Adapting the GA Approach to Solve Traveling Salesman Problems on CUDA Architecture

Ugur Cekmez
Computer Engineering Department
Yildiz Technical University
Istanbul, Turkey
ucekmez@yildiz.edu.tr

Mustafa Ozsiginan
Aeronautic and Space Technologies Institute
Turkish Air Force Academy
Istanbul, Turkey
mustafaozsiginan@gmail.com

Ozgur Koray Sahingoz
Computer Engineering Department
Turkish Air Force Academy
Istanbul, Turkey
sahingoz@hho.edu.tr

Abstract— **The vehicle routing problem (VRP) is one of the most challenging combinatorial optimization problems, which has been studied for several decades. The number of solutions for VRP increases exponentially while the number of points, which must be visited increases. There are 3.0×10^64 different solutions for 50 visiting points in a direct solution, and it is practically impossible to try out all these permutations. Some approaches like evolutionary algorithms allow finding feasible solutions in an acceptable time. However, if the number of visiting points increases, these algorithms require high performance computing, and they remain insufficient for finding a feasible solution quickly. Graphics Processing Units (GPUs) have tremendous computational power by allowing parallel processing over lots of computing grids, and they can lead to significant performance gains compared with typical CPU implementations. In this paper, it is aimed to present efficient implementation of Genetic Algorithm, which is an evolutionary algorithm that is inspired by processes observed in the biological evolution of living organisms to find approximate solutions for optimization problems such as Traveling Salesman Problem, on GPU. A 1-Thread in 1-Position (1T1P) approach is developed to improve the performance through maximizing efficiency, which then yielded a significant acceleration by using GPUs. Performance of implemented system is measured with the different parameters and the corresponding CPU implementation.**

Keywords—TSP; parallel GA; CUDA; GPU; High Performance; 1T1P

I. INTRODUCTION

The Vehicle Routing Problem (VRP) [1] is an important logistics problem, which has been studied for several decades, and it typically tries to find the lowest cost of the combined paths for one or more vehicles in order to facilitate delivery of products from a depot location to a number of destinations. The total cost is closely associated with the length of distribution path; therefore, if a logistic firm can reduce the length of this path, it can increase the firm's profit and market share. Traveling Salesman Problem (TSP) [2][3] is probably the most widely studied VRP, and it is accepted as a standard testbed for new algorithmic ideas to solve VRP.

The number of solutions for TSP increases exponentially while the number of points, which must be visited increases. Therefore, it is impossible to control all possible solution in a TSP problem with a large number of visiting points whose number of possible solutions is shown in Table 1. As depicted in this table there are 3.0×10^{64} different solutions for 50 visiting points.

TABLE I. THE NUMBER OF POSSIBLE SOLUTIONS ACCORDING TO THE NUMBER OF VISITING POINTS

# of points	*# of possible solutions*
1	1
5	120
10	3.6×10^{6}
20	2.4×10^{18}
50	3.0×10^{64}
100	9.3×10^{157}

To solve this type of complex problems, evolutionary algorithms are accepted as good solutions. Genetic Algorithm (GA) is a widely used evolutionary algorithm, which uses efficient search technique to reach optimal/near optimal solutions to any kind of optimization problems, especially for the NP-hard ones [4]. Based on adapting the principles of natural selection and genetic operations, GAs use evolutionary operators in a loop where the selection phase fires the fittest individuals to generate new populations, and crossover and mutation phases protect the diversity between these populations. Change over the generations yields better fitness values at each iteration until it is eligible through the limitations such as time, computational resources, optimality degree, etc. [5].

NP-hard problems require much time and computational power to solve complex and large. Since some of the particular calculations in the structure of the GAs are convenient to be computed independent from others, such as re-constructing the meta-heuristic. These independent computations can be easily parallelized and distributed among multi-cores and multi-processors, which brings very high performance in time [6][7].

Besides running the parallel approach only on traditional Central Processing Unit (CPU) architectures that have several high frequency cores, for the time being General Purpose Graphic Processing Units (GPGPU) with thousands of low frequency cores together with CPUs are started to be used as massively parallel computing platforms. Providing highly effective data parallelism with yet developing powerful task

978-1-4799-0197-5/13/$31.00 ©2013 IEEE

parallelism, GPGPU cores with multiple threads running at the same time can easily accelerate GAs [6].

In this study, taking advantage of the power of the GPGPUs, especially using NVIDIA's CUDA architecture, it is aimed to adopt the GA approach to solve some complex Traveling Salesman Problems (TSP) that are one of the most practiced NP-hard problems.

The rest of the paper is organized as follows; in the next section, the background information about genetic algorithms, TSP and CUDA is given. In Sections 3, details of the the proposed parallel implementation is described. Experimental results and related works are detailed in Section 4 and Section 5 respectively. Finally, in Section 6 some conclusions and future works are drawn.

II. Background

A. Genetic Algorithms

A genetic algorithm is a kind of evolutionary algorithms which simulates natural process of biological evolution of living organisms to find approximate solutions for optimization problems. In Genetic Algorithm (GA) approach, to find an optimal solution, there exist 4 primary steps in general [4]. The first step produces an initial population of set of candidate elements or patterns (according to the structure of the problem). The objective here is to be able to start the process of searching for the solutions by manipulating these candidates [5]. The underlying population can be produced by either a uniform distribution or by applying some criteria at hand. Each candidate solution in the population is called individual which has a constant fitness value expressing itself whether an optimal value is met. The fitness value is then going to be used at each iteration to separate the fittest individuals from the ones having inappropriate values, which might route the algorithm away from the optimal solution. So the fittest individuals are selected, and the others are discarded. This step is called selection. After the selection phase, new individuals are produced by the composition of the selected ones through crossover step. In this step, several options exist including order-1, arithmetic, partially mapped (PMX), cycle and edge recombination. In order to protect the diversity of the new generations, mutation step is usable where a certain probability is taken as a reference and each individual is mutated. This step also includes several options of mutating the individuals where insertion, swapping, inversion, scramble or random sliding techniques involve [6].

B. Traveling Salesman Problem

In TSP, the aim is to find the shortest path by passing every given point exactly once and reach a desired point which generally is the starting point the tour begins. In order to complete the task, it must be ensured that the route has the least cost. The TSP is a classical NP-hard combinatorial optimization problem, which states that the solution can be found in polynomial time interval which increases as the number of points to be visited is increased.

C. Underlying CUDA Architecture

As today's hardware systems are getting to be much more improved, the GPGPU systems are used to accelerate the existing algorithms with new approaches, each having tremendous performance speed ups. This gain routes the research to a whole new era where the big computations are not pain anymore.

CUDA [10] is the new future proof approach to the parallelism models of this era with ever-improving resources including hardware and software as well as the knowledge the developers are gaining.

In this study, nVidia's latest mobile GPU, GeForce 740M and CUDA SDK 5.0 are used. The GPU has 384 cores each having 810 MHz of frequency and access to total 2 GBs of global memory. Together with the GPU, the underlying hardware includes Intel i5-3317U 4x1.70 GHz CPU with 4 GBs of RAM. The operating system the implementations are run is 64-bit Ubuntu 12.04.

In the proposed model in GPU, full system resources are used where all the threads are dynamically distributed among 384 cores by the CUDA itself and kept busy as much as possible.

III. Proposed Algorithm For Parallel GA Model

The algorithm basically has two initial and two main cyclic computations synchronized and handled in parallel. Providing random number seeds for further use, creating the initial population and evolving the corresponding population steps are handled by the number of threads that are exactly equivalent to the population size. The data structures are specially considered so that both the data-parallelism approach is satisfied in CUDA and the performance gain is maximized by the usage of the threads.

A. Random number seeds

As the GA approach basically requires randomness at many stages, being able to provide high quality random number sequences yields better results since the randomness keeps the solution away from converging to local optimum points and encourages the perturbation probability to jump to the global optimal.

In order to produce random numbers other than standard C++ library's *rand()* function, nVidia's CUDA Random Number Generation Library (cuRAND) is used. The library uses GPU's available cores to maximize the parallel throughput. In this case, the seeds are stored in an array of length equal to the population size and each index is assigned to a thread. These seeds are then used to generate random numbers with uniform distribution.

Random number generation process is shown in Fig. 1.

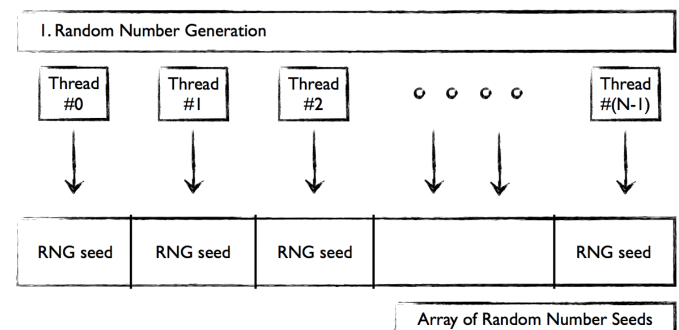

Fig. 1. Random Number Generation Process described briefly

B. Initial Population

Similar to the cycle of creating random numbers, each individual in the population, here namely chromosomes, is handled by a CUDA thread.

In order to avoid the strict rules of the fix-sized array and to be able to determine the number of individuals in the population at runtime, a pointer is used to represent a sequence of individuals. Each thread that has an ID number from 0 to size - 1 takes the initial chromosome and shuffles it by using the relevant seeder from the previous step and put it into the position considering its ID. The process is shown in Fig. 2.

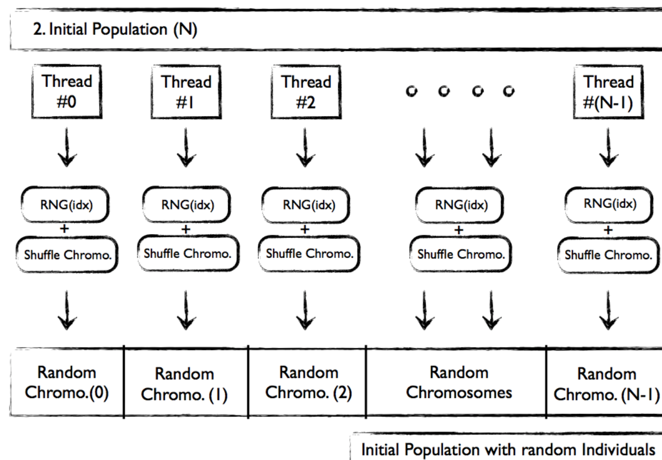

Fig. 2. Creating the initial population to start the evolution process

C. Fitness Function

The fitness function is an important aspect in developing a genetic algorithm approach to a specific problem. Designing a good fitness function is a crucial issue which is the guidance factor of the evolution of new solutions. For TSP-like problems generally total distance between starting point and ending point is used as the fitness value. However, in this calculation, lower fitness value means better solution. Therefore, ın this parallel solution, we select the same fitness function to reach an acceptable solution.

D. Sorting Individuals

Keeping the population sorted by the fitness value of individuals is a challenging step here. Since each thread will manipulate the population and the execution time of a thread cannot be known, elitism part of the GA, if needed, can only be applied after a population is sorted. So it is evaluated to be a distinct step and applied before the evolution process.

Number of sorting mechanisms can be used here. In this study, the Thrust library that is also in nVidia's CUDA SDK is chosen to sort the chromosome structures in a parallel fashion. Thrust is a powerful GPU-accelerated library to work with custom data structures, and it provides very simple high-level interface that can be easily adapted to any kind of project that requires working with a vector of structures.

E. Evolving the Population

This stage involves 5 (five) sub-steps including elitism, selection, crossover, mutation and local optimization. Each step is sequentially computed by a thread.

The proposed model is basically described in Fig. 3.

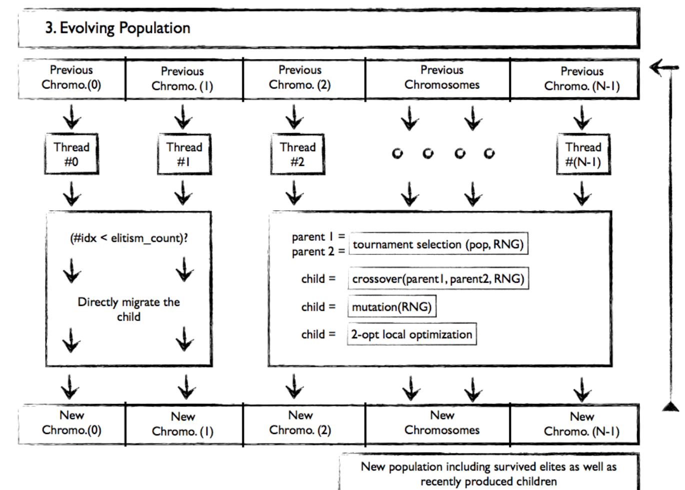

Fig. 3. 1T1P approach where each thread is responsible for 1 index in the resulting population. The termination criteria are the limited number of iterations

Since the sorting mechanism reorders the previous population and considering that each thread handles only the relevant individual of the population, elitism in this case is as simple as migrating the first N individual to the next generation (Elitism step). For the aim of simplicity elitism rate is kept constant for the experiments. If a thread does not have the ID smaller than the predetermined elitism number, then the tournament selection is applied to the population, and 2 individuals are chosen to be the parents (Selection step).

In order to maximize the parallelism and taking 1T1P approach into account, only 1 child is produced by the crossover. Each thread randomly chooses either 1-point or 2-point crossover methods with the probability of 0.5 while producing new individual (Crossover step). Then the child, with some probability given, is mutated. The mutation phase here is the swap of randomly chosen 2 points of the individual. Swapping procedure iterates several times according to the length of a chromosome.

As the population size gets bigger, mutated children keep the diversity in a high degree thus provides avoiding converging to the local optimal solutions (Mutation step). When the length of the individuals gets bigger, crossed paths take place with high probability. Therefore, a local optimization step may produce better individuals by optimizing these conflicts between points in an individual. Here, the 2-opt local optimization approach is taken into account (Local optimization step) [9].

After these steps, the corresponding thread calculates the new fitness value of the child, if needed, and puts it into the new population. The whole evolution process iterates until meeting the ending criterion.

In Genetic Algorithm approach, to make an evolution and to increase the quality of the solution, a new generation is produced from the old population by using genetic operators iteratively. If an ending criterion does not exist, this evolution continues infinitely. Therefore, some ending criterion, which mainly depends on the type of the problem, must be defined, and producing new solutions must end when this criterion is

met. This criterion mainly depends on the type of the optimization problem.

In the simple implementation of GA, a general stopping criterion is defined as the "maximum number of iterations". Because we mostly focused on the parallelizing of the algorithms and increase the performance of the system, at this stage we used the same ending criterion in out implementation.

IV. Experimental Results

In testing a parallel implementation of genetic algorithm, selection of the parameters is a very important decision and this has a direct impact on the solution quality and execution time. In the proposed system, these parameters are set as shown in Table 2.

TABLE II. Experimental Parameters of Genetic Algorithm

Parameters	Value
# of Control Points in TSP	52 / 76 / 100 / 225
Population Size	4096 / 8192 / 16384
Parent Selection	Tournament (select 10 solution and get the best)
Local Search Type	2-opt
Local Search Rate	1
# of Iteration (as ending criterion)	10
Elitism	First 32 chromosomes in each generation
Mutation type	Swap
Mutation rate	0,15
Crossover type	One Point Crossover

The experimental comparisons of the 1-Thread in 1-Position (1T1P) model in GPU with the relevant implementation of the CPU show that the time measured for solving the TSP instances requires significantly less time in GPU version as the problem size gets bigger. As shown in the Table 3, the GPU results have also smaller error rate comparing to the CPU version. One of the reasons of having better results is because the random number generation library uses the computational power of the GPU to produce better randomness comparing to the standard C random function. Providing better randomness yields better diversity when the selection and mutation parts are involved

In the proposed model, number of threads in the GPU is equal to the number of individuals in the population. Each thread is responsible of producing a new child for the nth position in the resulting population where N is the corresponding thread ID. As the number of threads used for the computation increases, much more latency takes place in the GPU because of the synchronization points in the implementation. The population size for each different population is set as the product of 32 due to maximize the throughput according to the warp size of the underlying hardware.

TABLE III. Time Measurements for the Serial and Corresponding Parallel Implementation of the Model

TSPLIB Name	Population Size	Result After 10 iteration on CPU	Result After 10 iteration on GPU	Expected Result	Total Computation Time on CPU in sec	Total Computation Time on GPU in sec	Speed up Rate
berlin52	4096	7697	7544	7542	586	1	532
berlin52	8192	8237	7544	7542	2284	2	1097
berlin52	16384	7678	7544	7542	8891	5	1777
eil76	4096	572	554	538	868	2	507
eil76	8192	590	550	538	3342	3	996
eil76	16384	571	546	538	16441	8	1955
kroA100	4096	21516	21294	21282	1138	2	478
kroA100	8192	21383	21285	21282	4336	5	915
kroA100	16384	21466	21285	21282	21604	12	1829
tsp225	4096	4525	4025	3916	2612	7	366
tsp225	8192	4518	3955	3916	9723	16	622
tsp225	16384	4280	3952	3916	47843	38	1258

While the serial version of CPU implementation converges to the sub optimal result in 52-point TSP instance with 16384 individuals in population in 8891 seconds with 10 iterations, the parallel model solves the problem to the optimal solution in 5 seconds with x1777 performance speed up. Similar performance improvements with better solutions exist in the other tests, too.

Overall results show that the GPU version of the implementation has a speed up varying from x366 to x1955 as well as better convergence to the optimal solutions depending on the input size.

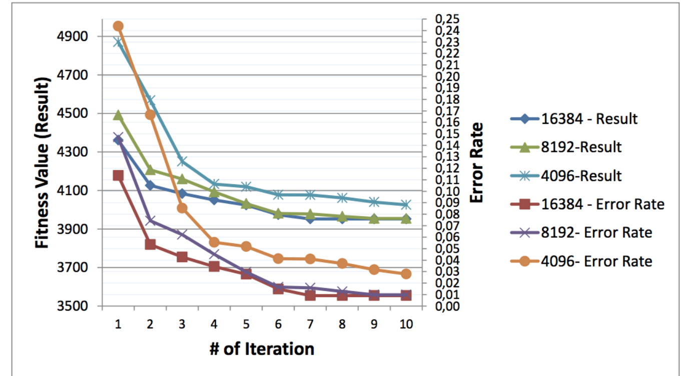

Fig. 4. The GPU results calculated at each iteration and corresponding error measurements of the tsp225 TSPLIB with 16384, 8192 and 4096 populations

As a detailed example, one of the used TSPLIB instances in the study, tsp225 with 16384, 8192 and 4096 populations, is shown in Fig.4. Through ten iterations, the error rate quickly goes to an acceptable value as the fitness converges to the expected optimal result.

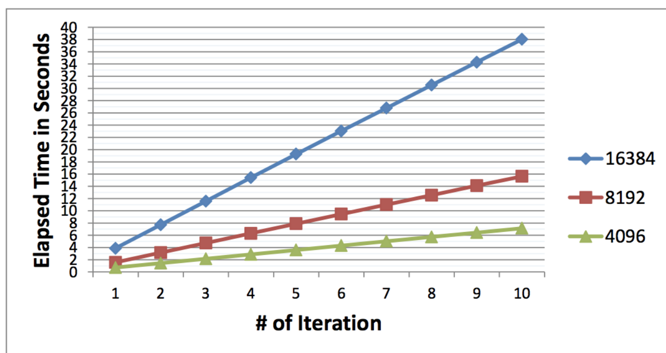

Fig. 5. The GPU results calculated at each iteration and corresponding time measurements of the tsp225 TSPLIB with 16384, 8192 and 4096 populations

As shown Fig.5, the calculation times of distinct populations at each iteration take almost identical thus yielding a linear chart in overall.

V. Related Studies

In order to have a basic knowledge of the techniques that are attempted so solve the GAs in parallel fashion, a few of the significant methods developed to parallelize the GAs are examined in [5] by collecting and analyzing some relevant publications. Most studied parallel methods include multi-population implementations where the algorithm is coarse-grained or distributed such that each population is handled separately and then has some interconnection by either exchanging or migrating the appropriate individuals to synchronize. The name "multiple-deme" is also applicable for this type of parallelization as the paper indicates. Other than that, two types of parallelization are also expressed in this study where 1) single population master-slave type is which a master administers the whole process cycle and distributes the evaluation of the fitness among the slaves since most time-consuming event is to compute the fitness and 2) single population fine-grained type consists of a spatially-structured population that includes many small populations or individuals interconnecting and synchronizing with only their neighborhood. There also exist some hybrid models those combining the single-population and multi-population techniques. The survey touches upon the generic island model that requires less communication in between populations. Early studies about parallelization of the GAs are generally based on simple GAs, and the development stages of multiple-deme parallel GAs shows that these kinds of techniques seem to run the serial GAs on multiple processors with some adequate connections between separate populations. Besides the experimental developments of algorithms so far, the early theoretical studies raise the question whether the parallel GAs results better than the equivalent serial implementations. Considering the comparisons in overall, some problem-specific criteria must be met to reach optimality along with time performance.

After learning the basic concepts, the next level is to convert the approaches so that they fit into the GPGPU architectures. For this aim, [6] studies and suggests basic understanding of the parallel GAs in GPUs. The main constraint of the GPUs, at that time, is that they are data-parallel systems in the first place which actually what the GAs need. With this respect, using massively parallel computing power yields thousands of times faster results since the GAs compute identical stages at each individual. The greatest limiting factor here is the probable bottleneck of transmission of the data from GPU to CPU and vice versa. Moving the whole algorithm, so it only works in GPU is another option which may yield better performance but unfortunately was not performed for the time being of this study. As a technical detail, in order to map the GA, in this paper, a coarse-grained island model is selected where every island corresponds to a block in GPU and each individual is represented and evaluated by a thread in the underlying block. To be able to provide integrity, the island model requires migrations of some fittest individuals from an island to another one through the device memory. For the aim of preventing performance loss, migrations are done asynchronously. The selection method in this implementation is chosen to be the tournament method, and the arithmetic crossover is used for the new individuals along with the Gaussian mutation technique. The performance gain in this study is relatively high, and it yields optimal solutions in an acceptable time where it clearly indicates the steps taken.

An experimental parallel approach to the GAs on CUDA architecture is studied by [4] that uses the GA to solve the TSP at hand. As a crossover step, two routes in a TSP cannot be directly exchanged one to one between the individuals as the binary chromosomes do; instead a custom crossover that is based on sequence order of individuals is produced to keep the fitness value fine. The importance of the mutation operation as a local optimizer is also mentioned here. In order to converge the individuals to an optimal fitness value, 2-opt mutation technique that fixes the crosses in the individuals is used, and a simple selection method is applied to ones those have best fitness values. As data management segment, this study keeps the most accessed variables such as the distances of the points of TSP in shared memory to get faster results. The computations those have no task dependencies such as mutation, fitness value and selection are separately handled by the threads in parallel. Those have task dependencies are analyzed, and the synchronization points are determined as they need barriers in the necessary points of the implementation. As test data, randomized and clustered routes are given to the algorithm, and the results are compared. The comparison is made between the sequential C code and the corresponding parallelized version. Performance gain exists but stays low since the implementation uses 1 over 16 computing resources on the GPU.

Another concrete parallel GA implementation to solve the TSP by using order crossover and 2-opt mutation is done by [8]. This study parallelizes some parts of each individual in addition to computing the separate parts among populations. The inner parallelization is of crossover and mutation parts. The difference in the crossover part, which provides of being parallelized, is to produce one individual with the composition of two parents. Although parallelizing the 2-opt heuristic either, it is seen as the limiting factor since it is insufficient for a large population size and another heuristic is decided to be used as a future work. The experimental comparisons show that the performance gain reaches up to 24 times than the equivalent CPU implementation in the overall result.

VI. Conclusion

Parallel Genetic Algorithms have been successfully applied to solve some complex combinatorial optimization problems like Traveling Salesman Problem to reduce the required time to find a feasible solution. The use of Graphics Processing Units (GPUs) has become increasingly popular in last years, and they are not only used for graphical computations but also used for general purpose parallel computation. Therefore, they are increasingly used in high performance computing applications with parallel implementation. In this paper, it is aimed to develop a GPU based parallel genetic algorithm on TSP for decreasing solution time. Performance of implemented system is measured with the different size of population due to its relevance with the number of parallel execution on GPU. At the same time, GPU performance is also compared with the corresponding CPU implementation, and results showed that GPU computations brings high performance gain varying from x366 to x1955 for parallelized computations.

As a future work, it is intended to test system on large scale TSP systems with a number of salesmen, and to increase performance gains by modifying the current algorithm. Some sample TSP-like scenarios as detailed in [11-13] are expected to be a good choice for testing this type of large scale problems.

References

[1] T. Vidal, T.G. Crainic, M. Gendreau, and C. Prins, “Heuristics for multi-attribute vehicle routing problems: A survey and synthesis,” European Journal of Operational Research, vol. 231, no. 1, pp. 1–21, 2013.

[2] D.L. Applegate, R.E. Bixby, V. Chv´atal, W.J. Cook, “The Traveling Salesman Problem: A Computational Study”, Princeton University Press, Princeton, 2007.

[3] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys, “The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.” Wiley, Chichester, 1985.

[4] S. Chen, D. Spencer, J. Hai, and A. Novobilski, "CUDA-Based Genetic Algorithm on Traveling Salesman Problem." In Computer and Information Science,Springer, pp. 241-252, 2011.

[5] D. S. Knysh and V. M. 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.

[6] P. Pospíchal, J. Jaros, and J. Schwarz. "Parallel genetic algorithm on the CUDA architecture." In Applications of Evolutionary Computation, Springer, pp. 442-451, 2010.

[7] A. Munawar, M. Wahib, M. Munetomo, and K. Akama, “A Survey: Genetic Algorithms and the Fast Evolving World of Parallel Computing,” in 10th IEEE International Conference on High Performance Computing and Communications, 2008. HPCC’08, pp. 897–902, 2008.

[8] Fujimoto, N., and Shigeyoshi Tsutsui. "A highly-parallel TSP solver for a GPU computing platform." In Numerical Methods and Applications, Springer ,2011 pp. 264-271.

[9] K. Rocki, , and R. Suda, "Accelerating 2-opt and 3-opt local search using GPU in the travelling salesman problem." 2012 International Conference on High Performance Computing and Simulation (HPCS), IEEE, 2012.

[10] M. Garland, S. Le Grand, J. Nickolls, et al, “Parallel computing experiences with CUDA”, IEEE Micro, vol. 28, pp.13–27, 2008.

[11] O. K. Sahingoz, “Large scale wireless sensor networks with multi-level dynamic key management scheme,” Journal of Systems Architecture, vol. 59, no. 9, pp. 801–807, 2013.

[12] O. K. Sahingoz and N. Erdogan, “Dispatching Mechanism of an Agent-Based Distributed Event System,” International Conference on Computational Science-ICCS 2004, Lecture Notes in Computer Science, vol. 3036, pp 184-191, 2004.

[13] O. K. Sahingoz, “Generation of Bezier Curve-Based Flyable Trajectories for Multi-UAV Systems with Parallel Genetic Algorithm,” Journal of Intelligent & Robotic Systems, pp. 1–13, Available online 12 October 2013, Online ISSN 1573-0409, doi: 10.1007/s10846-013-9968-6. 2013.