M. Massari, A. Wittig
Optimization of Multiple-Rendezvous Low-Thrust Missions on General-Purpose Graphics Processing Units
Journal of Aerospace Information Systems, Vol. 13, N. 2, 2016, p. 1-13
doi:10.2514/1.I010390

Optimization of Multiple-Rendezvous Low-Thrust Missions on General Purpose Graphics Processing Units

Mauro Massari [a] and Alexander Wittig [b]
Politecnico di Milano, Milan, Italy

A massively parallel method for the identification of optimal sequences of targets in multiple-rendezvous low-thrust missions is presented. Given a list of possible targets, a global search of sequences compatible with the mission requirements is performed. To estimate feasibility of each transfer, a heuristic model based on Lambert's transfers is evaluated in parallel for each target making use of commonly available General Purpose Graphics Processing Units (GPGPUs) such as the NVIDIA Tesla cards. The resulting sequences are ranked by user-specified criteria such as length or fuel consumption. The resulting preliminary sequences are then optimized to a full low-thrust trajectory using classical methods for each leg. The performance of the method is discussed as a function of various parameters of the algorithm. The efficiency of the GPGPU implementation is demonstrated by comparing it with a traditional Central Processing Unit (CPU) based branch and bound method. Finally, the algorithm is used to compute asteroid sequences used in a solution submitted to the seventh edition of the Global Trajectory Optimization Competition.

[a] Assistant Professor, Department of Aerospace Science and Technology, Via La Masa 34, 20156, Milan , Italy
[b] AstroNet II Experienced Researcher and Postdoctoral Fellow, Department of Aerospace Science and Technology, Via La Masa 34, 20156, Milan, Italy

I. Introduction

The goal of Multiple-Rendezvous Low-Thrust (MRLT) missions is to use a spacecraft equipped with low-thrust propulsion to visit as many targets as possible, out of a given list of possible targets, within a fixed duration for the mission. An example for such a mission are visits to as many asteroids in the main asteroid belts as possible with a given amount of fuel and time. The choice of the sequence of target objects in MRLT missions is crucial in order to achieve the primary and secondary objectives of such a mission. Besides the number of targets visited, a typical objective to be minimized is the fuel consumption.

The problem of selecting the ideal sequence and times of asteroid visits is extremely complex, and involves both discrete variables (the IDs of the targets) as well as continuous variables (starting epoch, transfer time). The search space is already large even for relatively small sets of possible targets; but once realistic sets of targets are considered, the number of possible sequences quickly becomes enormous. The underlying optimization problem is hence a very difficult one and consequently very popular as a test bed for various types of optimizers. In particular, MRLT missions are popular problems in the Global Trajectory Optimization Competition (GTOC) series[1].

In the past, this kind of problems have been approached using different variants of stochastic optimization algorithms, such as evolutionary branching algorithms [2], particle swarm optimization [3], or ant colony optimization [4]. Systematic search approaches to this type of problem typically employ branch-and-bound type algorithms [5]. In the past, those have relied heavily on extensive pruning of the targets [6]. This reduces the target sets to a relatively small number of targets by filtering based on various criteria such as orbital parameters or phasing, making a systematic search feasible.

In this work, we develop a systematic search algorithm capable of treating large sets of over 16.000 possible targets without previous pruning. Apart from a very fast execution time, the new algorithm is capable of returning not just one but many (often in the hundreds or thousands) high-scoring sequences of targets without significant performance impact. In practice this allows the integration of this search algorithm in larger optimization loops, selecting among the high-ranking sequences based on other criteria (e.g. common final targets). The key component in this algorithm

is the efficient implementation of the massively parallel global tree-search algorithm on modern General-Purpose Graphics Processing Units (GPGPUs).

Traditional algorithms have been developed considering the typical computing architecture of multicore Central Processing Unit (CPU) machines. Those systems are equipped with a small number of very powerful cores with several layers of memory caches. With the advent of modern GPGPUs, such as the NVIDIA Tesla GPGPU accelerator cards [7], a huge number of Arithmetic Logic Units (ALU) per card (typically in the thousands) are made available at costs comparable to those of a high-end multicore CPU [8]. However, the large increase in raw arithmetic computing power comes at the cost of limited performance of each core compared to traditional CPU cores mostly due to the lack of the sophisticated hardware caches present in CPUs.

The GPGPU architecture is not suitable to run different, complex subprograms on each core. Instead, the hardware is optimized for what NVIDIA refers to as the Single Instruction Multiple Threads (SIMT) parallel processing paradigm [9], which applies the same operation to different input data in parallel. This requires a different approach in algorithm design to maximize the impact of those new hardware capabilities in various fields of scientific computing [10].

Apart from typical GPGPU applications such as linear algebra [11] and computational fluid dynamics [12], as well as and fast n-body dynamics simulations [13], GPGPU programming recently has also found its way into more specifically aerospace related research topics. GPGPUs have been used for the parallel computation of trajectories to enable fast Monte-Carlo simulations [14], uncertainty propagation [15], as well as non-linear filtering [16].

We chose the standardized OpenCL package [17] for the GPGPU programming. The OpenCL heterogeneous computing platform provides a platform agnostic C like programming language along with compilers for most current accelerator cards such as those by NVIDIA, Intel and ATI. Furthermore, it provides libraries for both the host side (CPU) as well as the client side (GPGPU) to facilitate common tasks in heterogeneous programming in a hardware independent manner.

The remainder of this paper is structured as follows. In Section II the basic concept of sequence generation for MRLT mission and some of their possible applications are exposed. In Section III, the basic concepts of GPGPU programming will be briefly recalled and compared with the traditional

CPU programming approach. These considerations lead naturally to the proposed new sequencing algorithm which is described in Section IV. Several subsections will discuss in detail the different aspects of the algorithm such as the host based driver program as well as the GPGPU based massively parallel computation kernels. The final optimization of the resulting preliminary sequences to obtain complete low-thrust trajectories with a full thrust history is described in Section V. The performance of the algorithm and the effect of various parameters is analyzed on a realistic set of about 16.000 possible target asteroids in Section VI. The performance of the GPGPU implementation is also compared to a traditional CPU implementation of a sequence generation algorithm based on branch and bound technique [5]. Finally, the implementation of the proposed algorithm is applied to the computation of asteroid sequences for the optimization problem posed in GTOC7 in Section VII.

II. Sequence Generation in MRLT Missions

The problem of sequence generation in multi-rendezvous low-thrust missions has been featured prominently in several previous editions of the GTOC series. However, with the recent interest in Asteroid and Comet exploration, the problem of maximizing subsequent rendezvous with multiple small bodies is becoming interesting also for real mission design. This problem is different from the typical interplanetary exploration problems faced in the past as the rendezvous with small bodies, characterized by a very low mass when compared with planets and moons, can be performed efficiently with low-thrust propulsion in a reasonable amount of time. In this paper, the focus will be put mainly on MRLT mission in the heliocentric frame and considering Asteroids as possible targets, but the proposed approach can be applied directly to other multiple-rendezvous problems such as the ones faced in a debris removal mission in Earth orbit.

Following the above considerations, it is possible to make some assumptions on the problem dynamics that allows the efficient implementation of a massively parallel searching algorithm suitable for GPGPU architectures. The searching algorithm is not required to immediately provide an optimized low-thrust transfer trajectory; it should only return a sequence of targets with a high likelihood of yielding a feasible low-thrust trajectory after subsequent optimization. This fact allows the simplification of the search algorithm as only the feasibility of a transfer between two successive

rendezvous should be addressed, not the actual design of the optimal low-thrust transfer itself.

Following this idea, it has been decided that dynamics used to model the motion should be as simple as possible, because it should just represent with sufficient accuracy the feasibility of the transfer, not the actual transfer trajectory. For this purpose simple two-body dynamics have been chosen to represent the motion between rendezvous, and the rendezvous condition itself is imposed by just equating position and velocity of the spacecraft to those of the target, completely neglecting the gravity of the target and their relative dynamics. This assumption is made because the targets have very small mass and a very irregular gravity field. The actual rendezvous problem in such conditions must be designed ad-hoc for each target, and will mainly affect the actual transfer orbit but not the feasibility of the complete transfer sequence. The same is valid for the gravity attraction of other bodies of the Solar System, they perturb the motion of the spacecraft and thus affect the actual transfer orbit, but in most cases will only slightly affect the feasibility of the transfer.

A second consideration is related to the actual low-thrust transfer. Following the idea that during sequence generation only the feasibility of the transfer should be addressed, and not the actual design of the low-thrust transfer orbit, it has been decided to estimate the feasibility of the transfer using the classical impulsive dynamics for the computation of the cost of the transfer. The resulting cost estimate is then successively modified using the low-thrust propulsion features to obtain a modified cost and consequently the evaluation of the overall feasibility of the transfer. This should be possible because the low gravity of the small bodies allows the escape and capture using low-thrust in limited time, and because in typical settings the transfer arcs between targets are short and thus the difference between impulsive and low-thrust trajectories is not that pronounced. Thus, the approximation of the transfer using a weighted solution of the Lambert's problem is acceptable.

Lambert's Problem [18] consist of finding the orbit, i.e. the solution of the Two-Body problem, connecting two points in space r_1 and r_2 within a specified time Δt. The solution of Lambert's Problem forms the basic building block of the rendezvous problem as it allows to find the orbital transfer from one point and time to another target point at a later time. The solution of this problem consists of identifying the orbital elements of the orbit connecting the two points in space and by itself is not related to the required maneuvers to perform the actual transfer. However, the

problem is usually extended by considering not only the initial and final position vectors r_1 and r_2, but also the initial velocity and the desired velocity at the target point v_1 and v_2. The size of the two impulsive maneuvers needed to perform the complete orbital transfer of the Lambert's Problem is then easily computed from the velocity difference at the initial and target point:

$$\Delta V_i = v_1 - v_1^L \tag{1}$$

$$\Delta V_f = v_2^L - v_2 \tag{2}$$

These assumptions are validated by numerical experiments showing the typical optimal thrust profile of such kinds of transfers to be a bang-bang profile thrusting at the beginning and the end of the arc while coasting between. This behavior mimics the Lambert's problem solution, with an initial ΔV_i for escape and a final ΔV_f for the capture. Moreover, the condition of maximizing the number of targets in the searching algorithm, typically implemented by limiting the time of each transfer, automatically discards all the long transfers requiring multiple revolutions around the Sun in favor of short transfer arcs between targets. The total cost of a single transfer between two successive rendezvous targets is simply computed using the total $\Delta V_l = \Delta V_i + \Delta V_f$ of the solution of the Lambert's problem.

To pass from the total cost of the transfer to its feasibility F, also the limitation on the acceleration due to the maximum thrust available must be considered. To this end, the mass of the spacecraft m_{sc} is assumed constant and equal to the value at departure during each transfer. Thrust is assumed to be continuous at a fraction α_t of the maximum thrust T_{max} for the entire duration of the transfer $(t_f - t_0)$. This yields a constant force $F = \alpha_t T_{max}$ acting on the spacecraft. From Newton's second law we obtain $F/m = a = \frac{dv}{dt}$. Integrating over the time interval $[t_0, t_f]$ we obtain the maximum achievable ΔV_{max} with low-thrust as

$$\Delta V_{max} = \frac{\alpha_t T_{max}(t_f - t_0)}{m_{sc}}. \tag{3}$$

Depending on the restrictions on available fuel, this value ΔV_{max} is adjusted downward if needed. If the solution of Lambert's problem yields a ΔV_l smaller than the maximum achievable ΔV_{max},

the transfer is considered feasible:

$$F = \begin{cases} 1 & \text{if } \Delta V_l \leq \Delta V_{max} \\ 0 & \text{if } \Delta V_l > \Delta V_{max}. \end{cases} \tag{4}$$

The heuristic factor $\alpha_t \in (0, 1]$ represents the fact that the low-thrust bang-bang transfer efficiency is significantly less than that of impulsive maneuvers assumed in the Lambert's problem solution.

With these definitions of cost and feasibility of the transfer between two successive rendezvous, the sequence of target can be generated evaluating each single transfer, and combining the results to obtain a feasible sequence of target that can be reached given constraints on available propellant m_p, mission time T and maximum thrust T_{max}. The evaluation of each single leg is done in the same way, the only difference is the mass of the spacecraft at departure m_{sc}, used to estimate the ΔV_{max}, which is affected by the previous transfers' mass consumption.

The propellant mass consumption is estimated considering the same assumption of impulsive maneuvers done for the Lambert's Problem. Therefore, the total cost of the previous transfers is computed summing the ΔV_l^i opportunely weighted:

$$\Delta V_{tot} = \alpha_f \sum_i \Delta V_l^i \tag{5}$$

where ΔV_l^i is the cost of the i-th Lambert's transfer and $\alpha_f \in [1, \infty)$ represents another heuristic scaling factor to mitigate the fact that the fuel consumption of low-thrust propulsion is typically higher than that of the corresponding Lambert's transfer. The Tsiolkovsky Rocket Equation [19] is used to compute the propellant mass used to provide the ΔV_{tot}:

$$m_p = m_0 \cdot \left(1 - \exp\left(\frac{\Delta V_{tot}}{I_{sp} g_0}\right)\right) \tag{6}$$

where I_{sp} is the Specific Impulse of the Low-Thrust Propulsion engine, g_0 is the standard gravitational constant on Earth and m_0 is the initial mass of the spacecraft.

In figure 1 the comparison between the optimized low-thrust transfer and the approximate

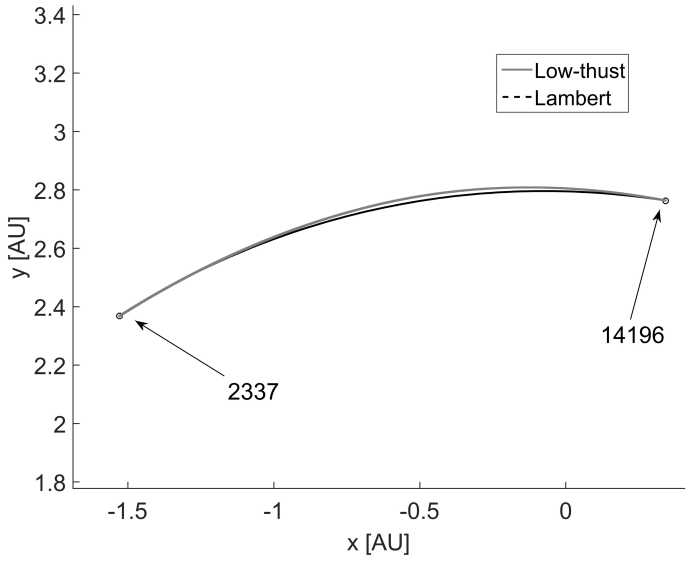

Fig. 1 Comparison of Lambert's arc and optimized low-thust transfer

transfer with the solution of the Lambert's problem is shown for one of the transfer arcs used in our GTOC7 solution.

As can be seen, the trajectories are almost superimposed, demonstrating that at least from the trajectory point of view the approximation is reasonable. Considering the propellant consumption, using $alpha_f = 1.5$ and $m_0 = 2000$ kg the computed propellant mass is respectively 145 kg for the Lambert's transfer and 150 kg for the optimized low-thrust transfer, demonstrating that also for propellant estimation the approximation is reasonable.

The accuracy of these estimations of the feasibility, and hence the quality of the resulting sequences, is highly affected by the choice of the parameters α_f and α_t. Increasing the value for α_t closer to 1 tends to make the estimation of feasibility less robust, but at the same time reduces the overestimation of the required flight time for a particular transfer. At the same time, reducing α_f reduces the overestimation of the fuel usage, but can lead to unfeasible sequences if fuel usage is underestimated. Thus, choosing both values closer to 1 allows the identification of longer sequences, at the risk of losing their real low-thrust feasibility. This in turn makes it more difficult to optimize actual low-thrust transfers matching the identified sequence. On the other hand, moving the values of those parameters away from 1 will increase the robustness of the feasibility estimation, yielding easily optimizable low-thrust trajectories at the cost of producing shorter sequences. We will analyze

the effect of the two parameters in more detail in Section VI.

III. GPGPU Programming

As stated in Section I this work is about exploiting the massively parallel architecture of GPGPUs for the problem of MRLTs. In recent years, GPGPUs have become very powerful following the increasing demand of both the gaming market and professional graphical applications. Usually not all available computational power is used when dealing with far from peak performance scene rendering. For this reason GPGPU producers started to investigate the possibility of using the unused computational power for general purpose computations. Today, high-end graphics cards are all capable of providing vast computational power for general purpose computation in all areas of scientific computing. Applications range from fluid dynamics to data mining [10].

However, GPGPU computational capabilities are conceptually very different from the ones of typical CPUs. This is due to the different architecture of GPGPUs, which originally were designed to perform the same operations in parallel on all the pixels of an image (*Single Instruction Multiple Threads*, SIMT) [9, 10, 20].

Thus GPGPUs are typically characterized by a very high number of Arithmetic Logic Units (ALUs), usually more than one thousand per GPGPU. They are equipped with dedicated fast memory without large intermediate caches in order to devote as many transistors as possible to ALUs. Figure 2 shows the typical architecture of CPUs and GPGPUs. Obviously each individual ALU is much more limited with respect to ALUs found on a general purpose CPU which is much more complex. However, by removing some features commonly found on CPUs such as elaborate branch prediction control and various levels of memory caching, GPGPUs can instead use the transistors freed up by removing these features to add more ALUs.

Consequently, single computations that run on a GPGPU typically run at a slower speed than the same computation on a modern CPU. But if the algorithm design allows it, those computations can run massively in parallel at the same time on a GPGPU, thus making up the shortfall in terms of a single computation by parallel computation.

GPGPU computation is affected also by a series of other limitations, mainly due to the fact that GPGPUs are separated from the main CPU and memory controller, thus reducing the bandwidth

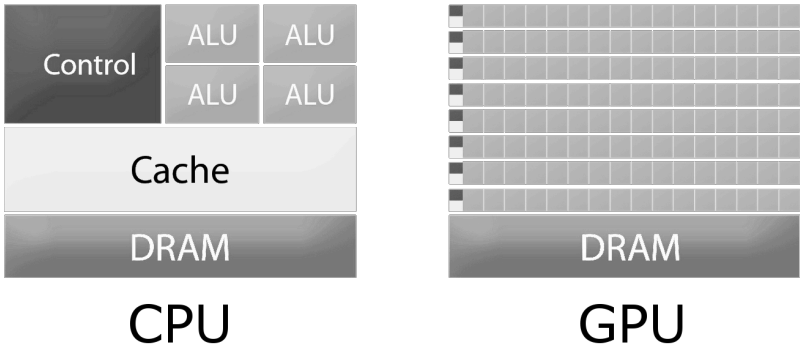

Fig. 2 CPU and GPGPU Architecture (after [9])

of memory transfers between the CPU main memory and the GPGPU memory to that of the PCI Express Bus. The theoretical peak data transfer rate of 16 GB/s for the x16 PCI express generation 2 connector of the NVIDIA Tesla K20 card, for example, is an order of magnitude slower than the 208 GB/s of the DDR5 memory on the card [21].

Moreover, until recently most GPGPUs were very fast when performing integer or single precision floating point computation, but are traditionally significantly slower for double precision computations [8]. Another limitation stemming from the SIMT structure of GPGPUs is the lack of true branching capabilities. Instead of performing real branching, until recently GPGPUs followed both possible branches and discarded the unneeded branch. While last-generation GPGPUs now support branching natively, the lack of pipelining and branch prediction control circuitry makes branching a very expensive operation [10].

These are just some of the reasons why traditional parallelization techniques commonly used, such as multithreading on multicore CPU systems with shared memory [22] or distributed computing with message passing on computing clusters [23] cannot be applied directly to GPGPU programming. Instead, code and algorithms developed for GPGPUs must be designed in a different way than those for CPUs.

Best performances on GPGPUs can be obtained when running the same piece of code (called a kernel) in parallel on different data sets while avoiding branching. In this way each thread performs the same operations on a different portion of the dataset which is common to all the threads [9]. This is often referred to as data parallelization. To illustrate the concept, consider the following analogy. A GPGPU is much like a subway: it moves many people at once, but all passengers will

go to the same place. On the other hand, a CPU is like a car: it moves only few passengers but it can go anywhere they want faster than the subway. However, too many cars trying to move a few passengers each result in a traffic jam that will slow down all cars, making the metro more efficient overall at moving large numbers of people.

Note that not all available GPGPUs can be used for general purpose computations. Only GPGPUs dedicated to high-end gaming or built explicitly for computing have reasonable performance. There are mainly two alternative available on the market, NVIDIA with CUDA cores and AMD (ATI) with the GCN architecture. NVIDIA GPGPUs are more commonly used for GPGPU computing than AMD ones, a fact influenced in large part by the existence of specialized CUDA libraries for scientific computing. Unfortunately the CUDA libraries are proprietary and can only be used to develop code for NVIDIA GPGPUs. However, both NVIDIA and AMD GPGPUs support the OpenCL toolkit [17] for parallel computing.

OpenCL is designed to take advantage of all the possible devices that allow parallel computation, similar to the OpenGL specification for 3D graphics rendering. It consists of a C-like language to write GPGPU code which is compiled at runtime for the selected platform. Thus OpenCL code can be executed on all major GPGPU cards such as those by NVIDIA, Intel and ATI. Furthermore, it provides libraries for both the host side (CPU) as well as the client side (GPGPU) to facilitate common tasks in heterogeneous programming in a hardware independent manner. For detailed description of OpenCL features, we refer the reader to the OpenCL 2.0 Specification [24].

This choice of framework for the implementation of the GPGPU part of the Sequence Searching algorithm allows for easy testing and comparison of the proposed algorithm on different GPGPU platforms. An implementation using the dedicated NVIDIA CUDA framework[9] to only target NVIDIA cards may under some circumstances yield a performance gain due to platform-specific optimizations. A detailed analysis of different GPGPU computing platforms and hardware, however, is beyond the scope of this work. We refer to the literature for comprehensive analyses of this topic [25, 26].

To summarize this brief introduction to GPGPU programming, the following considerations are crucial for the development of our GPGPU based Sequence Search code.

- The amount of data transferred from CPU to GPGPU (and vice-versa) should be reduced to avoid the bottleneck of the PCI Express Bus.

- The use of loops with variable numbers of iterations should be avoided in favor of fixed length loops so as to avoid branching. This facilitates the SIMT structure of the code.

- Dynamic memory allocation in GPGPU kernels is very inefficient due to the required locking. If necessary all memory should be preallocated on the GPGPU using the appropriate OpenCL calls before executing the kernels.

IV. Sequence Search Algorithm

The goal of the Sequence Search Algorithm is to find a preliminary feasible sequence of targets to be visited, together with their arrival and departure epochs. It is not required to obtain an optimized transfer for each leg, just a sufficiently accurate definition of the target IDs and of the visiting dates serving as an initial guess for a low-thrust optimizer. The actual optimization of the legs is performed later using a classical low-thrust optimization based on an indirect method described in more detail in Section V.

Starting from a given list of possible targets and their ephemerides, an initial time, and an initial object to start from, the algorithm returns a list of feasible optimal sequences ranked by user-specified criteria such as number of targets visited, fuel consumption, as well as total mission duration. The quality of the resulting sequences must be sufficiently high so that the actual low-thrust transfer can be quickly computed by e.g. an indirect method.

As described in Section II, the core of the sequence construction algorithm is a massively parallel implementation of a modified Lambert's problem solver, optimized for parallel execution on GPGPUs. The relatively simple Lambert's problem, unlike more complex and accurate methods, can achieve better performance in massively parallel GPGPU processing, as it allows a good load balancing which maximize the usage of all the available cores as described in Section III. From the cost of each single transfer, the heuristic averaging of the required ΔV_l over the time of flight is performed to estimate the low-thrust feasibility based on the thruster constraints as well as the total cost of each transfer as detailed in Section II. Exploiting the ability to run thousands of threads

in parallel, the feasibility of a transfer from a given target to all other possible targets is computed in parallel. Due to the independence of each computation, this process is highly efficient and can easily be parallelized on SIMT devices.

As this method has been designed to tackle very high dimension problems, it is not based on stochastic optimization. Stochastic optimization is very powerful when searching for a global optimum, but is highly sensitive to the number of parameters and does not handle problems which contain both discrete and continuous variables very well. The problem faced in this work is even more complex, as the discrete variables represent the IDs of the targets and can assume values within a very large set. In order to be effective in these conditions, a stochastic algorithm requires a very large population size, thus affecting considerably the performances of the search. For this reason, it has been decided to base the Sequence Search Algorithm on a constructive approach. Starting from a given ID and time, the sequence is generated exploring the search space extensively with a branch and bound approach. Obviously this approach is feasible only because the massively parallel architecture of the GPGPU allows to run a huge number of evaluation in parallel, thus reducing the computational time needed for the extensive search.

In order to construct the sequences, a stack-based construction method is employed. Starting from the initial target and time, the feasible targets within reach are identified. Out of those, a subset of promising targets is selected based on some user-defined selection criteria. Then in an iterative process the search is continued along all promising targets. In this way, the sequences are constructed with an extensive search of the solution space, which is not easily realizable on Multi-CPU machines. In Figure 3 this iterative process is illustrated. As shown, the sequences are constructed starting from the initial target, and considering only the promising (grey) targets out of all feasible targets (white) for the next steps, thus highly reducing the number of combinations that needs to be examined compared to a full combinatorial search.

It is important to note that this is not a brute-force approach, as only a subset of the feasible transfers are considered. This reduces very quickly the number of possible solutions to analyze. The algorithm can be further tuned using a parameter N_{min} which allows to select how many promising sequences to consider for further analysis in each step. A detailed description of this parameter is

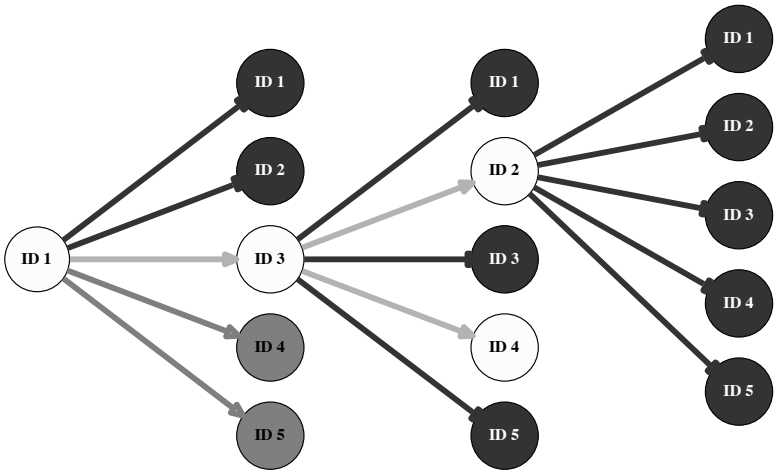

Fig. 3 Sequence Search iterative approach. Black are unfeasible targets, grey represents feasible targets, and white represents promising targets selected for further sequencing.

presented in Subsection IV A.

The overall architecture of the algorithm is presented in Figure 4 where one can see that it is mainly composed of two distinct parts, one on the CPU which coordinate all the computation and search, and one on the GPGPU which is mainly responsible for the evaluations. In the GPGPU part the modules implemented are dedicated to two distinct tasks, the actual evaluation of the feasibility and cost of the legs and the ephemerides retrieval and evaluation.

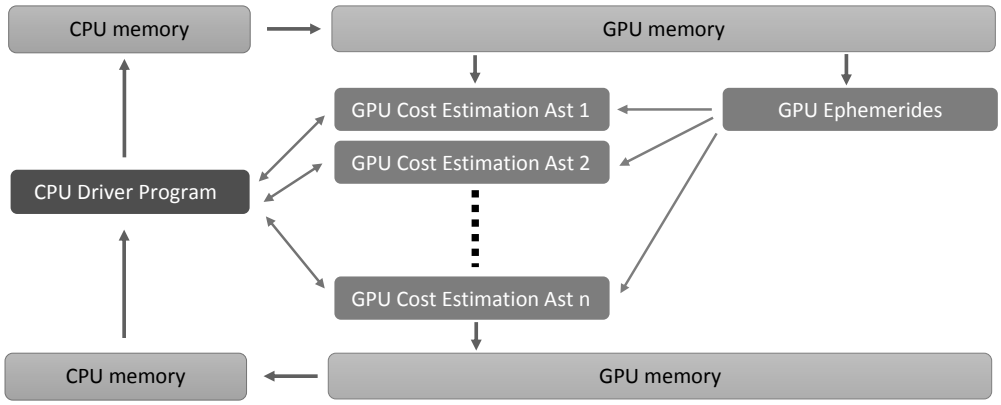

Fig. 4 Sequence Search Algorithm architecture

As the time consuming part of the Sequence Search Algorithm is concentrated in the evaluation of the single leg costs, it is possible to see that this algorithm is perfectly suited for exploiting the

massively parallel architecture of GPGPUs. It is worth noting that, in the presented architecture, the main memory transfer of the dataset and ephemerides information is performed only once at the beginning, thus avoiding any possible memory bandwidth bottleneck. In the following a description of the three main modules of the algorithm is presented.

A. CPU Driver Program

The CPU Driver module is the part of the code which supervises the operations and implements the actual searching framework. The operations performed in this modules are the following:

1. Initialize OpenCL library
2. Setup memory on the CPU and GPGPU(s)
3. Start sequence search main loop from starting ID
   (a) Start internal loop on time of flight (tof)
      i. Evaluate Lambert's transfer to all targets in parallel on GPGPU
      ii. Append feasible target(s) to current sequence and place on stack
      iii. Increase tof until at least N_{min} feasible targets found or mission time exceeded
   (b) If no feasible targets found, store the resulting final sequence
   (c) Continue main loop with next sequence on stack

After an initial part needed to initialize the GPGPU devices and to once transfer the ephemeris data from CPU to GPGPU memory, the main Sequence Searching iterative process is started. The sequences are constructed iteratively, using a stack to store all the intermediate sequences for further analysis. The stack initially contains only the 1 asteroid sequence made of the initial target and time. Within the main loop, the next sequence is taken from the top of the stack, and starting from the end of the sequence, all targets within reach for a given time of flight $tof = tof_0$ are computed on the GPGPU. While there are less than N_{min} feasible targets identified in total, the time of flight is increased by some Δtof, and the search is repeated for this new time of flight. This process continues until either some maximum tof_{max} is reached, or $N > N_{min}$ feasible targets have been identified. If feasible targets have been identified, new sequences are constructed by appending each

identified feasible target to the current sequence, and the new sequences are placed on the top of the stack. If, on the other hand, the maximum time of flight tof_{max} has been exceeded without identifying any further feasible targets (e.g. because the total time of the mission has been exceeded or the fuel reserves have been exhausted), the sequence is considered finished and ranked based on the specific objectives of the problem and stored in the list of results. The process in then repeated with the next sequence on top of the stack until the stack is empty.

As the goal is finding the maximum number of visited targets compatible with the given constraints on available propellant and mission time, it has been assumed that the optimal solutions to select for the single transfer are those that minimizes the transfer time. This is the reason for iterating on the time of flight, starting from a minimum and increasing until at least N_{min} feasible solutions are found. This way, out of all feasible transfers, only the most promising N_{min} ones are considered without inspecting further feasible transfers with longer time of flight.

The algorithm can be tweaked to increase the depth of search with this N_{min} parameter. It has the same goal as elitism in genetic algorithms. It forces not only the single best identified feasible target to be retained but the best N_{min}. Obviously this parameter highly affects the computational time, as it increases exponentially the number of branches and hence sequences to be analyzed. A sensitivity analysis has been performed and a suitable value that allows sufficient flexibility but maintaining a reasonable computational time has been identified. The value identified was $N_{min} = 2$ for quick searches and $N_{min} = 3$ for more in depth searches. Higher values resulted in very large computational times without yielding any improvement in the quality of sequences found.

This module has been coded in C++ and interfaced directly with MATLAB using the calllib interface, so that it can be easily plugged into other modules developed in MATLAB.

B. GPGPU Ephemeris Evaluator

This module is responsible of providing ephemerides for the targets that are needed by the Lambert's problem solver. The implementation of this module is not new, and relies on the well established techniques for ephemerides evaluation provided by JPL[27]. However, the fact that the GPGPU memory is not directly connected to the main CPU memory with a high bandwidth channel, necessitates moving all data regarding the ephemerides evaluation to the GPGPU mem-

ory. Moreover, code that runs on the GPGPU cannot take advantage of already existing libraries developed for the CPU, so in particular the use of the SPICE library[28] is excluded.

The module is implemented starting from the available JPL ephemerides routines, which are used to load data from the binary ephemerides file and provide the position and velocity of the main bodies of the solar system. Those routine are adapted to run as an OpenCL kernel, ready to be executed on a GPGPU, and the portion of the data contained in the binary file necessary for the mission is loaded directly in the GPGPU memory. The only small adjustment for the GPGPU is to remove the variable length loop in the solution of Kepler's equation by a fixed loop of length 3. For what concerns the ephemerides evaluation of the small bodies, they are computed using the analytical solution of the two-body problem, given the osculating orbital elements at a specific date. All the osculating elements for the targets present in the dataset are loaded on the GPGPU memory, so that they can be accessed directly by the GPGPU ephemerides evaluation module.

C. GPGPU Lambert's Problem Solver

The Lambert's problem solver module implements a modified version of the algorithm proposed by R. Battin[18]. Also this module is implemented in OpenCL C, and takes as input the two IDs of the initial and final targets, the starting date and the time of flight, and returns ΔV_l which is the sum of both the initial and final maneuvers.

The algorithm of R. Battin for the solution of the Lambert's problem foresees a series of internal checks to identify possible singularities in the solution; moreover, the solution of several nonlinear equations is required, which is implemented using Newton's method. Those aspects of the original algorithm don't pose any particular problem to the implementation itself, and allows to increase the efficiency of the computation, completely avoiding part of the computations if not needed (via the checks) and providing accurate solutions of the nonlinear equations (via Newton's methods). However, the programming of a GPGPU is different, as reported in Section III, and it is usually better to execute all code in all cases if this allows two separate instance of a kernel to perform the same operations, as this will increase the efficiency of the code execution on the GPGPU.

Following this approach, all the checks have been removed and substituted with the setting of singularity flags, which allows the returning of a high cost ΔV_l to signal an error condition, which

automatically leads to the solution being discarded in the CPU Driver program. While it may seem like a particularly strong restriction, in practice this does not affect the quality of the resulting sequences as in following iterations the time of flight is increased, and a solution for a target which is not near the singularity anymore can be found.

Moreover, all Newton solvers where implemented with a fixed number of iterations. This allows to fix the number of operations in the code, optimal for the SIMD architecture, while possibly not obtaining maximum accuracy for all solutions of the nonlinear equations. However, by choosing the number of iterations high enough, in most cases a sufficiently accurate solution is found. In our code, we chose 5 iterations for all Newton loops in the code.

V. Low Thrust Optimization

The preliminary sequences generated by the GPGPU algorithm described above are not yet complete low-thrust trajectories. Instead, they serve as an initial guess for a full fuel-optimal low-thrust trajectory optimization. We use a code developed by Chen Zhang and Francesco Topputo at Politecnico di Milano [29] that solves the optimal control problem by transforming it into a two-point boundary value problem (TPVBP). The resulting TPBVP is first solved in the easier energy optimal setting where the control is smooth, and then slowly transformed into the fuel optimal setting characterized by a discontinuous bang-bang control by a homotopy between the two cost functions. In the present MATLAB implementation applied to a typical single leg transfer this process typically takes a few seconds to converge to the solution.

This algorithm allows us to obtain the thrust profile $u(t)$ for a single leg, given by an initial asteroid id A_1 and departure time T_1 as well as final asteroid A_2 and arrival time T_2. As the preliminary sequences are based on estimates of the real cost, it is commonly found that a single leg of a sequence does not converge to a solution (i.e. it is not feasible). On the other hand, it is also often the case that a leg overestimates the fuel and time required to perform the low thrust transfer, resulting in a transfer with long coast arcs.

These problems can often be solved by simply making small adjustments to the departure and arrival times of the asteroid sequences. In this process, transfers with long coast arcs are shortened to gain time, while infeasible transfers are given more time to complete. As the asteroid positions

change rather slowly, changing the timing by shifting often does not affect the feasibility of the entire sequence.

In order to perform this full sequence optimization in an automated fashion, we have implemented an algorithm to "wiggle" the sequences into place using a binary search like method. Let the preliminary sequence of asteroids A_i with corresponding departure times T_i^a and T_i^d, $i = 0, 1, \ldots, N$ be given. Furthermore, let ρ be the ratio of the time spent thrusting and the total transfer time for any given low-thrust transfer.

Then with parameters $w > 0$, $\epsilon > 0$, and $0 < \rho_m < 1$ our algorithm for the nth leg works as follows:

1. Compute the transfer time $\Delta T = T_{n+1}^a - T_n^d$ between asteroid A_n and A_{n+1}.
2. Let $\underline{T} = T_{n+1}^a - w\Delta T$ and $\overline{T} = T_{n+1}^a + w\Delta T$ which form the search interval $[\underline{T}, \overline{T}]$ for the new optimized arrival time $\widetilde{T}_{n+1}^a$ of the transfer.
3. For the midpoint $t = (\underline{T} + \overline{T})/2$, compute the low-thrust transfer from (A_n, T_n^d) to (A_{n+1}, t).
4. If the low-thrust transfer is feasible, let $\overline{T} = t$. If $\rho > \rho_m$, terminate the algorithm.
5. If the low-thrust transfer is not feasible, let $\underline{T} = t$
6. Continue the algorithm at step 3 unless $\overline{T} - \underline{T} < \epsilon$.

This procedure yields a new optimized value $\widetilde{T}_{n+1}^a = \overline{T}$ for the arrival time at A_{n+1}. The thrust ratio ρ of that transfer will be larger than ρ_m if possible. The time shift $\delta t = \widetilde{T}_{n+1}^a - T_{n+1}^a$ is applied to all following times T_i^a, T_i^d, $i \geqslant n$, in the sequence. Then the same algorithm is applied to the next leg $n + 1$.

In this way, the arrival time of each leg (and all the following legs) is shifted to enforce for each leg an optimal transfer with at least thrust ratio ρ_m where possible. In our application, we typically set the size of the initial search interval $w = 0.5$, which corresponds to a mission duration between 50% and 150% of the nominal value. The cutoff value ϵ is set to 1 day, so that the algorithm stops when the interval of possible final times becomes smaller than that. The value for ρ_m is somewhat flexible, depending on the needs of the sequence. In general, we found that a lower value for ρ_m,

Table 1 Preliminary sequence of 13 asteroids in the setting of GTOC7

Asteroid ID	T^a [MJD]	T^d [MJD]
566	–	63625
2328	63805	63835
656	64065	64095
1148	64265	64295
5690	64475	64505
3418	64655	64685
7384	64795	64825
2337	64965	64995
7398	65175	65205
12538	65295	65325
14196	65455	65485
1402	65595	65625
14188	65735	65765

that is a less optimal thrust ratio, leads to longer flight times but lower fuel consumption, while a higher value for ρ_m reduces the flight time but typically increases the fuel consumption. Values for the target thrust ratio used in our work lie in the range $0.5 < \rho_m < 1$. The typical runtime for this sequence optimization on an average desktop machine (iMac with a 2.9 GHz Intel i5 CPU and 8 GB DDR3 RAM) is about 30 seconds.

We remark that a thrust ratio of $\rho_m = 0$ means that the very first convergent solution identified will be used. In particular, if a sequence as given is already feasible, the algorithm will simply redo the verification without any changes to the times.

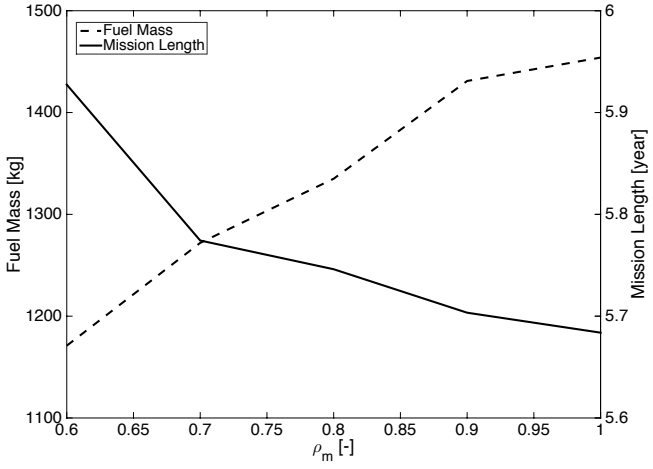

Fig. 5 Low-thrust sequence optimization with various values for the minimum thrust ratio ρ_m versus the resulting consumed fuel mass m_f and total mission length T.

To illustrate the effect of the parameter ρ_m, we use a generic preliminary sequence of 13 asteroids,

given in Table 1, generated by the sequence GPGPU search algorithm. In the settings of the Seventh Global Trajectory Optimization Competition (GTOC 7) problem definition (see Section VI), this sequence is not feasible. That is the transfers given in Table 1 are not possible with the available low-thrust propulsion system. In order to arrive at feasible low-thrust trajectories, the sequence is optimized using various values for ρ_m and the resulting consumed fuel mass m_f and total mission time T is reported in Figure 5. At all levels optimization was possible and a feasible trajectory was identified. A clear trend can be established, showing the growing fuel consumption (rising from 1171 kg to 1454 kg) as the thrust ratio increases, while the mission time is reduced (from 5.9274 years to 5.6838 years). Thus, depending on whether a mission is time constrained or fuel constrained, different values for ρ_m can be chosen to obtain an optimal result.

VI. Analysis

In the following sections we provide analyses for various aspects of the algorithm. In all of this section, we use the following values (derived from the GTOC7 problem statement) for sequence generation:

- Initial probe mass $m_i = 2000$ kg, of which $m_p = 1200$ kg are available fuel.
- Maximum thrust $T_{max} = 0.3$ N and specific impulse $I_{sp} = 3000$ s.
- Maximum mission length $T = 6$ years.

We furthermore use a dataset of asteroid ephemerides provided by Politecnico di Torino for GTOC7. This dataset is very big, it comprises 16256 asteroids of the main asteroid belt of the solar system and thusly provides ample opportunity for our algorithm to show its power.

A. Algorithm Parameter Dependence

As described in Section II, the algorithm depends strongly on two parameters α_f and α_t that allow tuning of the sequence generation. In the following we investigate the dependence of the sequence generation on these parameters.

For each low-thrust transfer the low-thrust propulsion system must be able to provide the necessary amount of ΔV in order to achieve the required change in orbit. This requires two criteria

to be met: firstly, as the change in momentum is not instantaneous there must be enough time during the transfer for the low-thrust propulsion to exert the required thrust. Secondly, there must be enough fuel available to perform this maneuver and the fuel consumption must be estimated. While related, the estimation of the transfer time and the fuel consumption have been found to behave somewhat differently from each other relative to the impulsive Lambert's transfer. For this reason, we have introduced the two correction factors which independently adjust each measure.

We recall that $\alpha_t \in (0, 1]$ is the correction factor for the maximum ΔV obtainable from the low-thrust propulsion over the time of flight, and it is applied before comparing to the ΔV_l required by the Lambert's transfer. As such, a lower value for α_t generally means longer flight time and higher estimated fuel consumption to achieve the same Lambert's transfer. The parameter $\alpha_f \in [1, \infty)$, on the other hand, represents a correction factor for the estimated fuel consumption of a transfer. The higher this value, the more fuel is consumed relative to the Lambert's transfer.

In our experiments we found values of $\alpha_t \in [0.6, 0.8]$ and $\alpha_f \in [1.0, 1.3]$ to yield reasonably accurate preliminary sequences. To illustrate the effect of the parameters in this range, we consider the fixed initial asteroid ID 381 and initial departure time 62233 MJD and analyze the best preliminary sequence found (ranked first by length and then by estimated fuel consumption). For each case, we apply the low-thrust optimization described in Section V with $\rho_m \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$. Out of the resulting low-thrust trajectories that satisfy the mission constraints, the one with the lowest fuel consumption is selected. If none of the low-thrust trajectories satisfy the mission constraints, the one with the smallest fuel consumption is selected. In Table 2 we report the resulting sequence length and, if a low-thrust trajectory was found, the relative error of the estimated vs. the real value for both mission time and consumed fuel mass. For low-thrust trajectories that violates the mission constraints, and thus are not considered feasible, the values that violate the constraint are put in parenthesis. Preliminary sequences for which no low-thrust trajectory could be found at all, are marked with $-*-$.

As can be clearly seen, the effect of α_t is rather straightforward: a higher value leads to longer sequences, which then however tend to fail in the low-thrust optimization. First, the optimization process is able to identify low-thrust trajectories for the preliminary sequences that fail to meet

Table 2 Dependence on the parameters α_t and α_r in the setting of GTOC7. Trajectories violating mission constraints are in parentheses, unfeasible sequences are marked $-*-$

α_f	α_t:	0.6	0.65	0.7	0.75	0.8
	Length:	11	12	12	13	14
1.0	Time:	4.37%	(−4.5%)	1,42%	$-*-$	$-*-$
	Fuel:	−22.5%	(−22.1%)	(−30.8%)	$-*-$	$-*-$
	Length:	11	12	12	13	13
1.1	Time:	4.38%	(−4.3%)	1.42%	$-*-$	$-*-$
	Fuel:	−16.9%	(−16,7%)	(−26.2%)	$-*-$	$-*-$
	Length:	11	12	12	12	13
1.2	Time:	4.39%	(−4.48%)	1.42%	(−8.07%)	$-*-$
	Fuel:	−11.6%	(−11.7%)	(−21.8%)	−4.87%	$-*-$
	Length:	11	12	12	12	12
1.3	Time:	4.4%	0.00%	1.42%	(−8.07%)	$-*-$
	Fuel:	−6.52%	−12.0%	(−17.7%)	0.00%	$-*-$

the problem restrictions, but for values of $\alpha_t > 0.7$ the algorithm is not even able to obtain any low-thrust trajectories any more as the preliminary sequences are too far from feasible. Thus α_t can serve as a measure of how aggressive the search is.

The interpretation of α_f is a bit more difficult. Its main effect on sequence generation is that the estimate of the actual fuel usage becomes more accurate. For the (rather unrealistic) $\alpha_f = 1.0$, the only feasible sequence underestimates the actual fuel consumption by -22.5%. As α_f increases, the underestimation drops to only about -6.52% at $\alpha_f = 1.3$. Furthermore, at high values of α_t it becomes evident that α_f restricts the length of the sequences found. This is because the probe runs out of fuel faster at higher values for α_f, even if a high α_t would allow the probe to reach more asteroids.

We remark that as sequences become increasingly unfeasible in the low-thrust optimization, the low-thrust optimization procedure applied aggressively changes the times of the sequences in order to make the optimal control problem converge. As shown in Section V, this basically leads to a trade-off between the flight time and the fuel consumption. As a result, large variations in sequence quality are possible, as can be seen e.g. for $\alpha_t = 0.75$ and $\alpha_f = 1.2$ and 1.3. After the optimization, both sequences (which were quite similar to start with) became very accurate in the fuel consumption, while violating the time constraints. However, this effect is not due to the

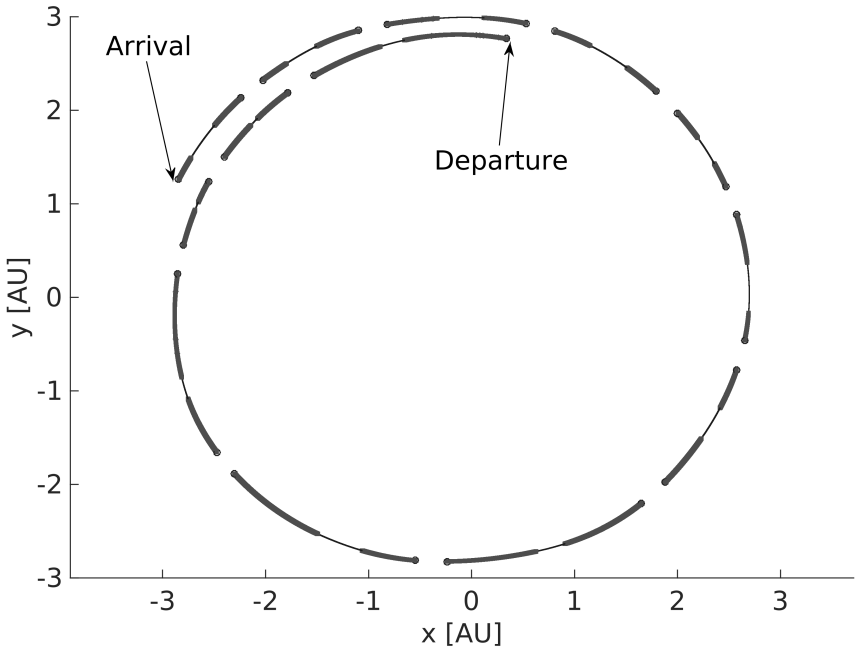

Fig. 6 Trajectory of a sequence of 14 asteroids in the setting of GTOC7.

GPGPU sequencing algorithm and in general of course does not hold for these particular parameter values. It is merely a side effect of the low-thrust optimization process.

In conclusion, while more aggressive values of α_t and α_f yield more impressive looking initial sequences, the best results in the particular test case shown here are obtained with rather conservative values of $\alpha_t = 0.65$ and $\alpha_f = 1.3$. In our experience those values very consistently yield good, easily optimizable preliminary sequences.

B. Long Sequence Search

As one of the main goals of sequence generation is the identification of particularly long feasible sequences, we can tune the algorithm in particular to find such sequences. From the above sensitivity analysis, we conclude that a setting of $\alpha_f = 1.3$ is necessary in order to obtain reasonable estimates of the fuel consumption and hence make the sequences feasible. A higher setting could be selected to be on the safe side and estimate fuel consumption more accurately. However, as fuel consumption is a cutoff criterion for sequence generation, we would rather underestimate the consumption slightly in order to obtain more aggressive preliminary sequences. For α_t we chose a value of $\alpha_t = 0.68$ instead, again operating at the upper limit of where sequences appear to be feasible.

A random search in the asteroid set indicates that for the given mission constraints, it is relatively easy to identify feasible sequences of length 12 and also 13 sequences appear quite frequently. Often long sequences that violate the mission constraints just barely can be made feasible by careful manual adjustment after the automatic optimization algorithm described in Section V is run.

Consider as an example the case of a 14-asteroid sequence that was identified in the random search. After the automatic optimization process, it violated the mission constraints by less than 0.1% in fuel consumption. After a manual adjustment of the last few legs of the sequence, it was possible to obtain a feasible 14-asteroid sequence given in Table 3. The fuel mass consumed by this sequence is 1199.89 kg and it takes 5.947 years to complete the mission.

Figure 6 illustrates the corresponding orbit of the probe around the Sun. Thrusting arcs are reported as thick lines, while the coasting arcs are reported as thin lines. As can be seen, the sequence is nearly time optimal, with only short coast arcs in the middle. This sequence is very typical for the trajectory of a "good" sequence. Motion starts at an asteroid closer to the Sun, and then spirals outward counter-clockwise in a more or less circular motion, hopping from asteroid to asteroid. Physically, this is of course expected as moving outward requires less energy than moving inward and the rendezvous condition with the asteroids imposes that the probe move along with the asteroids in counter-clockwise motion. However, we remark that none of these considerations were programmed into the algorithm, and as such it is remarkable that the solution identified exhibits these physical properties so clearly.

Unfortunately, despite our best efforts, we were unable to locate any 15-asteroid sequences satisfying the mission constraints.

C. Comparison with CPU Implementation

The Sequence Search Algorithm performances has been compared with a branch & bound algorithm based on the same consideration laid out in Section II but implemented to run on a CPU within MATLAB. The tests have been performed on the following two different systems running the same Linux distribution, with the same kernel version and the current version of the proprietary graphical drivers at the time:

Table 3 Fully optimized sequence of 14 asteroids in the setting of GTOC7

Asteroid ID	T^a [MJD]	T^d [MJD]
14196	–	65085.1
2337	65280.0	65310.0
7384	65398.4	65428.4
10990	65499.3	65529.3
3838	65738.6	65768.6
3919	65972.3	66002.3
566	66196.1	66226.1
5727	66353.2	66383.2
9645	66510.3	66540.3
963	66629.3	66659.3
5039	66778.0	66808.0
1403	66943.8	66973.8
2993	67087.4	67117.4
8472	67227.4	67257.4

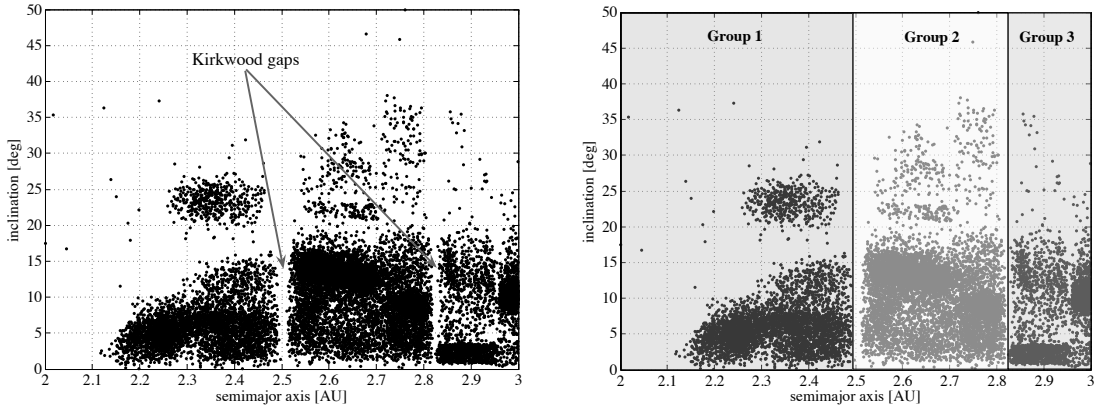

Fig. 7 Visualization of the asteroid distribution in the GTOC7 dataset.

- System 1: Intel Core i7-4820K @3.7GHz 32GB, (2) NVIDIA Tesla K20c 5GB
- System 2: AMD Phenom II X6 1075T @3.0GHz 8GB, AMD Radeon HD7950 3GB

For the test we search sequences starting from Asteroid with ID 381 starting at 62233 MJD. For the CPU algorithm, however, it was necessary to limit the search to a subset of asteroids in order to keep the runtime of the algorithm sustainable.

In Figure 7 the semi-major axis and inclination of the entire dataset of available asteroids is shown. It is evident that the asteroids are separated in three main groups by the Kirkwood gaps. A natural idea is therefore to apply a pre-filtering to the data set, trying to reduce the number of asteroid in each filtered data set.

The search on the CPU has been limited to group 3 (the farthest from the sun) pre-filtered

Table 4 Performance comparison of the massively parallel GPGPU implementation with CPU branch & bound

Platform	Asteroids	Seq. Length	Time System 1 [s]			Time System 2 [s]		
			CPU	GPGPU	Sum	CPU	GPGPU	Sum
CPU	1013	11	57	–	57	61	–	61
GPGPU	16256	12	9	3.5	12.5	9	7	16

with a relative inclination of the orbital plane with respect to asteroid 381 of ±5 deg. The resulting set comprises only 1013 of the initial 16256 asteroids. No pre-filtering is needed with the GPGPU Sequence Searching Algorithm as the performance difference between the full data set and the reduced data set is minimal.

The results of the comparison are shown in Table 4. Looking at the results of the test, it is apparent that the CPU version of the searching algorithm is performing roughly equally on comparable CPUs, while the GPGPU Algorithm is more strongly affected by the GPGPU model. On both the systems analyzed the time spent on the CPU is the same. However, the time spent on the GPGPU is highly different. The NVIDIA GPGPU is clearly faster and more efficient. Obviously the cards are not of the same class, the NVIDIA Tesla card used here was the best available general purpose computing card at the time of the test, while the AMD is merely an high-end graphics card. We note, however, that taking into account only the cost of the two cards, the AMD card is the winner, as it costs only about 10% of the NVIDIA one, while providing about half of the performance.

Another conclusion that can be drawn from the results is that limiting the search to a pre-filtered group is clearly affecting the quality of the search, as evidenced by the shorter sequences found by the CPU algorithm.

In this comparison, we do not consider in detail other differences in the quality of the results such as the total number of high-quality sequences returned or the distribution of the results in the overall search space. However, we have observed that the GPGPU sequencing algorithm yields very good results that can typically be optimized into true low-thrust trajectories, and the identified sequences are typically quite different, indicating a good global distribution in the search space. The same was not found for the CPU algorithm, which generated very similar sequences.

VII. Application to GTOC 7

The Global Trajectory Optimisation Competition was inaugurated in 2005 by the Advanced Concepts Team at the European Space Agency. After the first edition, following competitions were organized by the winning team of the preceding GTOC edition. The work presented in this paper has initially been developed to find a solution of the 7th edition of the GTOC.

The problem posed by GTOC7 was the following: A mothership equipped with nuclear thermal propulsion (impulsive) is sent from Earth to the main asteroid belt and carries three low-thrust probes that need to be released at appropriate times and gather science on as many asteroid as possible (via a rendezvous) before returning to the mothership to deliver their findings.

The initial mass of each probe is 2000 kg, of which 1200 kg are available fuel mass. The characteristics of the low-thrust propulsion system on each probe are a maximum continuous thrust of 0.3 N and a specific impulse I_{sp} of 3000 s. Each probe has a lifetime of 6 years after departing from the mothership, before the end of which it must be picked up again by the mothership via a rendezvous. The complete mission duration from Earth departure until rendezvous with the last probe is 12 years. The dynamics of the problem are assumed to be those of a pure two-body problem between the mothership or probe and the Sun. Lastly, the GTOC7 problem description provides a dataset of 16256 asteroids from which to chose the targets for the rendezvous.

The GPGPU Sequence Search Algorithm has been integrated in the complete mothership optimization to find suitable sequences for the three probes of GTOC7.The mothership strategy imposes that all the three probes depart form the same asteroid and arrive at the same asteroid (different from the initial one), thus allowing an easier optimization of the mothership maneuvers. This is not a limitation of the Sequence Search Algorithm, but of the strategy employed for the optimization of the Mothership trajectory.

As this paper focuses on the sequence generation, we will not present the Mothership trajectory optimization in detail. Instead, we show how the features of the Sequence Search Algorithm allow an easy identification of Mothership solutions compatible with the low-thrust sequences of the three probes. As reported in Section IV the algorithm constructs the sequence starting from a given initial asteroid, but is a priori not capable of constraining also the final one.

Table 5 Final sequences for GTOC7 probes with identical initial and final asteroids highlighted

Probe 1		*Probe 2*		*Probe 3*	
Asteroid ID	T^d [MJD]	Asteroid ID	T^d [MJD]	Asteroid ID	T^d [MJD]
3326	62637	**3326**	63350	**3326**	63653
4828	62912	11037	63630	16080	63863
3749	63162	3418	63875	4998	64128
8149	63357	3899	64080	12543	64323
10943	63607	9645	64300	1891	64483
5711	63817	6575	64495	10772	64683
9646	64037	1402	64725	656	64818
8442	64237	16193	64840	3016	65018
12527	64397	8472	65040	1906	65163
5727	64577	531	65220	5046	65343
3412	64757	4393	65360	12614	65523
		3412	65480	**3412**	65703

However, the GPGPU algorithm is capable of returning all possible sequences ranked by their quality. Thus, it is possible to search within the returned sequences of the three probes for those that have a common final asteroid, or at least a common asteroid among the last few asteroids visited in each sequence. This approach is possible only in the GPGPU version of the algorithm that considers the entire dataset, thus allowing the identification of a large number of high-quality sequences for each given starting asteroid and time (on the order of $500 - 1000$ in our the test case) which are distributed broadly over the entire search space.

Implementing such a search, inside an evolutionary optimizer that also takes into account the mothership trajectory, the sequences in Table 5 have been identified. These sequences were optimized using the low-thrust optimization method in Section V which yielded the the trajectories for the probes shown in Figure 8. In the figure the thrusting arcs are reported as thick lines, while the coasting arcs are shown as thin lines. It is clear that the GPGPU Sequence Search algorithm found indeed feasible solutions, which are very near to the optimal one, as the optimized low-thrust trajectory is thrusting almost all time at maximum thrust as shown exemplary by the thrust profile of probe 1, indicating that the solutions are very close to time-optimal.

With the sequences identified here, along with the Mothership trajectory, we were able to achieve 7th place in the GTOC competition. After studying the results of other the teams, we believe that the main weakness of our approach is the Mothership strategy. The quality of the low-thrust sequences we can generate using out algorithm seems to us to be the same as that of the

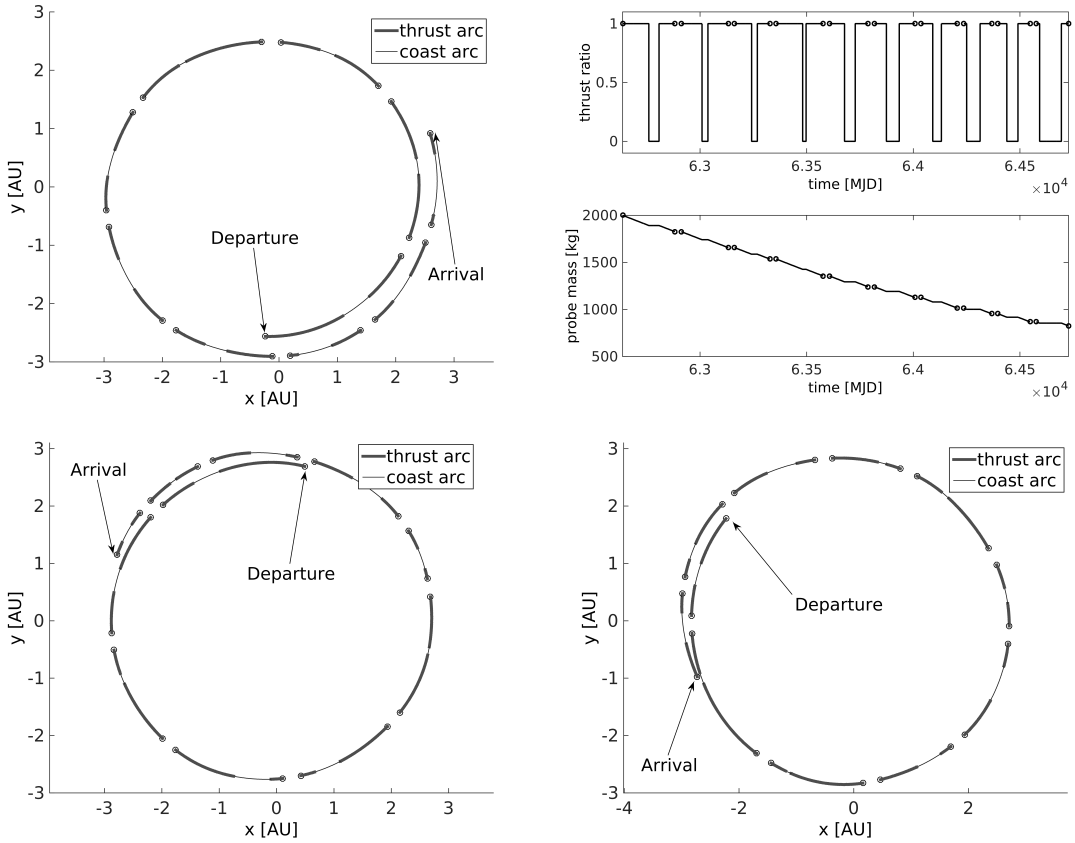

Fig. 8 Optimized trajectories of the three probes for our GTOC7 solution and thrust profile of the first probe.

top ranking teams.

VIII. Conclusions

The presented massively parallel implementation of a GPGPU Sequence Searching Algorithm for MRLT missions has been shown to be very powerful in identifying optimal sequences. The effect of the various parameters of the algorithm on the resulting sequence generation have been analyzed. The GPGPU implementation has been compared with a branch-and-bound implementation on the CPU. It has been found that the GPGPU implementation yields significantly better results both in terms of sequence quality as well as computational time. Finally, the results of the application to the GTOC7 problem demonstrate the efficiency of the proposed method on very complex problems, allowing a more extensive search in less time than the CPU implementation.

The approach presented here is not limited to missions to asteroids, its potential could also be exploited very well in the planning of multiple debris removal missions or in any other application

characterized by a search space that combines both continuous and discrete variables.

IX. Acknowledgments

The authors would like to thank C. Zhang and F. Topputo for their low-thrust trajectory optimizer, as well as C. Colombo and F. Letizia for their CPU implementation of a branch-and-bound algorithm and A. Morselli for his work on the Mothership trajectory optimization code. The authors are grateful for the generous support of this work by NVIDIA Corporation through the donation of two NVIDIA Tesla K20c GPGPU accelerator cards within the NVIDIA academic partnership program. A. Wittig gratefully acknowledges the support received from the EU Marie Curie network AstroNet-II (PITN-GA 2011-289240).

References

[1] "Global Trajectory Optimization Portal http://sophia.estec.esa.int/gtoc_portal/,"

[2] M. Vasile and P. De Pascale, "Preliminary Design of Multiple Gravity-Assist Trajectories," *J. Spacecraft and Rockets*, Vol. 43, No. 4, 2006.

[3] B. Conway, *Spacecraft Trajectory Optimization*. Cambridge Aerospace Series, Cambridge University Press, 2010.

[4] J. Stuart, K. Howell, and R. Wilson, "Design of End-to-End Trojan Asteroid Rendezvous Tours Incorporating Potential Scientific Value," *Advances in the Astronautical Sciences Spaceflight Mechanics*, Vol. 152, 2014, pp. 1–20. AAS 14-267.

[5] J. Clausen, "Branch and bound algorithms-principles and examples," *Department of Computer Science, University of Copenhagen*, 1999, pp. 1–30.

[6] K. Alemany and R. Braun, "Design Space Pruning Techniques for Low-Thrust, Multiple Asteroid Rendezvous Trajectory Design," *Advances in the Astronautical Sciences*, Vol. 129, 2007, pp. 1–12. AAS 07-348.

[7] NVIDIA Inc., "NVIDIA Tesla Accelerated Computing http://www.nvidia.com/object/tesla-supercomputing-solutions.html,"

[8] J. Nickolls and W. Dally, "The GPU Computing Era," *IEEE Micro*, Vol. 30, March 2010, pp. 56–69, 10.1109/MM.2010.41.

[9] *CUDA C Programming Guide*. NVIDIA Inc., 2015.

[10] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. Purcell, "A survey of general-purpose computation on graphics hardware," 2007.

[11] J. Bolz, I. Farmer, E. Grinspun, and P. Schröoder, "Sparse matrix solvers on the GPU: conjugate gradients and multigrid," *ACM Transactions on Graphics (TOG)*, Vol. 22, ACM, 2003, pp. 917–924.

[12] A. Corrigan, F. Camelli, R. Löhner, and F. Mut, "Semi-automatic porting of a large-scale Fortran CFD code to GPUs," *International Journal for Numerical Methods in Fluids*, Vol. 69, No. 2, 2012, pp. 314–331, 10.1002/fld.2560.

[13] L. Nyland, M. Harris, and J. Prins, *Fast N-Body Simulation with CUDA*, Vol. Part 3: Geometry of *GPU Gems*, ch. 31, p. 677–695. NVIDIA Corp., 2008.

[14] N. Arora, V. Vittaldev, and R. P. Russell, "Parallel Computation of Trajectories Using Graphics Processing Units and Interpolated Gravity Models," *Journal of Guidance, Control, and Dynamics*, June 2015, pp. 1–11, 10.2514/1.G000571.

[15] N. Nakhjiri and B. F. Villac, "An Algorithm for Trajectory Propagation and Uncertainty Mapping on GPU," *23rd AAS/AIAA Space Flight Mechanics Meeting*, American Astronomical Society, 2013. 2013-376.

[16] H. Shen, V. Vittaldev, C. D. Karlgaard, R. P. Russell, and E. Pellegrini, "Parallelized Sigma Point and Particle Filters for Navigation Problems," *36th Annual AAS Guidance and Control Conference*, American Astronomical Society, 2013. 2013-034.

[17] Khronos Group, "OpenCL The open standard for parallel programming of heterogeneous systems https://www.khronos.org/opencl/,"

[18] R. H. Battin, *An introduction to the mathematics and methods of astrodynamics*. Aiaa, 1999.

[19] C. Brown, *Spacecraft Propulsion*. AIAA education series, American Institute of Aeronautics & Astronautics, 1996.

[20] J. Owens, "GPU architecture overview," *ACM SIGGRAPH*, Vol. 1, 2007, pp. 5–9.

[21] NVIDIA Inc., "Tesla K20 GPU Accelerator," July 2013.

[22] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," *Computational Science Engineering, IEEE*, Vol. 5, Jan 1998, pp. 46–55, 10.1109/99.660313.

[23] H. Attiya and J. Welch, *Distributed computing: fundamentals, simulations, and advanced topics*, Vol. 19. John Wiley & Sons, 2004.

[24] Khronos OpenCL Working Group, *The OpenCL Specification, Version 2.0*. Khronos Group, revision 22 ed., 2014.

[25] J. Fang, A. Varbanescu, and H. Sips, "A Comprehensive Performance Comparison of CUDA and OpenCL," *International Conference on Parallel Processing (ICPP)*, September 2011, pp. 216–225, 10.1109/ICPP.2011.45.

[26] K. Komatsu, K. Sato, Y. Arai, K. Koyama, H. Takizawa, and H. Kobayashi, "Evaluating Performance and Portability of OpenCL Programs," *Fifth International Workshop on Automatic Performance Tuning*, June 2010.

[27] X. Newhall, E. M. Standish, and J. G. Williams, "DE 102-A numerically integrated ephemeris of the moon and planets spanning forty-four centuries," *Astronomy and Astrophysics*, Vol. 125, 1983, pp. 150–167.

[28] C. Acton, "Ancillary data services of NASA's Navigation and Ancillary Information Facility," *Planetary and Space Science*, Vol. 44, No. 1, 1996, pp. 65 – 70. Planetary data system, http://dx.doi.org/10.1016/0032-0633(95)00107-7.

[29] C. Zhang, F. Topputo, F. Bernelli-Zazzera, and Y. Zhao, "Low-Thrust Minimum Fuel Optimization in the Circular Restricted Three-Body Model," *2nd IAA Conference on Dynamics and Control of Space Systems (DyCoSS 2014)*, Rome, March 2014, pp. 1–21. IAA-AAS-DyCoSS2-14-09.