An Optimized GPU-Accelerated Route Planning of Multi-UAV Systems Using Simulated Annealing

Seval Capraz, Halil Azyikmis, and Adnan Ozsoy

***Abstract*—Usage of multiple unmanned aerial vehicles (UAV) in a certain mission makes flight route planning more complicated and slower. In order to obtain better performance, in the literature, most of the researchers propose using evolutionary algorithms and artificial intelligence approaches based on heuristics as optimization techniques. In addition to this, parallel programming approaches increase the computation performance. Therefore, this study focuses to discuss and solve the route planning problem for multi-UAV systems by using optimization techniques based on an evolutionary algorithm: simulated annealing. The travel cost and execution time are downsized in this work by optimization on algorithm and code. We implemented CPU based parallel solution to compare results with the GPU-accelerated one. The efficiency and the effectiveness of our parallelized and optimized solution is demonstrated through simulations under different scenarios. The results show that our optimized GPU based parallel solution for route planning problem is up to 1.6 times faster than serial and parallel CPU solutions. Moreover, our optimized GPU solution is better on cost than other solutions. It is shown that our GPU based approach is the fastest one and increases performance thanks to the massive parallelization capabilities of GPUs.**

***Index Terms*—GPU programming, parallel programming, route planning, simulated annealing.**

I. Introduction

Unmanned aerial vehicles (UAV) have various usage areas from delivery of goods to battlefield use. As the UAVs' cost decreased with current technological developments, multiple minimized UAVs can be used for better performance instead of using a single large UAV. This kind of multiple usage of UAVs makes the flight route planning problem for these systems more complicated.

In multi-UAV systems, total travel distance should be divided wisely among UAVs. Each UAV have to travel at minimum cost and their cost have to close each other if they have equal resources. In literature, Simulated Annealing (SA) algorithm is used many times for shortest path problem. In our case, multiple UAVs are traversing the predefined target locations -waypoints- with total minimum cost which means total distance traversed by each UAV required to be minimum. It is a kind of NP-hard problem. This kind of problems are hard to solve and take a lot of time and energy. The problem is to find an acceptable solution that is near-optimal solution rather than the best solution.

In order to simulate the problem in best way we have used Traveling Salesman Problem Library (TSPLIB [1]) which is created and presented by University of Heidelberg, Germany. We used this library as a dataset because it is used in many study in literature as well as in Turker *et al.*(2016) [2]. Therefore we compare our results with other studies.

Dealing with these kind of complex algorithms is hard. The SA algorithm is difficult to implement and very slow. It is one of the best algorithms to find best route with best cost among other algorithms for many UAVs. For this reason, we focus on this algorithm and how to implement it faster. The modern supercomputing shows that GPUs (Graphics Processing Units) are very good accelerators speeding up all sorts of tasks from very hard problems to these kind of algorithms. Why is CPU not enough for it? Because GPUs offer many benefits. Architecturally, the CPU is composed of just a few cores with lots of cache memory. This cache memory can handle a few software threads at a time. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously. It is ideal for algorithms which do a lot of little jobs like comparison. GPU-based route planning of multi-UAV system speeds the overall calculation.

In this study, firstly, we implemented SA algorithm in CPU. It works only on CPU and it is slow. There are many proposed solutions in literature which uses only CPU and they are all slow. We need to minimize the calculation time to find best rouse plan for many UAVs. The best route plan means less in cost. It is so difficult to find best solution so we only want to find an acceptable one. There are also a few studies which uses GPU to run SA. It helps to minimize the run time of the algorithm. We realized the implemented solutions have lack of optimization. GPU has its own infrastructure. If we optimized the algorithm to fit well in GPU, we can gain more time. The optimization is so important and it improves both cost and run time in good manner.

This paper is organized as follows. In the second section, related works are discussed. In the third section, background information of the proposed solution algorithm with simulated annealing (SA) are defined and algorithms for serial and parallel run are given. In the fourth section, serial

Manuscript received February 8, 2019; revised January 11, 2020.

Seval Capraz is with Ante Grup Bilisim Ticaret A.S., Ankara, Turkey. She is also with the Department of Computer Engineering, Hacettepe University, Ankara, Turkey (e-mail: seval.capraz@antegrup.com.tr).

Halil Azyikmis and Adnan Ozsoy are with the Department of Computer Engineering, Hacettepe University, Ankara, Turkey (e-mail: hazyikmis@hacettepe.edu.tr, adnan.ozsoy@hacettepe.edu.tr).

and parallel codes are reviewed in CPU platform. Also, solution on GPU is given and optimization techniques are described. In the fifth section, results obtained from the experiments are presented and compared. In the last section, the paper is concluded with brief information of what we did and what we can do in future work.

II. Related Work

Simulated Annealing (SA) algorithm is proposed in [3] in 1983. It is an old algorithm however it is very powerful for Traveling Salesman Problem. There are many works which use SA for route planning problem in literature. In 1999, Pant Rajkumar *et al.* [4] proposed aircraft configuration and flight profile optimization using SA. In 2002, Rostami S *et al.* [5] proposed a SA for multi-route cluster tools. In 2012, Taylor Christine *et al.* [6] proposed dynamically generating operationally acceptable route alternatives using SA. S. Zaghloul Soha [7] proposed a parallel solution with SA algorithm for flight route planning problem in 2017. Alsafi Eman [8] proposed comparison of parallel simulated annealing on SMP and parallel clusters for same problem in 2018.

In 2014 Hossain Roksana *et al.* [9] proposed GPU enhanced path finding for an UAV and they claim that GPU code works 4.8 times faster than serially implemented code. In 2016, Cekmez Ugur *et al.* [10] proposed multi-UAV path planning with parallel genetic algorithms on GPU. In [10], the area is partitioned with K-means clustering and then the problem is solved in each cluster with parallel genetic algorithm approach on CUDA architecture.

Turker *et al*. (2016) [2] proposed a solution in SA algorithm by using a simple heuristic. In Turker's solution, UAVs start at the same point which is center of the area. Different UAVs visit different regions, therefore they try to keep cost at minimum. The algorithm is run as serial on CPU and as parallel on GPU.

There are other solutions based on different algorithms for the same problem. For example, flight route planning problem for a UAV can be approached using parallelized Ant Colony Optimization (ACO) algorithm, a BAT algorithm, A* algorithm, RRT algorithm or genetic algorithms on CUDA platform. In TSP-type problems, finding the best solution generally requires testing all the search space and this seems to be impractical. In our case, with N waypoints, search space consists of N! solutions. Adding one more waypoint to the system has a considerable effect on the overall system performance because of the increasing number of elements in the search space from N! to (N+1)!. When dealing with this kind of NP-hard problems, we can use different optimization algorithms to search and find an acceptable (near-optimal) solution rather than to find the best solution.

In this study, first of all, we have optimized the existing serial CPU solution proposed by Turker *et al.*(2016) [2] by using optimization techniques and also created brand-new parallel CPU solution to see its speed and compare it with GPU solution.

III. Background

For the problem of multiple UAVs' flight route planning, the algorithm of simulated annealing is given in detail in the following subsections.

A. Simulated Annealing

SA is one of the most used optimization algorithms in especially TSP-like problems. In SA, the objective function is used instead of the energy of a material. The algorithm is a simulation of decreasing temperature. It uses random move to jump instead of best move. It always uses the selected move if it improves the solution. There is a probability of making moves which is between 0 and 1. This probability decreases exponentially with the amount *deltaE* which is the amount of worsening move. The probability is given in (1).

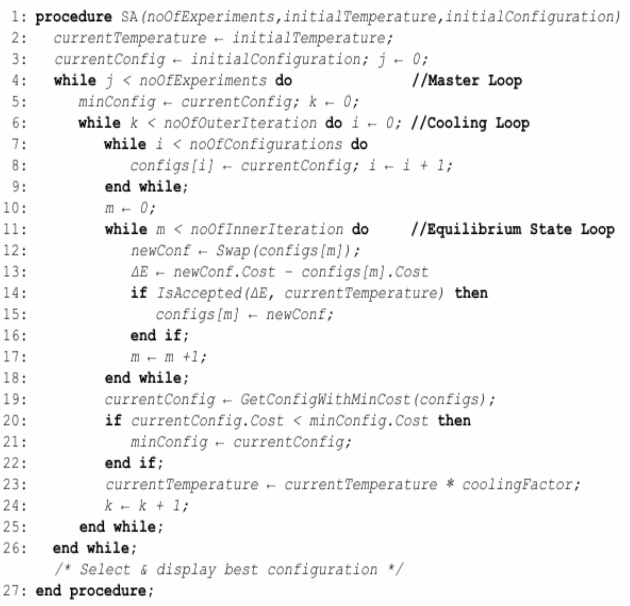
$$Probability = 1 - e^{(deltaE/kT)} \quad . \qquad (1)$$

The algorithm start with high temperature and gradually decreased temperature according to an "annealing schedule". T is temperature and if it is higher, the probability is close to 1. This means that the algorithm more likely to accept uphill move if the temperature is high. If the temperature is low, the algorithm accept less likely uphill move. This simulation is taken from annealing system of thermodynamics. The parameter *k* in (1) is Boltzmann's constant.

Greedy algorithms gets stuck at local minima. Simulated Annealing algorithms are usually better than greedy algorithms, when it comes to problems that have numerous locally optimum solutions. To escape from local optima, Metropolis acceptance function based on the Metropolis Algorithm [11] is used. The global minima and local minima is represented in Fig. 1.

Simulated annealing is very suitable for Traveling Salesman Problem if there are many waypoints to visit. Consequently, it is an ideal solution for flight route planning problem of multiple UAVs.

Algorithm 1. Simulated Annealing in CPU Proposed in [2].



```
procedure SA(noOfExperiments,initialTemperature,initialConfiguration)
  currentTemperature ← initialTemperature;
  currentConfig ← initialConfiguration; j ← 0;
  while j < noOfExperiments do                //Master Loop
    minConfig ← currentConfig; k ← 0;
    while k < noOfOuterIteration do i ← 0; //Cooling Loop
      while i < noOfConfigurations do
        configs[i] ← currentConfig; i ← i + 1;
      end while;
      m ← 0;
      while m < noOfInnerIteration do      //Equilibrium State Loop
        newConf ← Swap(configs[m]);
        ΔE ← newConf.Cost - configs[m].Cost
        if IsAccepted(ΔE, currentTemperature) then
          configs[m] ← newConf;
        end if;
        m ← m +1;
      end while;
      currentConfig ← GetConfigWithMinCost(configs);
      if currentConfig.Cost < minConfig.Cost then
        minConfig ← currentConfig;
      end if;
      currentTemperature ← currentTemperature * coolingFactor;
      k ← k + 1;
    end while;
  end while;
  /* Select & display best configuration */
end procedure;
```

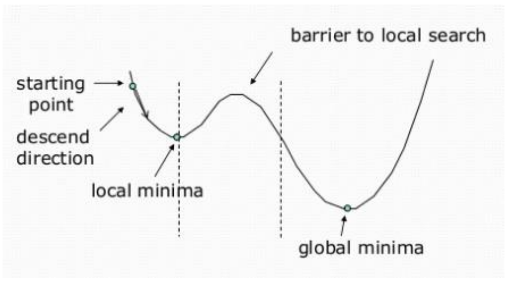


Fig. 1. The local minima vs. global minima.

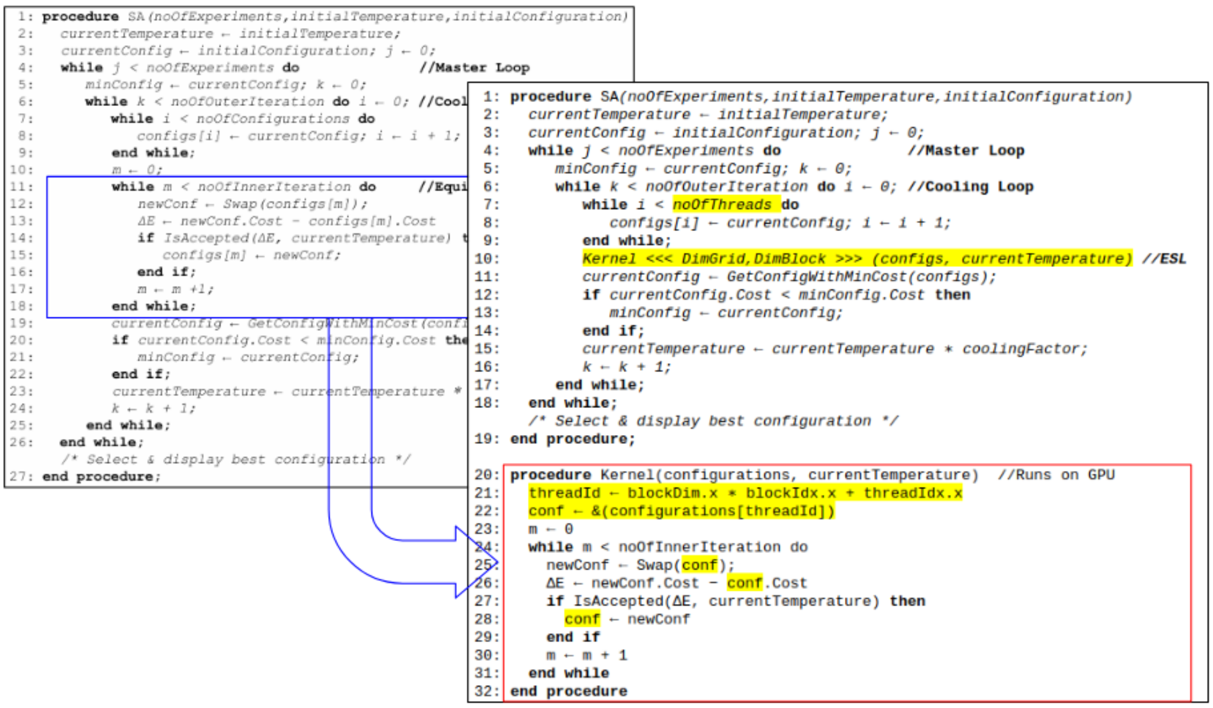
B. Implementation of Simulated Annealing using Serial and Parallel Algorithms

The simulated annealing algorithm given in Algorithm 1 can be implemented with two nested loops. The temperature starts with a higher degree and gets lower gradually. This is done in *cooling loop*. In this loop, there is another loop which name is *equilibrium state loop* (ESL). This loop tries to reach the equilibrium state. For serial implementation all the iterations run sequentially. The SA algorithm uses a random waypoint and calculates the cost and time in order to decide to use this waypoint. We need to escape from local minimum and local maximum waypoints. SA distinguishes between different local optima. We used Metropolis acceptance function in our SA implementation.

In our dataset Traveling Salesman Problem Library (TSPLIB [1]), there are numerous examples on cities on Germany with X and Y coordinates. All waypoints have 2 decimal numbers: X and Y axis values. Here are the datasets tested/used in this work:

- berlin52.tsp : 2D, 52 waypoints, integer
- berli52_3D.tsp: 3D, 52 waypoints, integer
- pr1002.tsp : 2D, 1002 waypoints, integer
- tsp225.tsp : 2D, 225 waypoints, decimal

In this dataset, cooling is done at least 100000 times. ESL runs 1000 times for each outer cooling iteration. This means the program runs many loops like 100000 × 1000 times in total. ESL is a good candidate for GPU parallelization because it does very little job with immense iteration. The comparison of CPU and GPU algorithms are given in Fig. 2. We highlighted the differences.



```
procedure SA(noOfExperiments,initialTemperature,initialConfiguration)
  currentTemperature ← initialTemperature;
  currentConfig ← initialConfiguration; j ← 0;
  while j < noOfExperiments do                    //Master Loop
     minConfig ← currentConfig; k ← 0;
     while k < noOfOuterIteration do i ← 0; //Cool
        while i < noOfConfigurations do
           configs[i] ← currentConfig; i ← i + 1;
        end while;
        m ← 0;
        while m < noOfInnerIteration do        //Equi
           newConf ← Swap(configs[m]);
           ΔE ← newConf.Cost - configs[m].Cost
           if IsAccepted(ΔE, currentTemperature) t
              configs[m] ← newConf;
           end if;
           m ← m +1;
        end while;
        currentConfig ← GetConfigWithMinCost(confi
        if currentConfig.Cost < minConfig.Cost the
           minConfig ← currentConfig;
        end if;
        currentTemperature ← currentTemperature *
        k ← k + 1;
     end while;
  end while;
  /* Select & display best configuration */
end procedure;
```

```
procedure SA(noOfExperiments,initialTemperature,initialConfiguration)
  currentTemperature ← initialTemperature;
  currentConfig ← initialConfiguration; j ← 0;
  while j < noOfExperiments do                    //Master Loop
     minConfig ← currentConfig; k ← 0;
     while k < noOfOuterIteration do i ← 0; //Cooling Loop
        while i < noOfThreads do
           configs[i] ← currentConfig; i ← i + 1;
        end while;
        Kernel <<< DimGrid,DimBlock >>> (configs, currentTemperature) //ESL
        currentConfig ← GetConfigWithMinCost(configs);
        if currentConfig.Cost < minConfig.Cost then
           minConfig ← currentConfig;
        end if;
        currentTemperature ← currentTemperature * coolingFactor;
        k ← k + 1;
     end while;
  end while;
  /* Select & display best configuration */
end procedure;

procedure Kernel(configurations, currentTemperature)  //Runs on GPU
  threadId ← blockDim.x * blockIdx.x + threadIdx.x
  conf ← &(configurations[threadId])
  m ← 0
  while m < noOfInnerIteration do
    newConf ← Swap(conf);
    ΔE ← newConf.Cost - conf.Cost
    if IsAccepted(ΔE, currentTemperature) then
      conf ← newConf
    end if
    m ← m + 1
  end while
end procedure
```

Fig. 2. The comparison of algorithms implemented on CPU and GPU. Solution on GPU is done by running Equilibrium State Loop in GPU.

We can open 1000 threads and give the jobs these threads in GPU. On the other hand, if we try to give the cooling job to GPU, each cooling job calls 1000 ESL and this exceeds the limits of a thread can do. We have to keep jobs minimum in GPU threads. We tried giving whole cooling loop function to the GPU. This method makes slower because there are a lot of iterations in nested loops. Running sufficiently large number of iterations guarantees a convergence. We need to find an acceptable solution which means we need to reach at least cost of CPU solutions.

The GPU solution slows when there are a lot of data migration between CPU and GPU. In each iteration, cooling loop needs the results of equilibrium state kernel, so that there are a lot of copy jobs between CPU and GPU. These copy jobs take a lot of time. Therefore we tried to optimize the algorithm by adding asynchronous methods instead of synchronous for transferring data. The data transfer job can be asynchronous because the used waypoints are not used again. We can copy some results while calculation is still running. This optimization provide us more speed. We also tried host-parallel option like running kernel on GPU at least number of CPU cores. It does not give us better results, therefore we did not use host parallelism.

Reference [2] uses 128 threads on GPU. We increased the number of threads from 128 to 1024. So 1 is enough for inner loop count when there are 1024 threads if the dataset has 1024 node to visit. Threads per block was 32 in previous work, we increased it to 256. So 4 blocks and 256 threads per block is used in this study. It helps to gain performance. These optimizations provide us better performance on GPU. Our optimized method is the fastest one if we compare it with [2]. The overview of proposed solution is given in Fig. 3.

To sum up, SA algorithm can be optimized in several ways. Firstly, we made the data transfer jobs asynchronous. Secondly, we increased the number of threads which works on GPU to run the algorithm and use whole capacity of GPU card. These optimizations improved the performance. The results are given in next section.

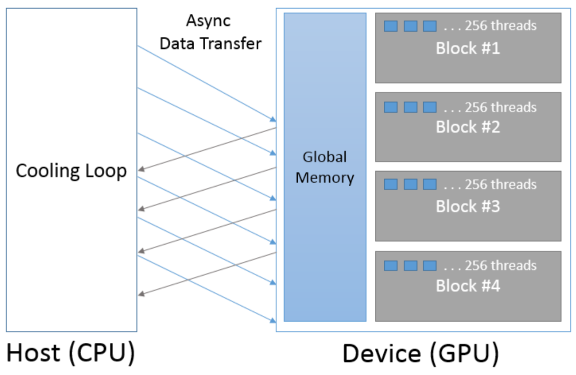


Fig. 3. Overview of proposed solution.

IV. Experimental Evaluation and Results

A. Configuration Settings

The solution is implemented both in sequential and parallel mode on CPU and only parallel mode on GPU. We needed to compare the CPU parallel solution with GPU solution. Sometimes CPU parallelization is enough for problems. So we need to be sure GPU-accelerated solution is the best solution. The hardware and operating system specifications are presented in Table I. Both of the serial and parallel algorithms are implemented, built and tested on Ubuntu 64bit environment (version 16.04) using with NVIDIA CUDA COMPILER (CUDA Version 9.1.85) in order to keep the chance of making comparisons for GPU implementation.

TABLE I: System Specifications

Operating System	Ubuntu 16.04 LTS x86_64
Processor	AMD Phenom(tm) II X6 1090T Processor x 6 Cores
RAM	16GB DDR3 1333 MHz
Compiler	NVCC (Nvidia Cuda Compiler CUDA Version 8.0.61) and g++ 5.4.0
NVIDIA Graphics Card	GeForce GTX 750 Ti
NVIDIA Driver Version	375.26
CUDA Runtime version	8.0.61
CUDA Capability	5.0

B. Results of Serial Algorithm

The results are taken for serial run on CPU when there are 5 experiments. 3 UAVs are used and pr1002 is selected as a test data in dataset. Average cost is 384,634. Average time is 25,356.5 milliseconds. This means it generates results in 25.5 seconds.

In serial test, the second experiment produced the best result according to execution time. The dataset is cached before first run, so that the second experiment produce result on lower time. The computation time in this specific run is the lowest one and required 25,206.9 milliseconds to calculate this result. The last execution produces the best computation cost and its execution time is 25,341.7 milliseconds. However, because of serial implementation, there were nearly 100 seconds elapsed for previous runs (there are four runs before last execution to find the optimum solution at the last run. Please note that, this solution might not be the global optimum. Average acceptance rate is 1.4, average worsening move acceptance rate is 0.8. Graphical representation of the solution is shown in Fig. 4.

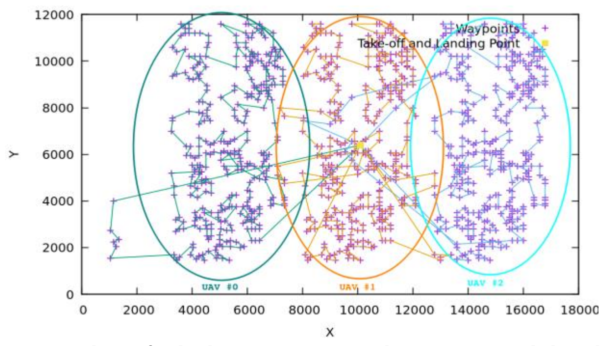


Fig. 4. Best solution for the dataset pr1002.tsp with 3 UAVs in serial algorithm.

C. Results of Parallel Algorithm

Results are taken for parallel run on CPU when there are 5 experiments and 6 threads. Average cost is 358,648. Average time is 24,135 milliseconds.

In the parallel test, the best execution time is 23,719 milliseconds. The longest execution time is 25,198.9 milliseconds. Average acceptance rate is 0.8, average worsening move acceptance rate is 0.2. It diminishes the cost from ~384K to ~358K. The results shows that 6-threaded parallel solution gives a little bit better results than serial solution because increasing number of threads from 1 to 6 does not effect on execution time but travel cost. We call ESL function 6 times rather than 1 and it tries to find better travel route 6000 times rather than 1000 times.

D. Results of GPU Based Solution

In GPU based solution, each experiment takes nearly 15 seconds. These results are better than both serial and parallel implementation in CPU, which takes nearly 25 seconds for each run. After running parallel code on GPU, the total distance traversed cost and computation time is calculated on 5 experiments. Average cost is 344,681. Average time is 15,949.9 milliseconds.

In the parallel test on GPU, the best cost is 341,429 and best time is 15,755.8 milliseconds. The last execution is produced the best result according to total distance traversed cost. The longest execution time is 16,698.3 milliseconds. The cost is better than the result of CPU based solutions. Average cost is 344,681 on GPU while average cost is 358,648 on CPU parallel solution. The average time is 15,9 seconds which is the best time because average time is 25.3 seconds in serial and 24.1 seconds in parallel run on CPU. The comparison of the results is given in next section.

V. Comparison of Results

According to the results given previous section, we can calculate speedup rate of tests according to (2). *Ts* is time of serial run and *Tp* is time of parallel run.

$$\text{Speedup Rate} = Ts \,/\, Tp \,. \qquad (2)$$

In order to compare the results of both serial and parallel implementations against different number of experiments, we have presented the Fig. 5. Based on the statistical data, parallel implementation on GPU executes 1.6 times faster than the serial implementation as given in Fig. 6. It shows that GPU based solution is faster than CPU based solutions.

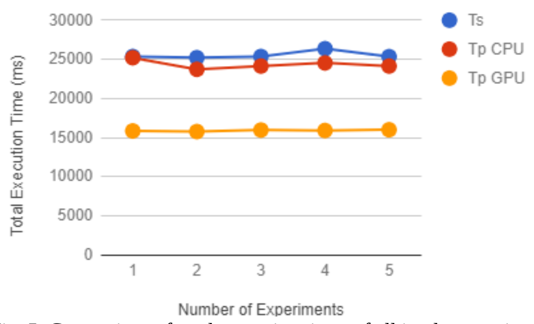
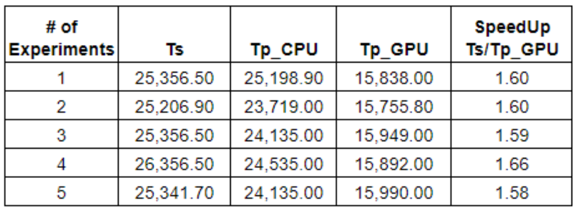


Fig. 5. Comparison of total execution times of all implementations.



# of Experiments	Ts	Tp_CPU	Tp_GPU	SpeedUp Ts/Tp_GPU
1	25,356.50	25,198.90	15,838.00	1.60
2	25,206.90	23,719.00	15,755.80	1.60
3	25,356.50	24,135.00	15,949.00	1.59
4	26,356.50	24,535.00	15,892.00	1.66
5	25,341.70	24,135.00	15,990.00	1.58

Fig. 6. Comparison of speeds and speedup of Ts/Tp_GPU.

The algorithm is run on CPU with different number of waypoints and different number of UAVs to see its performance. The tests for different number of UAVs give approximately same results. There are minor differences. However it becomes more difficult to process as the number of waypoints increases. Number of waypoints affects the process time exponentially. So, the time is very important factor for systems which have more than 1000 waypoints.

Genetic algorithm used for this task in Cekmez Ugur *et al.*(2016) [10] finds the paths in 32 seconds for 4 UAVs and 1002 waypoints on GPU. Our solution finds the route in 15 seconds in our environment. Therefore our parallel solution of SA on GPU is far faster than genetic algorithm on GPU. Of course, in [10], they used different GPU card and different environment. We should implement the same solution on same GPU card to compare and get actual run time. This may be the future work.

If we compare the results of CPU and GPU solutions, the speedup is approximately 1.6 according to (2) as given in Fig. 6. To sum up, we can say that GPU solution is up to 1.5 times faster than the serial and parallel solution on CPU.

VI. Conclusion

In conclusion, route planning for multiple UAVs needs a lot of execution time and resources. We can decrease the time and travel cost by using parallel solutions. GPU is the latest technology to run many parallel threads. GPU accelerated route planning for multi-UAVs provides better solutions as compared to serial and parallel implementation in CPU. We can also optimize the solutions on GPU by some optimization techniques. The optimization of increasing number of threads and transferring data asynchronously can make the algorithm faster. In this study, simulated annealing (SA) algorithm is used on CUDA architecture. SA is a good choice for search algorithm because it escapes from local minima. SA is also fast and one of the best search algorithm for travelling salesman problem. It calculates the cost in order to find the shortest path. We use SA algorithm for three UAVs and more than one thousand waypoints. We inspected results of parallelization on both CPU and GPU. The statistical results provide that our optimized GPU based parallelized approach of route planning problem for multiple UAVs is nearly 1.6 times faster than CPU solutions. In addition to this, our GPU based parallel solution for SA algorithm using NVIDIA CUDA platform is the fastest solution so far thanks to the massive parallelization capabilities of GPUs and optimization techniques on GPU programming. It is the expected result that GPU solution makes algorithm faster than both the recent serial and parallel CPU-based ones. Not only the execution time diminishes but also the travel cost decreases. In future works, alternative traveling salesman problem algorithms proposed in the literature can be considered for this problem and compared with each other in order to obtain better computation performance in the context of route planning problem.

Conflict of Interest

The authors declare no conflict of interest.

Author Contributions

This research is done with all authors' contributions. The CUDA scripts are written by first and second authors. The scripts are run by all authors on different GPU cards. We selected the best GPU card. We also optimized the CUDA C++ scripts by working on it and trying many different solutions. The last author is our mentor at university and advised us the optimization techniques of GPU programming. He helped to finalize the research on each step. This paper is written mainly by first author. Some parts are mainly written by second author and edited by first author. All authors had approved the final version.

References

[1] G. T. Turker, G. Yilmaz, and O. K. Sahingoz, "GPU-accelerated flight route planning for multi-UAV systems using simulated annealing," in *Proc. International Conference on Artificial Intelligence: Methodology, Systems, and Applications*, Springer International Publishing, 2016, pp. 279-288.

[2] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, pp. 671–680, 1983.

[3] R. Pant and J. P. Fielding, "Aircraft configuration and flight profile optimization using simulated annealing," *Aircraft Design*, vol. 2, no. 4, Dec. 1999.

[4] S. Rostami and B. Hamidzadeh, “A simulated annealing technique for multi-route cluster tools,” in *Proc. IEEE International Conference on Systems, Man and Cybernetics*, 2002, vol. 7, p. 6.

[5] C. Taylor and C. Wanke, “Dynamically generating operationally acceptable route alternatives using simulated annealing,” *Air Traffic Control Quarterly*, vol. 20, no. 1, pp. 97-121, Jan. 2012.

[6] S. S. Zaghloul, “Drone route planning for military image acquisition using parallel simulated annealing,” *International Journal of New Computer Architectures and Their Applications*, vol. 7, no. 3, pp. 77-89, 2017.

[7] E. Alsafi and S. S. Zaghloul, “Comparison of parallel simulated annealing on smp and parallel clusters for planning a drone’s route for military image acquisition,” *International Journal of New Computer Architectures and Their Applications*, vol. 8, no. 1, pp. 21-33, 2018.

[8] R. Hossain, S. Magierowski, and G. G. Messier, “GPU enhanced path finding for an unmanned aerial vehicle,” in *Proc. 2014 IEEE International Parallel & Distributed Processing Symposium Workshops*, 2014, pp. 1285-1293.

[9] U. Cekmez, M. Ozsiginan, and O. K. Sahingoz, “Multi-UAV path planning with parallel genetic algorithms on CUDA architecture,” in *Proc. the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO ’16 Companion*, 2016, pp. 1079-1086.

[10] G. Reinelt, “TSPLIB—a traveling salesman problem library,” *ORSA Journal on Computing*, vol. 3, no. 4, Nov. 1991.

[11] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, “Equation of state calculations by fast computing machines,” *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087-1092, Mar. 1953.

Seval Capraz worked at Ante Grup Bilisim Tic. A.S. in Ankara, Turkey as a software engineer and researcher during this study. She is also a Ph.D. student at Hacettepe University, Department of Computer Engineering, Ankara, Turkey. She has a B.Sc. degree at TOBB University of Economics and Technology, Ankara, Turkey in 2012 and M.Sc. degree at Middle East Technical University, Ankara, Turkey in Computer Science in 2016. Her research areas are distributed and parallel computing, high performance computing with GPUs, big data, blockchain and cryptocurrencies.

Halil Azyikmis is a Ph.D. student at Hacettepe University, Department of Computer Engineering, Ankara, Turkey. He has B.Sc. degree at Bogazici University, Department of Computer Engineering in 1997, and M.Sc. degree at Selcuk University in 2006. His current research interests are big data analytics, blockchain technology and smart contracts on ethereum blockchain.

Adnan Ozsoy is an assistant professor at Department of Computer Engineering, Hacettepe University, Ankara, Turkey. He received his Ph.D. degree from School of Informatics and Computing of Indiana University, Bloomington, USA in 2014. He did his M.Sc. degree in Computer Science from University of Texas at Austin, USA in 2007, and his B.Sc. degree from Virginia Polytechnic Institute and State University, USA in 2005. His research interests include parallel programming, high performance computing with GPUs, string matching algorithms, big data problems, distributed systems, application parallelism, blockchain applications and cryptocurrencies.