-2729-x
14. Talbi, E.-G., 'A Unified View of Parallel Multi-Objective Evolutionary Algorithms', ***Journal of Parallel and Distributed Computing***, Vol. 133, (2019), 349-358. https://doi.org/10.1016/j.jpdc.2018.04.012
15. Nayak, S. and Panda, M., Hardware Partitioning Using Parallel Genetic Algorithm to Improve the Performance of Multi-Core Cpu', ***Advances in Intelligent Computing and Communication***, (Springer, 2020) ISBN: 978-981-15-2774-6
16. Giap, C.N. and Ha, D.T., 'Parallel Genetic Algorithm for Minimum Dominating Set Problem', in, Computing, Management and Telecommunications (ComManTel), 2014 International Conference on, (IEEE, 2014). https//doi.org/10.1109/ComManTel.2014.6825598
17. Zhu, J. and Li, Q., 'Application of Hybrid Mpi+ Tbb Parallel Programming Model for Traveling Salesman Problem', in, Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, (IEEE, 2013). https://doi.org/10.1109/GreenCom-iThings-CPSCom.2013.408
18. Fujimoto, N. and Tsutsui, S., 'A Highly-Parallel Tsp Solver for a Gpu Computing Platform', in, International Conference on Numerical Methods and Applications, (Springer, 2010). https://doi.org/10.1007/978-3-642-18466-6_3
19. Chen, S., Davis, S., Jiang, H., and Novobilski, A., Cuda-Based Genetic Algorithm on Traveling Salesman Problem', Computer and Information Science 2011, (Springer, 2011). https://doi.org/10.1007/978-3-642-21378-6_19
20. Sánchez, L.N.G., Armenta, J.J.T., and Ramírez, V.H.D., 'Parallel Genetic Algorithms on a Gpu to Solve the Travelling Salesman Problem', Difu100ci@ Revista en Ingeniería y Tecnología, UAZ, 2015, 8, (2). https://doi.org/10.1007/978-3-662-45049-9_96

21. Kang, S., Kim, S.-S., Won, J., and Kang, Y.-M., 'Gpu-Based Parallel Genetic Approach to Large-Scale Travelling Salesman Problem', ***The Journal of Supercomputing***, 2016, Vol. 72, No. 11, 4399-4414. https://doi.org/10.1007/s11227-016-1748-1
22. Moumen, Y., Abdoun, O., and Daanoun, A., 'Parallel Approach for Genetic Algorithm to Solve the Asymmetric Traveling Salesman Problems', in, Proceedings of the 2nd International Conference on Computing and Wireless Communication Systems, (ACM, 2017) https://doi.org/10.1145/3167486.3167510
23. Cekmez, U., Ozsiginan, M., and Sahingoz, O.K., 'Adapting the Ga Approach to Solve Traveling Salesman Problems on Cuda Architecture', in, Computational Intelligence and Informatics (CINTI), 2013 IEEE 14th International Symposium on, (IEEE, 2013) https://doi.org/10.1109/CINTI.2013.6705234
24. Radford, D. and Calvert, D., 'A Comparative Analysis of the Performance of Scalable Parallel Patterns Applied to Genetic Algorithms and Configured for Nvidia Gpus', ***Procedia Computer Science***, Vol. 114, (2017), 65-72. https://doi.org/10.1016/j.procs.2017.09.009
25. Li, C.-C., Lin, C.-H., and Liu, J.-C., 'Parallel Genetic Algorithms on the Graphics Processing Units Using Island Model and Simulated Annealing', ***Advances in Mechanical Engineering***, Vol. 9, No. 7, (2017), 12-25. https://doi.org/10.1177%2F1687814017707413
26. Saxena, R., Jain, M., Sharma, D., and Jaidka, S., 'A Review on Vanet Routing Protocols and Proposing a Parallelized Genetic Algorithm Based Heuristic Modification to Mobicast Routing for Real Time Message Passing', ***Journal of Intelligent & Fuzzy Systems***, Vol. 36, No. 3, (2019), 2387-2398. https://doi.org/10.3233/JIFS-169950
27. NVIDIA. NVIDIA CUDA (Compute Unified Device Architecture) Programming Guide, Available: http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf, (Accessed July 2020).
28. Jam, S., Shahbahrami, A., and Ziyabari, S., 'Parallel Implementation of Particle Swarm Optimization Variants Using Graphics Processing Unit Platform', ***International Journal of Engineering Transactions A: Basics***, Vol 30, No. 1, (2017), 48-56. https://doi.ir/10.5829/idosi.ije.2017.30.01a.07
29. Yip, C.M. and Asaduzzaman, A., 'A Promising Cuda-Accelerated Vehicular Area Network Simulator Using Ns-3', in, *P*erformance Computing and Communications Conference (IPCCC), 2014 IEEE International, (IEEE, 2014) https://doi.org/10.1109/PCCC.2014.7017048
30. Kim, C.G., Kim, J.G., and Lee, D.H., 'Optimizing Image Processing on Multi-Core Cpus with Intel Parallel Programming Technologies', ***Multimedia Tools and Applications***, Vol. 68, No. 2, (2014), 237-251. https://doi.org/10.1007/s11042-011-0906-y
31. Reinders, J. 'Intel threading building blocks: outfitting C++ for multi-core processor parallelism', O'Reilly Media. Inc, 2007.
32. Hougardy, S. and Wilde, M., 'On the Nearest Neighbor Rule for the Metric Traveling Salesman Problem', ***Discrete Applied Mathematics***, (2014). https://doi.org/10.1016/j.dam.2014.03.012
33. Groba, C., Sartal, A., and Vázquez, X.H., 'Solving the Dynamic Traveling Salesman Problem Using a Genetic Algorithm with Trajectory Prediction: An Application to Fish Aggregating Devices', ***Computers & Operations Research***, Vol. 56, (2015), 22-32. https://doi.org/10.1016/j.cor.2014.10.012
34. Hussain, A. and Muhammad, Y.S., 'Trade-Off between Exploration and Exploitation with Genetic Algorithm Using a Novel Selection Operator', ***Complex & Intelligent Systems***, (2019), 1-14. https://doi.org/0.1007/s40747-019-0102-7
35. Hassanat, A., Prasath, V., Abbadi, M., Abu-Qdari, S., and Faris, H., 'An Improved Genetic Algorithm with a New Initialization Mechanism Based on Regression Techniques', ***Information***, Vol. 9, No. 7, (2018), 167. https://doi.org/10.3390/info9070167
36. Doughabadi, M.H., Bahrami, H., and Kolahan, F., 'Evaluating the Effects of Parameters Setting on the Performance of Genetic Algorithm Using Regression Modeling and Statistical Analysis', ***Journal of Industrial Engineering***, *University of Tehran*, (2011), 61-68.
37. Contreras-Bolton, C. and Parada, V., 'Automatic Combination of Operators in a Genetic Algorithm to Solve the Traveling Salesman Problem', ***PloS One***, Vol. 10, No. 9, (2015), e0137724-e0137724. https://doi.org/10.1371/journal.pone.0137724
38. VLSI-TSP-Collection, Available: http://www.math.uwaterloo.ca/tsp/vlsi/index.html, (Accessed July 2020).

Persian Abstract

چکیده

موازی‌سازی کارآمد الگوریتم‌های ژنتیک روی بسترهای multi-threading یا many-threading موجود به دلیل دشواری زمانبندی منابع سخت‌افزاری با توجه به همزمانی نخ‌ها، موضوعی چالش برانگیز است. در این مقاله، برای حل این مشکل یک روش جدید ارائه شده است که الگوریتم ژنتیک را با طراحی سه کرنل همزمان که هرکدام تعدادی از عملگرهای موثر وابسته به یکدیگر از الگوریتم ژنتیک را اجرا می‌کنند موازی می‌سازد. روش پیشنهادی را می‌توان به راحتی با استفاده از بسترهای Compute Unified Device Architecture (CUDA) و Threading Building Blocks (TBB) برای اجرا روی پردازنده‌های many-core و multi-core تطبیق داد. برای استفاده بهینه از منابع ارزشمند این نوع پردازنده‌ها در اجرای موازی الگوریتم ژنتیک، پردازش‌های نخی که یکی از کرنل‌های سه‌گانه را اجرا می‌کنند، توسط مکانیسم جابجایی پرسرعتی هماهنگ می‌شوند. روش پیشنهادی، برای موازی‌سازی مسئله فروشنده دوره‌گرد مبتنی بر الگوریتم ژنتیک با استفاده از پلتفرم‌های CUDA و TBB با تنظیمات یکسان آزمایش شده است. نتایج، کارایی روش پیشنهادی در موازی‌سازی الگوریتم ژنتیک را روی واحد پردازش گرافیکی و همچنین روی واحد پردازش مرکزی تایید می‌کنند. علاوه بر این، در مسئله‌های ژنتیک با جمعیت اولیه متوسط، با اینکه زمان جابجایی بین کرنل‌های واحد پردازش گرافیکی بسیار ناچیز است اما، الگوریتم ژنتیک موازی مبتنی بر TBB از منابع بطور موثرتری استفاده می‌کند.