Parallel Hybrid 2-Opt Flower Pollination Algorithm for Real-Time UAV Trajectory Planning on GPU

Vincent Roberge[1*], and *Mohammed* Tarbouchi[1]

[1]Royal Military College of Canada, Department of Electrical and Computer Engineering, Canada

Abstract. The development of autonomous Unmanned Aerial Vehicles (UAVs) is a priority to many civilian and military organizations. An essential aspect of UAV autonomy is the ability for automatic trajectory planning. In this paper, we use a parallel Flower Pollination Algorithm (FPA) to deal with the problem's complexity and compute feasible and quasi-optimal trajectories for fixed-wing UAVs in complex 3D environments, taking into account the vehicle's flight properties. The global optimization algorithm is improved with the addition of 2-opt local search providing a significant improvement. The proposed trajectory planner in implemented and parallelized on a multicore processor (CPU) using OpenMP and a Graphics Processing Unit (GPU) using CUDA resulting in a 9.6x and a 68.5x speedup respectively compared to the sequential implementation on CPU.

Index Terms—Flower Pollination Algorithm, Graphics Processing Unit, Parallel Programming, Trajectory Planning, Unmanned Aerial Vehicle.

1 Introduction

Unmanned Aerial Vehicles (UAVs) have gained importance in military operations for a variety of reasons. Compared to manned aircraft, UAVs are small, harder to detect, inexpensive to acquire and operate, capable of reliably performing repetitive tasks, and, perhaps most importantly, do not expose aircrew to enemy threats. Although UAVs are unmanned, they still need to be operated remotely. To alleviate this task and improve the autonomy of UAVs, researchers have developed different approaches to automatic UAV trajectory planning. Many of the earlier works used simplistic model and were limited to 2D terrains. On the other hand, modern approaches strive to include the dynamic properties of the UAV, more realistic 3D terrains and multi-objective optimization functions. In the last decade, a shift from deterministic to non-deterministic methods, primarily metaheuristics, has been observed in response to this increased complexity [1]. Metaheuristics are generic optimization algorithms that depend on the iterative improvement of one or more candidate solutions in order to arrive at a near-optimal solution. Recent examples of metaheuristics used for UAV trajectory planning are the simulated annealing algorithm [2], the differential evolution algorithm [3], the genetic algorithm (GA) [4], the particle swarm optimization (PSO) [5], the ant colony optimization [6], the central force optimization [7], the predator-prey [8], the bee colony [9], and the grey wolf [10]. Despite their effectiveness, they require significant computation power which often results in an execution time that is too long for in-flight planning. To reduce the execution time, we have previously developed in [11–12] parallel implementations of the GA and the PSO for UAV trajectory planning using multicore processors (CPU), but the performance is still too slow to allow for real-time computation.

The Flower Pollination Algorithm (FPA) is another metaheuristic that has shown in many recent studies its superiority over the GA and the PSO [13–15].

The FPA algorithm is a relatively new metaheuristic that imitates the process of flower pollination to solve optimization problems and is shown to perform better than GA and PSO [13–15]. The benefit of FPA is its ability to efficiently use both global and local search for better refinement of the solutions. The local/global search traits which FPA possesses has allowed it to be used successfully in numerous fields and provide optimal solutions with minimal drawbacks [13–15]. Although various variants and applications of FPA are mentioned, very little focus was given to the applications of FPA to solve the complex problems of UAV trajectory planning in 3D environment. For that, an efficient application of FPA to successfully determine optimal trajectory for fixed-wing UAVs is needed. In this paper, we propose a parallel FPA on multicore CPU using OpenMP and on GPU using CUDA to reduce the execution time and to allow for real-time UAV trajectory planning.

The remainder of the document is organized into sections. Section 2 presents the architectures of GPUs. Section 3 covers the details of the proposed method including the solution encoding, the fitness function, the FPA and the 2-opt operator. We present in Section 4 our

* Corresponding author: vincent.roberge@rmc.ca

approach to parallelizing the FPA and the 2-opt operator. Finally, we present and discuss in section 5 the results of our numerical simulation.

2 Graphics Processing Unit

GPUs were originally built using fixed pipelines optimized for a very high throughput of operations to support graphics applications. With a constant demand from the gaming industry for new graphics features a more flexible design was needed and pipelines were replaced by programmable processors. This change has allowed GPUs to be used for scientific computing applications. Initially limited to integer calculations, the processors found on today's GPUs support single and double floating-point operations and even special functions. Equipped with thousands, and in some instances, tens of thousands of cores, GPUs have an extremely high processing capability, much more than what is typically found in standard CPUs, given that the algorithms they run can be efficiently parallelized for their architecture. The architecture of an NVIDIA GPU is shown in Fig. 1. The GPU is composed of several building blocks containing Streaming Multiprocessors (SM). These SMs have the necessary components to perform highly threaded computations such as Streaming Processors (SP), registers, control logic, and Shared Memory. The SM is interconnected to a Global Memory which receives and transfers data to and from the host computer.

On NVIDIA GPUs, a programmer can use the Compute Unified Device Architecture (CUDA) to code sequential instructions to be run on the CPU, and *kernels* which are functions that run in parallel on the GPU. When a kernel is called from the CPU, threads are launched on the GPU to execute the kernel. Threads are organized in blocks and are mapped to SMs. All threads in the same thread block can communicate together through their Shared Memory. Threads from different thread blocks can communicate through Global memory, but the consistency of the Global Memory is only guaranteed at kernel completion.

In this work, the GPU is particularly well suited to implement a parallel FPA as the algorithm exhibits a high level of parallelism and the GPU contains a very large number of compute cores which allows the parallel FPA to run efficiently.

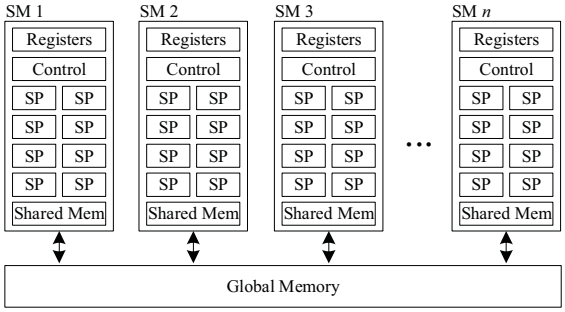

Fig. 1. GPU Architecture

3 Proposed Optimization Strategy

This section covers the optimization strategy proposed including the environment representation, the encoding of the candidate solutions, the fitness function, the FPA and the 2-opt operator.

3.1 Solution Encoding and Environment Representation

In the FPA, each candidate solution represents a path connecting the starting point to the end point. The path is defined as a series of waypoints in a 3D environment. The (x, y, z) coordinates of the waypoints are stored sequentially in a vector as in $[x_0, y_0, z_0, x_1, y_1, z_1 \ldots, x_n, y_n, z_n]$. The terrain is represented using a 2D matrix where each element of the matrix represents the elevation of the terrain at that location. Some of the terrains used in this work are fictional while others are real. Elevation maps for real terrains were taken from the Canadian Digital Elevation Model Repository [16]. The environment representation used here also allows the definition of no-fly or danger zones in the form of cylindrical areas of infinite height. These zones are inputted by identifying the (x, y) coordinates of the center of the cylinder and its radius. From these cylinders, a no-fly zone matrix is built. This matrix has the same dimensions as the elevation matrix and contains a 0 for all locations that are not in a no-fly zone and a 1 for location that falls in a no-fly zone. Using a matrix, speedup the remaining of the calculations.

By encoding the solutions as a series of waypoints, the trajectories are composed uniquely of line segments and therefore include discontinuities at every waypoint. To be flyable by fixed-wing UAVs, these trajectories, must first be smoothed. In this work, we use a technique proposed by Labonté in [17] to connect the line segments using circular arcs as shown in Fig. 2. This technique has the advantageous of generating simple trajectories. The smoothing operation is performed on every candidate solutions before its fitness is computed.

3.2 Fitness Function

Once a candidate solution has been smoothed, its fitness can be computed. This process is divided into two parts: the calculation of a penalty term and the calculation of a cost term.

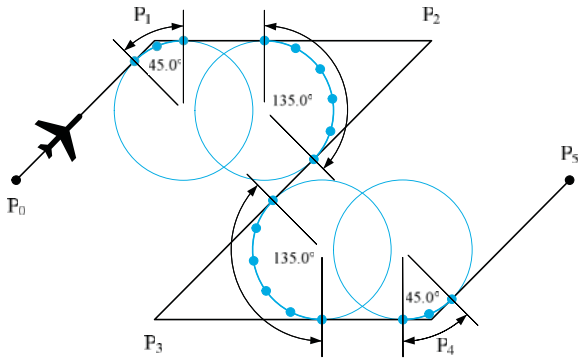

Fig. 2. Smoothing of a 6-waypoint trajectory using circular arcs

3.2.1 Penalty Term

The first part consists of computing the penalty term P which is used to penalize solutions that violate a feasibility constraint. This term is computed as follows:

$$P = d_{ut} + d_{nf} + d_{ea} + \left(\frac{N_{impossible\ arcs}}{N_{arc}} * l_{traj}\right) \quad (1)$$

where d_{ut} is the distance flown under the terrain, d_{nf} is the distance flown through a no-fly zone, d_{ea} is the distance flown at an excessive descent or climbing angle as defined by the UAV characteristics, $N_{impossible\ arcs}$ is the number of circular arcs that could not be constructed to connect two linear segments due to the incoming or outgoing segment being too short, N_{arc} is the total number of circular arcs that were attempted to be constructed and l_{traj} is the total length of the trajectory. The number of impossible arcs is normalized and multiplied by the length of the trajectory so that the last term of the equation has the same units as the other terms in the equations (i.e., meters). To calculate d_{ut} and d_{nf} we use both the elevation matrix and the no-fly zone matrix and we check at regular intervals along each segment forming the trajectory to see if the segment hits the terrain or breaches a no-fly zone.

3.2.2 Cost Term

The second part of the fitness function is calculating the cost term which assesses the quality of feasible solutions and represents the optimization objectives. In this work, the optimization objectives are to minimize the distance time altitude of the trajectory to avoid detection by enemy radars and to minimizing the fuel consumption to increase the range of the UAV. It is important to note that one cannot simply minimize the overall average altitude of a trajectory as the final path could include multiple loops at low altitude in an effort to reduce the overall altitude of the path which is obviously not wanted. These optimization objectives used here are problem specific and could be changed based on the missions. The equation for the cost term used in this work is as follows:

$$C = w_1 * \frac{a_{avg} * l_{traj}}{a_{utopia} * l_{utopia}} + w_2 * \frac{f_{req}}{f_{utopia}} \quad (2)$$

where a_{avg} is the average altitude of the trajectory, l_{traj} is the length of the trajectory, f_{req} is the quantity of fuel required to fly the trajectory calculated using [18], w_1 and w_2 are coefficients set to 0.5, but that can be adjusted to define the relative importance of the two optimization objectives. It is important to note that the altitude, length and fuel terms must be normalized before they can be added as they do not have the same units. The normalization is based on [19] and uses utopia points as references. We defined the utopia altitude a_{utopia} as the average altitude of the fictional line segment connecting the start point to the end point. Likewise, we defined the utopia length l_{utopia} and utopia fuel f_{utopia} as the length of that fictional segment and the fuel required to fly that fictional segment.

3.2.3 Combined Fitness Function

The fitness function $F(\vec{x})$ of candidate solution $\vec{x}$ is a combination of the penalty term and the cost term and is defined as follows:

$$F(\vec{x}) = \begin{cases} 0 + \frac{1}{1+P} & , \quad P > 0 \\ 1 + \frac{1}{1+C} & , \quad P = 0 \end{cases} \quad (3)$$

This function is carefully built so that infeasible solutions have a fitness ranging between 0 and 1 and feasible solutions have a fitness ranging between 1 and 2. During its operation, the FPA uses this fitness function to guide the search towards optimized solutions until an optimum is found.

3.3 Flower Pollination Algorithm

The FPA was first published by Xin-She Yang in 2012 [20] and has since been used in a wide range of fields. It is based on pollination process of flowering plants and uses global and local pollination to improve a population of candidate solutions. The global pollination step is represented by a pollinator (such as insects or wind) transporting flower pollen over a long distance, and this approach guarantees that the fittest pollen of high quality is transferred over to the next generation by following the biotic method's:

$$x_i^{t+1} = x_i^t + L(\lambda) * (x_i^t - g^*) \quad (4)$$

where x_i^t is the i[th] pollen or solution at iteration t, g^* is the current best solution and L(λ) is the lévy flight distribution. Lévy flight is used to imitate the movement of insects over short and long distances.

Local pollination (self-pollination) and flower constancy can be given as:

$$x_i^{t+1} = x_i^t + \epsilon * (x_j^t - x_k^t) \quad (5)$$

where x_j^t and x_k^t are pollens selected randomly from different flowers, while ϵ is a random number that follows the uniform distribution in [0, 1].

In general, the flower pollen population or solutions are randomly initialized by FPA. A new solution is generated for each generation using either global pollination or local pollination, as determined by the switch probability p [0, 1]. The FPA flowchart is given in Fig. 4.

3.4 2-Opt Operator

In the proposed FPA-based algorithms, solutions are randomly initialized and their waypoints are erratically placed on the map. This means that trajectories initially include several loops. It is hard for the FPA to remove these loops, but there exist local search algorithms that are well suited for this task. One of them is the 2-opt operator [21]. The 2-opt operator is a deterministic local search algorithm that is often used to solve the Travelling Salesman Person. Starting from the first node, it visits the next node and generates a new solution by reversing the order of the sub-trajectory and verifying if the new trajectory has a better fitness. If it does, the new trajectory replaces the solution and the 2-opt

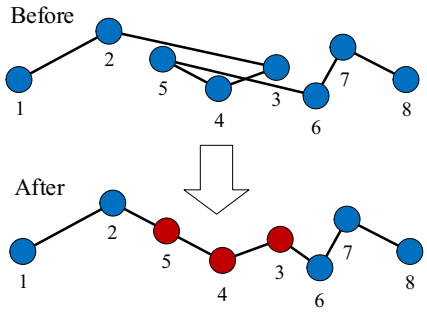

Fig. 3. 2-opt operator to remove loops

algorithm restart the search from the first node. If the new trajectory is not better, the search continues to the next node. The algorithm is implemented as two nested loops and the search restarts when a better solution is found. The details of the 2-opt algorithm is covered in [21].

An example of the 2-opt algorithm applied to the UAV trajectory planning problem is shown in Fig. 3 where it successfully removes a loop from the trajectory. Because the 2-opt operator requires the evaluation of the candidate solution after every sub-trajectory reversing, it is a very expensive function computationally. To reduce its effect on the overall runtime of the algorithm, we run the 2-opt operator every 20 iterations of the FPA as identified in Table 3. This has the benefit of using the 2-opt without affecting too much the overall runtime. Another important note regarding the 2-opt is that the number of iterations required to optimize each solution

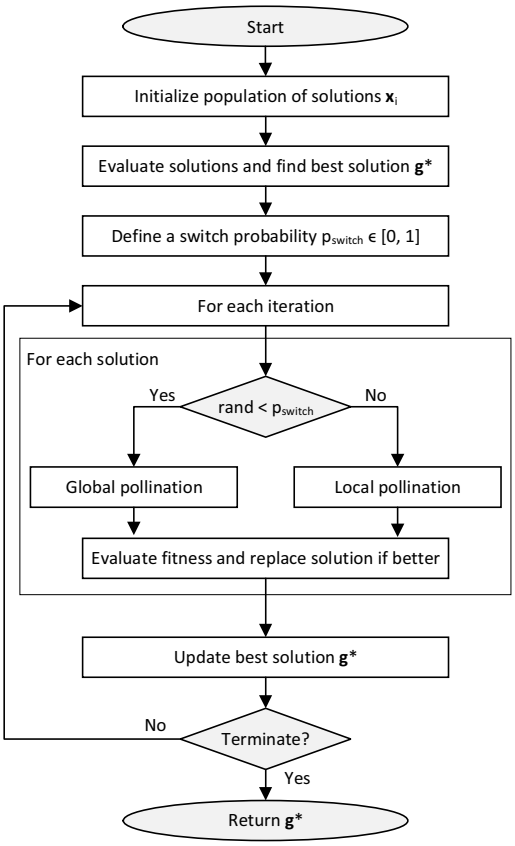

Fig. 4. Flowchart of the Flower Pollination Algorithm (FPA)

is different. This has implications for the parallelization strategy as it creates work imbalance between threads. Our parallelization strategy deals with this work imbalance.

4 Parallel Implementations

In this work, two parallel implementations were developed: one for multicore CPU using OpenMP and one for GPU using CUDA. As shown in Fig. 4, at each iteration of the FPA, a for-loop is used to improve each candidate solution. The 2-opt operator is also implemented as a for-loop operation where the 2-opt operator is applied independently on each solution.

In the parallel implementation on multicore CPU, the iterations of the for-loops are distributed among multiple OpenMP threads. Because the number of solutions (i.e., the number of iterations in the for-loop) is much larger than the number of threads, work imbalance between the iterations is easily hidden resulting in a very efficient parallelization strategy.

In the parallel implementation on GPU, a much higher level of parallelism must be exploited to benefit from the massively parallel architecture of the GPU. The check for the p_{switc} is done using one thread per solution, the global and local pollination is done using one thread per dimension of each solution and the replacement of the solutions is also done using one thread per dimension of each solution.

Calculating the fitness of the candidate solutions involves several operations and one thread block is used for each candidate solution. For the smoothing of the trajectory, an initial run is performed to compute the number of points needed for each circular arc, then, each thread processes one or more points of the arc. The result of the smoothing is a trajectory with a large number of linear segments. For the calculation of d_{ut} and d_{nf}, the number of points to be verified along each linear segment from the smoothed trajectory is first calculated, then each thread verifies one or more point. To calculate d_{ea}, a_{avg} and f_{req}, each thread verifies one or more linear segment from the smoothed trajectory. To compute $N_{impossible\ arcs}$, each thread computes one or more waypoints from the original trajectory. Finally, to compute the overall fitness, the thread block performs several parallel reductions to compute the sums of all the terms found on all segments from the smoothed trajectory. The calculation of the fitness therefore uses in a very large number of CUDA threads which exploits the massively parallel architecture of the GPU.

Parallelizing the 2-opt operator on the GPU is a challenge as each solution requires a different number of iterations which creates work imbalance between the solutions. To deal with this, we use a feature from NVIDIA GPUs called *Dynamic Parallelism* which allows kernels to launch child kernels. With Dynamic Parallelism, we launch a first kernel with one thread block per solution. Each thread block contains only a single thread which runs the 2-opt operator for the solution. To evaluate the fitness of the candidate trajectories, the single thread launches a child kernel composed of a single block and 128 threads which

performs the fitness calculation as explained in the previous paragraph. Using Dynamic Parallelism allows the GPU to dynamically schedule all the child kernels and handle the work imbalance between solutions. Solutions that complete the 2-opt operator early terminates and other solutions can continue longer.

5 Experimental Results

5.1 Quality of Computed Solutions

To validate the proposed trajectory planning algorithm based on the FPA and 2-opt operator, three tests were performed. The first test aims to assess the quality of the solutions computed by the proposed FPA-2-opt. Because each previous paper used their own maps and own UAV model, it is difficult to replicate and compare our work to previous works. For this reason, our assessment is qualitative and based on the feasibility of the final solutions and their quality as viewed on plotted 3D figures. For this test, we used the model for a small-scale UAV similar to the Silver Fox manufactured by BAE Systems. The specifications for the UAV are listed in Table 1. These specifications are useful to compute the fuel required to fly the trajectory. The flight properties of the UAV-autopilot system which models the dynamic behavior of the UAV are listed in Table 2. These are useful when assessing the feasibility of a trajectory and when smoothing the trajectory. To test the FPA-2-opt algorithm, four maps were used. Two are synthetic maps, the third one is from mountainous regions in North-East Afghanistan and the last one is from northern Russia, East of Norilsk. The FPA-2-opt algorithm was configured using the parameters listed in Table 3. The computed trajectories are shown in Fig. 5 to Fig. 8. All computed trajectories are feasible, do not collide with the terrain, avoid the no-fly zones, respect the maximum ascent and descent rates of the UAV. They look optimized in the sense that the minimize the average altitude and the overall fuel consumption as defined by the optimization objectives and they are smoothed using the minimum turning radius of the UAV demonstrating the good functioning of the proposed algorithm.

Table 1. Specifications of the Silver Fox-like UAV [18]

Parameter	Value
Weight (N)	113
Weight of fuel (N)	19
Wing span (m)	2.4
Wing area (m^2)	0.768
Global lift coefficient	1.26
Drag coefficient at zero lift	0.0251
Engine power (W)	1850
Specific fuel consumption (N/W*s)	*1.1569E-6
Propeller efficiency	0.8

* Modified from reference

Table 2. Flight properties of the UAV-autopilot system

Parameter	Value
Cruising speed (m/s)	23
Turning radius (m)	150
Max ascent angle (degrees)	10
Max descent angle (degrees)	30

Table 3. Flower Pollination Algorithm parameters

Parameter	Value
Number of iterations	500
Number of solutions	512
Switching probability	0.5
Initial epsilon	0.25
Final epsilon	0.02
2-opt frequency	Every 20th iteration

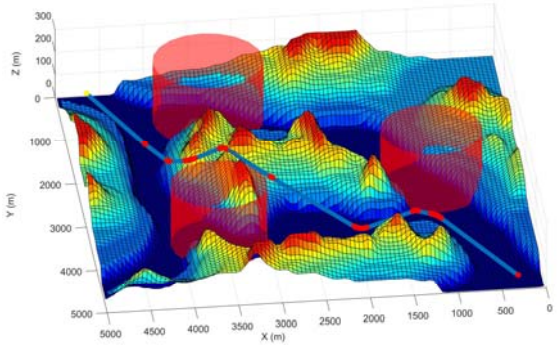

Fig. 5. UAV trajectory computed by the FPA for the canyon map (synthetic map, 25 km^2, terrain elevation from 0 to 260 m AMSL)

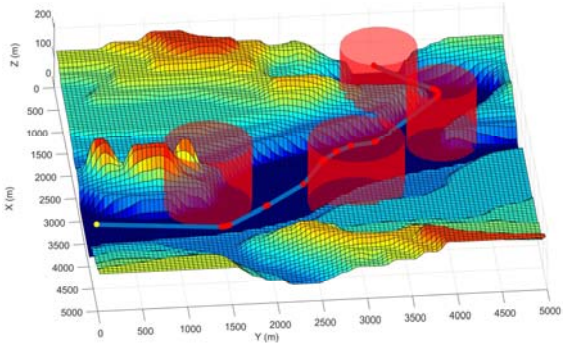

Fig. 6. UAV trajectory computed by the FPA for the river map (synthetic map, 25 km^2, terrain elevation from 0 to 217 m AMSL)

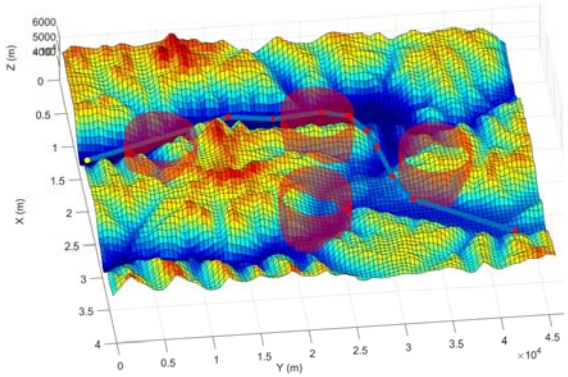

Fig. 7. UAV trajectory computed by the FPA in North-East Afghanistan (Top-left corner is 37.0705N-73.0772E, 1,672 km^2, terrain elevation from 3,139 to 6,082 m AMSL)

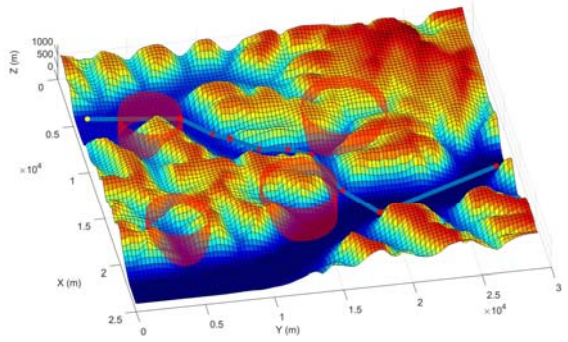

Fig. 8. UAV trajectory computed by the FPA in Russia, East of Norilsk (Top-left corner is 69.2464N 91.1934E, 716 km^2, terrain elevation from 40 to 1,382 m AMSL)

5.2 Benefit of the 2-Opt Operator

In the second test, we verify the advantage of the 2-opt operator. For this, we run the FPA algorithm 50 times without and with the 2-opt operator for each of the four maps and lists in Table 4 the statistics about the solutions found. To compare the fitness of the solutions computed without and with the 2-opt operator, we use a T-test and compute the p-value between the two distributions. A p-value below 0.05 means that the two means are statistically different. In the case of the data listed in Table 4, the mean fitness of the solutions found by the FPA with 2-opt is higher for all four maps and the p-value is zero for all four maps. This means that in all cases, using the 2-opt generates solutions that are statistically better proving the superiority of the hybrid FPA-2-opt compared to the FPA alone. As explained in section 3.4, the 2-opt operator is very efficient at removing loops from the candidate trajectories which are randomly initialized at the beginning of the search.

5.3 Performance of Parallel Implementation

The final test aims to evaluate the performance improvement brought by the two parallel implementations. The test is done on a Dell 5820 workstation equipped with an Intel Xeon W-2195 processors with 18 cores running at a base frequency of 2.30 GHz and a turbo frequency of 4.30 GHz. The computer is configured with an NVIDIA RTX A6000 GPU with 10,752 running at 1.41 GHz. Given the CPU frequency, the maximum theoretical speedup of a parallel application is:

$$\mathrm{N} * \frac{F_{turbo}}{F_{base}} = 18 * \frac{2.3}{4.3} = 9.63\mathrm{x} \quad (6)$$

In this test, we ran the FPA-2-opt algorithm 10 times using 1 to 36 threads and have measured the runtime which is plotted in Fig. 9. It can be noted that there is a performance slow done when the number of threads approaches the number of physical cores, but that a maximum speedup of 9.6x is achieved when the number of threads is equal to the number of logical cores of the CPU (here, the CPU supports hyperthreading). This maximum speedup is slightly above to the theoretical speedup which means that the proposed FPA-2-opt parallelizes very efficiently on multicore CPUs.

Next, the second parallel implementation, which is the CUDA version, is run on the RTX A6000 GPU. This version is much more complex, but benefits for a much more parallel processor. Now, the number of threads cannot be controlled, as the entire GPU is used, but we vary the number of solutions to demonstrate the point at which the GPU becomes saturated and maximum speedup is achieved. In the case of the RTX A6000 GPU, this happens with 512 solutions. Because our algorithm uses one thread block per solution, hundreds of threads are launched for each solution so saturation occurs when several thousand threads are launched. The proposed GPU implementation has a speedup of 68.5x compared to the sequential implementation on CPU and runs in just 1.14 seconds compared to 78.4 seconds for the CPU for 512 solutions. Because UAVs operate in a dynamic environment and often need to recompute their trajectory in flight, this speedup is truly an advantage.

Table 4. Mean results of 50 trials for the FPA with and without 2-opt

Scenario	FPA without 2-opt				FPA with 2-opt				T Test
	Length (m)	Average Alt (m)	Fuel (Newton)	Fitness (std dev)	Length (m)	Average Alt (m)	Fuel (Newton)	Fitness (std dev)	p-value
Canyon	8,418	119.3	0.127	1.295±0.021	7,320	86.6	0.117	1.348±0.019	0.00
River	6,559	72.2	0.099	1.363±0.025	6,126	50.0	0.090	1.420±0.018	0.00
Afghanistan	52,874	4236.2	1.003	1.451±0.024	52,737	4049.6	0.992	1.461±0.010	0.00
Russia	34,031	445.6	0.489	1.346±0.026	33,386	322.0	0.477	1.395±0.017	0.00

5.4 Real-Time Performance

The real-time performance of the proposed algorithm is assessed in terms of the distance travelled by the UAV during the time it takes to compute a new trajectory. This distance is computed based on the velocity of the UAV and the computation time measured in section 5.3. In this work, because the UAV flies at constant velocity, the distance can be computed using:

$$d = v * t = 23\ m/s\ * 1.14\ s\ = 26.22\ m \quad (7)$$

This means that the Silver Fox-like UAV used in this work travels only 26.22 meters during the times it takes for a new trajectory to be computed. This distance is longer than a typical trajectory which means that the trajectory planning system will always be able to compute the next trajectory while the UAV is flying the current trajectory. Also, assuming that the onboard sensors are able to detect obstacles at a distance greater than 26.22 meters, this gives enough lead time for the trajectory planning system to replan the trajectory in case a new obstacle is detected. Based on those assumptions, we conclude that the parallel implementation on GPU allows for real-time trajectory planning which is not the case for the sequential implementation on CPU.

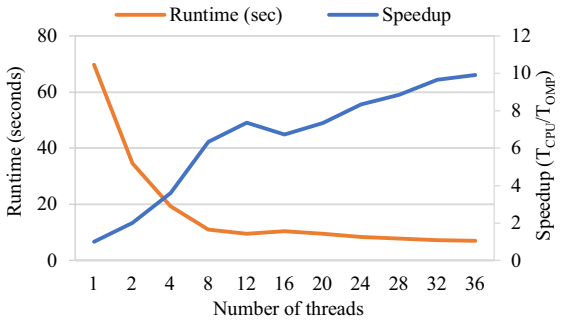

Fig. 9. Runtime and speedup for the parallel implementation on multicore CPU using OpenMP

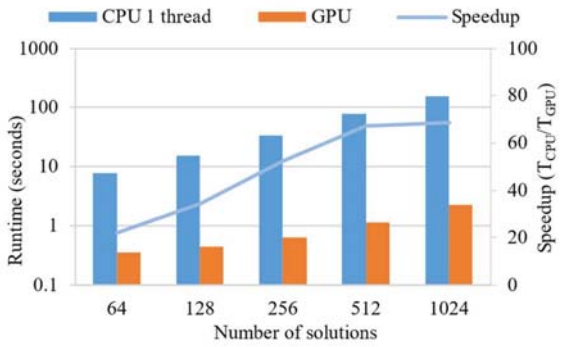

Fig. 10. Runtime and speedup for the parallel implementation on GPU using CUDA

6 Conclusion

This paper presented a parallel trajectory planning algorithm for fixed-wing UAVs based on the FPA and the 2-opt operator. The FPA is a global optimization algorithm while the 2-opt operator provides a local search strategy. The proposed approach was tested in a numerical simulation using four realistic and complex 3D scenarios and successfully computed quasi-optimal trajectories for the UAV while considering the flying characteristics of the UAV such as its minimum turning radius and maximum ascent and descent rates. To cope with the complexity of the problem and allow for real-time planning, the algorithm was parallelized on multicore CPU using OpenMP and on GPU using CUDA. A speedup of 9.6x on the CPU and 68.5x on the GPU were measured. The algorithms on GPU runs in just 1.14 seconds compared to 78.4 seconds for the sequential version on CPU. The proposed trajectory planning module contributes to the development of autonomous UAVs by allowing them to plan their route while in flight. In future works, we intent to implement and test other metaheuristics for UAV trajectory planning to compare their efficiency in this particular application.

Acknowledgement

We gratefully acknowledge the support of NVIDIA Corporation who donated the RTX A6000 GPU used for this research.

References

1. E. Masehian and D. Sedighizadeh, “Classic and Heuristic Approaches in Robot Motion Planning – A Chronological Review,” *World Acad. Sci. Eng. Technol.*, vol. 1, no. 5, 2007.
2. M. Colnarič, L. P. Behnck, D. Doering, C. E. Pereira, and A. Rettberg, “A Modified Simulated Annealing Algorithm for SUAVs Path Planning,” in *2nd IFAC Conference on Embedded Systems, Computer Intelligence and Telematics*, Maribor, Slovenia, Jun. 2015, vol. 48, pp. 63–68. doi: 10.1016/j.ifacol.2015.08.109.
3. X. Zhang and H. Duan, “An improved constrained differential evolution algorithm for unmanned aerial vehicle global route planning,” *Appl. Soft Comput.*, vol. 26, pp. 270–284, Jan. 2015, doi: 10.1016/j.asoc.2014.09.046.
4. Y. V. Pehlivanoglu, “A new vibrational genetic algorithm enhanced with a Voronoi diagram for path planning of autonomous UAV,” *Aerosp. Sci. Technol.*, vol. 16, no. 1, pp. 47–55, Jan. 2012, doi: 10.1016/j.ast.2011.02.006.
5. Y. Fu, M. Ding, and C. Zhou, “Phase Angle-Encoded and Quantum-Behaved Particle Swarm Optimization Applied to Three-Dimensional Route Planning for UAV,” *IEEE Trans. Syst. Man Cybern. - Part Syst. Hum.*, vol. 42, no. 2, pp. 511–526, Mar. 2012, doi: 10.1109/TSMCA.2011.2159586.
6. U. Cekmez, M. Ozsiginan, and O. K. Sahingoz, “A UAV path planning with parallel ACO algorithm on CUDA platform,” in *2014 International Conference on Unmanned Aircraft Systems (ICUAS)*, May 2014, pp. 347–354. doi: 10.1109/ICUAS.2014.6842273.

7. Y. Chen, J. Yu, Y. Mei, Y. Wang, and X. Su, "Modified central force optimization (MCFO) algorithm for 3D UAV path planning," *Neurocomputing*, vol. 171, pp. 878–888, Jan. 2016, doi: 10.1016/j.neucom.2015.07.044.
8. W. Zhu and H. Duan, "Chaotic predator–prey biogeography-based optimization approach for UCAV path planning," *Aerosp. Sci. Technol.*, vol. 32, no. 1, pp. 153–161, Jan. 2014, doi: 10.1016/j.ast.2013.11.003.
9. C. Xu, H. Duan, and F. Liu, "Chaotic artificial bee colony approach to Uninhabited Combat Air Vehicle (UCAV) path planning," *Aerosp. Sci. Technol.*, vol. 14, no. 8, pp. 535–541, Dec. 2010, doi: 10.1016/j.ast.2010.04.008.
10. S. Zhang, Y. Zhou, Z. Li, and W. Pan, "Grey wolf optimizer for unmanned combat aerial vehicle path planning," *Adv. Eng. Softw.*, vol. 99, pp. 121–136, Sep. 2016, doi: 10.1016/j.advengsoft.2016.05.015.
11. V. Roberge, M. Tarbouchi, and G. Labonte, "Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning," *Ind. Inform. IEEE Trans. On*, vol. 9, no. 1, pp. 132–141, Feb. 2013, doi: 10.1109/TII.2012.2198665.
12. V. Roberge, M. Tarbouchi, and F. Allaire, "Parallel Hybrid Metaheuristic on Shared Memory System for Real-Time UAV Path Planning," *Int. J. Comput. Intell. Appl.*, vol. 13, no. 2, pp. 1450008-1-1450008–16, Jun. 2014.
13. Z. A. A. Alyasseri, A. T. Khader, M. A. Al-Betar, M. A. Awadallah, and X.-S. Yang, "Variants of the Flower Pollination Algorithm: A Review," in *Nature-Inspired Algorithms and Applied Optimization*, X.-S. Yang, Ed. Cham: Springer International Publishing, 2018, pp. 91–118. doi: 10.1007/978-3-319-67669-2_5.
14. S. Pant, A. Kumar, and M. Ram, "Flower pollination algorithm development: a state of art review," *Int. J. Syst. Assur. Eng. Manag.*, vol. 8, no. 2, pp. 1858–1866, Nov. 2017, doi: 10.1007/s13198-017-0623-7.
15. H. Chiroma, N. L. M. Shuib, S. A. Muaz, A. I. Abubakar, L. B. Ila, and J. Z. Maitama, "A Review of the Applications of Bio-inspired Flower Pollination Algorithm," *Procedia Comput. Sci.*, vol. 62, pp. 435–441, Jan. 2015, doi: 10.1016/j.procs.2015.08.438.
16. Government of Canada, "Canadian Digital Elevation Model." https://open.canada.ca/data/en/dataset/7f245e4d-76c2-4caa-951a-45d1d2051333 (accessed May 18, 2022).
17. G. Labonté, "Sur la construction de trajectoires dynamiquement réalisables pour les avions à partir de suites de segments de droites," Collège militaire royal du Canada, Kingston, Ontario, Canada, Nov. 2009.
18. G. Labonté, "Simple formulas for the fuel of climbing propeller driven airplanes," *Adv. Aircr. Spacecr. Sci.*, vol. 2, no. 4, pp. 367–389, 2015, doi: http://dx.doi.org/10.12989/aas.2015.2.4.367.
19. Y. Ding *et al.*, "Discussions on Normalization and Other Topics in Multi-Objective Optimization," Toronto, Aug. 2006. [Online]. Available: http://www.cas.mcmaster.ca/~romanko/fmipw-2006-moo.pdf
20. J. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
21. H. Li, S.-Y. Liu, Y.-W. Huang, Y.-Q. Chen, and Z.-H. Fu, "An Efficient 2-opt Operator for the Robotic Task Sequencing Problem," in *2019 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, Aug. 2019, pp. 124–129. doi: 10.1109/RCAR47638.2019.9044008.