AAS 10 - 198

A GPU ACCELERATED MULTIPLE REVOLUTION LAMBERT SOLVER FOR FAST MISSION DESIGN

Nitin Arora*and Ryan P. Russell†

Lambert's algorithm acts as an enabler for a large variety of mission design problems. Typically an overwhelmingly large number of Lambert solutions are needed to identify sets of mission feasible trajectories. We propose a Graphics Processing Unit (GPU) accelerated multiple revolution Lambert solver to combat this computationally expensive combinatorial problem. The implementation introduces a simple initial guess generator that exploits the inherent structure of the well-known Lambert function formulated in universal variables. Further, the approach builds from the concepts of parallel heterogeneous programming utilizing both the central processing unit (CPU) and GPU in tandem to achieve multiple orders of magnitude speedup. The solution strategy is transparent and scalable to specific user resources. Speedups of two orders of magnitudes are found using a state of the art GPU on a single personal workstation, while single order of magnitude speedups are observed using the GPU on a common laptop. Example gravity assisted flyby trajectories are used to demonstrate performance and potential applications.

INTRODUCTION

Lambert's problem is one of the most significant and extensively studied problems in astrodynamics.[1–5] Given two points relative to a point mass gravitating body and time of flight, a Lambert algorithm computes all possible Keplerian transfers. A variety of contributors have provided unique soluiton methods, yet the Lambert problem is always reduced to a one dimensional root-solve of a trancendental function. It was first introduced by Lambert in 1761 and subsequently extended by Gauss.[4] With the arrival of the space age, the multiple revolution (multi-rev) Lambert's problem has since been studied and applied to a wide variety of space mission applications.

There exists a plethora of literature discussing various approaches developed over the years to solve the Lambert problem. Most of these solutions techniques can be divided in two general types: 1) direct geometry based methods and 2) universal variable based methods. The types are characterized primarily by their choice of the iteration variable. The direct geometry based methods iterate in the conventional space of orbit elements to solve some equivalent Lambert equation. Escobal[5] in his text gives multiple approaches where the iterates include semimajor axis, true anomaly, semiparameter, eccentricity and the f and g series. He presents only the zero-rev formulations but extending such techniques to multi-rev formulations is relatively straight forward. The work done by Ochoa and Prussing[6,7] during early 1990s extends another geometry based approach, the Lagrange formulation, to its full multi-rev case. Their approach is robust and therefore it is commonly employed. However it suffers from three main drawbacks that are similar to most methods in its class: 1) the method is valid for elliptical orbits only, 2) the iteration variable is semi-major axis and is therefore unbounded which can lead to numerical problems, and 3) in the multi-rev case, the bottom of the two solution branches is not single valued in flight time, causing notational complexity and a tedious implementation. A recent addition to this class of methods is the eccentricity-vector based solution by Avanzini[8] which has been more recently extended to multi-revs by He et. al.[9] Other approaches in this category include the p-iteration

*Graduate Student, Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, 270 Ferst Drive, Atlanta, GA, 30332-404-483-7015,N.arora9@gatech.edu
†Assistant Professor, Guggenheim School of Aerospace Engineering, Georgia Institute of Technology, 270 Ferst Drive, Atlanta, GA, 30332-0150, 404-385-3342 (voice), 404-894-2760 (fax),ryan.russell@gatech.edu

method by Herrick and Liu[10] and the classical method by Gauss himself.[11] Another recent solution strategy includes the series inversion solution by Throne.[12]

The need for a universal (valid for all conics) and numerically robust solution technique led to the "universal variable" solution of the Lambert problem.[2,3,13–16] This approach is an efficient alternative solution method that combats many if not all of the shortcomings of the Lagrange and other methods that iterate directly on orbital elements. The transformation to an auxiliary variable that is better behaved than one of the anomalies was first introduced by Sundman. This transformation was later applied to a "unified" time of flight formulation (for all types of conics) by Battin.[3,15] It was shortly thereafter that Lancaster and his colleagues published in their short note[1] (and later in a more detailed technical report[14]) the first universal solution to the complete multi-rev Lambert problem. Battin was soon to follow with his own universal approach that was highly tuned for computer implementation using hypergeometric functions and continued fractions.[3,17] Meanwhile, Gooding[2] extended the work of Lancaster by formulating an initial guess generator using rational functions and the higher order Halley's method for rapid root solving. Battin's method is mathematically elegant and computationally efficient, yet is not as intuitive as other approaches. Gooding was able to produce an accurate Lambert algorithm but relied on first, second and third derivatives of the root function for rapid convergence. Another approach first proposed by Bate, Mueller and White[15] utilizes a simple transformation on the standard universal variable and results in just one root function valid for all revolutions. The search variable is related to the transfer angle and is therefore easily bounded, except in the case of the hyperbola. Like all universal formulations, the full domain of the iteration variable is single valued. Overall, the implementation is straight-forward and computationally efficient.

Due to its general formulation and wide applicability, the solution to the multi-rev Lambert problem acts as a building block for various problems like grand tour design,[18–22] interplanetary trajectory optimization,[19,23,24] and orbit determination.[25,26] Legacy codes that compute and optimize ballistic trajectories often require excessive numbers of Lambert solutions.[19,27] To make matters worse, inclusion of intermediate flybys and maneuvers further compounds the computational burden. The sheer number of Lambert calls needed plus the cost of CPU clusters that are often employed as a solution strategy make the problem very expensive to solve in terms of time and computer resources. Accordingly, to narrow the search space and provide for tractable problems, heuristic pruning techniques are generally required in practice. Optimal and perhaps mission enabling solutions therefore may be overlooked.

Parallel Heterogeneous Programming (PHP) [28] along with Nvidia's Compute Unified Device Architecture (CUDA)[29] technology provides a new context for combating the large design space and long runtimes of the combinatorial Lambert problem. The inherent fine grain parallelism of this problem is ideal for the new Nvidia Graphics Processing Unit (GPU) Tesla Architecture. Dramatic orders of magnitude speedups are possible if the underlying algorithms can be modified to adapt to the GPU architecture.[28,30]

We propose CULAM, a GPU implementation of the multi-rev Lambert algorithm enhanced with a new simple initial guess strategy designed to reduce iterations and balance the distributed workloads. The Lambert algorithm is formulated as a multi-rev extension to the universal variable algorithm by Bate et. al.[15] Accordingly, the cases of the hyperbola, parabola, and n-rev ellipse are considered using a single formulation. We choose not to adopt the efficient Goodings approach as it requires computing third derivatives. The use of higher derivatives on the GPU taxes the already limited fast-memory resources. We also note that the current GPU hardware is not able to perform recursions and therefore Battin's universal algorithm is not currently suitable for the GPU.

The paper starts by highlighting the key components of the universal variable multi-rev algorithm. Next, we provide a detailed description of the initial guess generation technique and the GPU accelerated multi-rev Lambert algorithm (CULAM). For comparison purposes we also implement a tuned, robust CPU implementation of the algorithm utilizing analytic derivatives and a bounded Newton-Raphson method with a trust region safeguard. A detailed direct comparison between CULAM and the CPU implementation is carried out and speedups of multiple orders of magnitude are reported. Finally we demonstrate the performance of CULAM on example gravity assisted flyby trajectories in both the interplanetary and planetary satellite design spaces.

We note that the short periods of planetary satellites allow for mission feasible solutions with potentially many tens of revolutions. Therefore, this problem is orders of magnitude more difficult than its interplanetary cousin. The time efficient solution to this planetary satellite combinatorial Lambert problem is thus a main motivator of this study, especially in the context of the renewed interest by the space community to send flagship missions to the outer planet systems. * †

MULTIPLE REVOLUTION LAMBERT SOLVER

Keeping in mind the GPU hardware, we adopt the universal variable formulation of Bate et al and extend it to multiple revolutions. This extension to multiple revolutions utilizes analytic derivaitves and a typical Newton-Raphson method for the required minimizations and root solving procedures. We present a relevant overview plus the multi-rev extension here, but refer the interested reader to Ref[15] for details. A general diagram of the Lambert geometry is depicted in Fig. 1

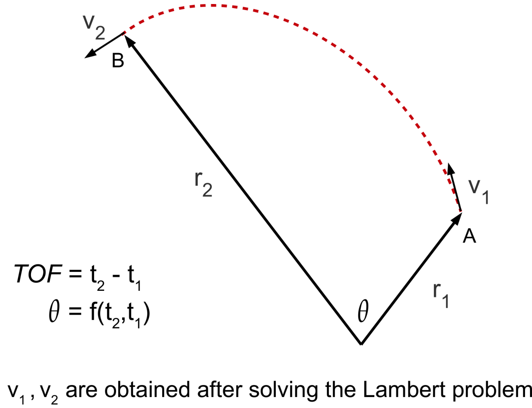


Figure 1. General Lambert geometry for transfer between two bodies with prescribed positions

Vectors $\vec{r_1}$ and $\vec{r_2}$ are the position vectors of bodies 1 and 2 corresponding to times t_1 and t_2 respectively. TOF is the time of flight for the transfer and θ is the angle between the two position vectors. All possible arcs connecting the two points A and B with the required TOF and which satisfy Keplerian motion are possible Lambert solutions. There are $2nmax^D + 1$ direct solutions and $2nmax^R + 1$ retrograde solutions where $nmax^D$ and $nmax^R$ are the maximum number of revolutions possible for the direct and retrograde cases respectively. Provided T^*, each solution is found via a one-dimensional root-solve of the trancendental function $TOF(z) = T^*$. We refer to the $TOF(z)$ function as the Lambert Universal Function (LUF) and T^* as the target value of TOF.

The LUF is given by Eq 1, where Y is Bate's auxiliary variable, $C(z)$ and $S(z)$ are Stumpff functions of z (Eqs 5 and 6). A is a function of geometry and the direction of motion and is given by Eq 7.

$$\sqrt{\mu}TOF = \sqrt{Y}\,(k2Y + A) \tag{1}$$

$$Y(z) = |\vec{r_1}| + |\vec{r_2}| - k1A \tag{2}$$

$$k1(z) = \frac{1 - zS}{\sqrt{C}} \tag{3}$$

$$k2(z) = \frac{S}{C\sqrt{C}} \tag{4}$$

*http://opfm.jpl.nasa.gov/files/EJSM%20Summary%20Report%20Final%20for%20Print_090120_rk.pdf - cited Feb 25 2010

†http://opfm.jpl.nasa.gov/files/TSSM_Joint%20Summary%20Report_Public%20Version_090120.pdf - cited Feb 25 2010

$$C(z) = \begin{cases} \frac{1-cosh(\sqrt{-z})}{z}, & \text{for } z < 0 \\ \Sigma_{k=0}^{\infty} \frac{(-z)^k}{(2k+2)!}, & \text{for } z \sim 0 \\ \frac{1-cos(\sqrt{z})}{z}, & \text{for } z > 0 \end{cases} \tag{5}$$

$$S(z) = \begin{cases} \frac{sinh(\sqrt{-z})-\sqrt{z}}{\sqrt{(-z)^3}}, & \text{for } z < 0 \\ \Sigma_{k=0}^{\infty} \frac{(-z)^k}{(2k+3)!}, & \text{for } z \sim 0 \\ \frac{\sqrt{z}-sin(\sqrt{z})}{\sqrt{(z)^3}}, & \text{for } z > 0 \end{cases} \tag{6}$$

$$A = \frac{\sqrt{|\vec{r_1}||\vec{r_2}|}sin(\theta)}{\sqrt{1-cos(\theta)}} \tag{7}$$

We note that there are sign errors in Bate et. al[15] for the $C(z)'$ and $S(z)'$ (derivatives of Stumpff functions wrt z) series representations. We also note that the auxiliary functions $k1$ and $k2$ are only functions of z. Their independence of the transfer geometry will be useful later in our efforts to derive a simple initial guess strategy. Contrary to the other universal variable methods [Lancaster, Gooding, Battin] and the classic Lagrange approach, we note that the root solving function is independent of n, the number of revolutions. Because z is the eccentric anomaly squared, to search for a n rev solution we simply adjust the bounds of z appropriately.

All gradient based root solving methods require at least the first derivative of the root function. The multi-rev case, as shown in Fig. 2, includes a minimum for each value of n. These minimizations require at least the second derivative and are necessary to identify $nmax$ and robustly bound any root-solve. The derivatives of LUF are given in Eqs 8 and 9. A Newton-Raphson technique is used to implement both the minimzation and the root solving phases.

$$\frac{d(LUF)}{dz} = \frac{x^3}{\sqrt{\mu}}\left(S' - \frac{3SC'}{2C}\right) + \frac{A}{8\sqrt{\mu}}\left(\frac{3S\sqrt{Y}}{C} + \frac{A}{x}\right) \tag{8}$$

$$\frac{d(LUF)^2}{dz^2} = -\frac{(s1+s2+s3+s4)}{16\sqrt{\mu}(C^2\sqrt{y}x^2)} \quad \text{where} \tag{9}$$

$$q = 0.25\,A\sqrt{C} - x^2C'$$

$$s1 = -24\,qx^3C\sqrt{y}S'$$

$$s2 = 36\,qx^3\sqrt{y}SC' - 16\,x^5\sqrt{y}S''\,C^2$$

$$s3 = 24\,x^5\sqrt{y}\left(S'\,C'\,C + SC''\,C - SC'^2\right) - 6\,AS'\,yCx^2$$

$$s4 = -0.75\,A^2SC^{3/2}x^2 + 6\,ASyC'\,x^2 + \frac{A^2C\left(0.25\,A\sqrt{C} - x^2C(z)\right)\sqrt{y}}{x}$$

Figure 2 shows the LUF for a representative $\vec{r_1}$ and $\vec{r_2}$ geometry and direction of motion. Note that a very similar plot to Fig. 2 exists for the same $\vec{r_1}$and $\vec{r_2}$ but with opposite direction of motion. The function can be divided into four regions based on the value of the iteration variable z as follows. We note that z is related to the universal variable x by the relation $z = \frac{x^2}{a}$.

$$\text{Type of transfer} \begin{cases} z < 0, & \text{Hyperbolic} \\ z = 0, & \text{Pararbolic} \\ 0 < z < 4\pi^2, & \text{Zero-rev elliptical} \\ 4n^2\pi^2 < z < 4(n+1)^2\pi^2, & \text{n revolutions elliptical} \end{cases}$$

Each revolution transfer has a minimum possible time of flight associated with it (t_{bi}), as shown in Fig. 2. Hence, if T^* for the transfer is less than t_{bi} for a given revolution i, then the ith revolution solution does not exist. If a solution exists, then the accurate location of the associated z_{bi} provides robust bounds for the ensuing root search on each branch. The basic multi-rev Lambert solution strategy can be divided into three main phases (assuming a known direction of motion).

1. Minimzation phase : Root solve $\frac{dTOF}{dz} = 0$ to get values of t_{bi} and z_{bi} for all $i = 1 \ldots nmax$

2. Root solve phase : Given T^*, bound each search based on n and z_{bi}, and root solve the LUF for z_i^*, for $i = -2nmax \ldots 2nmax$; where $i < 0$ is a long period solution (left branch), $i = 0$ is the zero-rev case, and $i > 0$ is a short period solution (right branch)

3. Solution phase : Use the converged z to compute the values of $\vec{v_1}$ and $\vec{v_2}$ using the f and g functions[15]

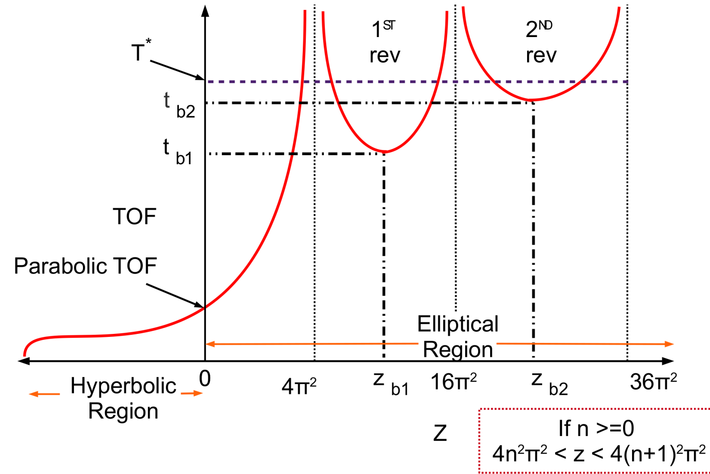


Figure 2. Representative Lambert Universal Function (LUF)

INITIAL GUESS GENERATION

A robust Lambert solver relies on a fast and efficient root solving algorithm. The Newton-Rapshon method is very attractive due to its quadratic convergence property. However the convergence of any root solver benefits from a close initial starting guess. The lack of such starting points has led researchers to adopt other less efficient root solving algorithms (like the bisection method) for solving the Lambert's problem.[4] On the other hand, Gooding's method demonstrates robust convergence with very few iterations using an accurate although admittedly complicated initial guess strategy combined with a high-order solution method. With the goal of a GPU implementation, we seek the middle ground: that is a first order solution method and a simple initial guess strategy. Accordingly, we take advantage of the known properties of LUF to provide a simple approximation for improved initial guesses. The approximation improves robustness and reduces the required iterations. More importantly, the workload balance is more evenly distributed in the context of solving many problems concurrently in parallel.

We observe that the basic structure of the LUF is independent of the problem geometry. The most obvious property perhaps is the convexity of the multi-rev regions. The apparent next step then is to fit a low order interpolating function using a limited number of function calls. As mentioned already, the bottom point of each curve is already identified for robustly bounding the soluiton. We can use this known bottom location and its derivative values to further reduce the number of required interpolation points. In the following section we discuss suitable interpolation functions for both the single and multi-rev cases. Noting that the multi-rev calls dominate the combinatorial problem, we consider the multi-rev case first as it is our primary motivation to develop an efficient LUF approximation.

Multiple revolution Approximation

While the diagram in Fig. 2 appears as if a quadratic or cubic may suffice as a first order approximation, in practice we find that these functions are not well suited across all geometries. Instead, we find that a $log_e()$ transformation of the function is well behaved when we separate the function into what the classic formulation calls the long and short period branches.[24,31] We note that z is inversely related to the period and therefore the right and left branches are the short and long period solutions respectively. Unlike the zero-rev case (to be discussed in the next section), the log transformed LUF for the multi-rev transfer is not well approximated as linear. Instead we resort to a quadratic fit (see Equation 10)

$$\log(TOF) = az^2 + bz + c \tag{10}$$

where a, b and c are the polynomial coefficients.

To obtain the polynomial coefficients we make use of the already computed (in phase 1) minimum time of flight (t_{bi}). We also use the fact that $\frac{dTOF}{dz}$ is zero at this point. Hence we only require one additional point sufficiently close to the lower and upper limit of z for the short period or long period transfer respectively. We could also choose to include the value of the $\frac{d^2(TOF)}{dt^2}$ to achieve a cubic fit or eliminate the extra point. For simplicity and overhead reductions, we currently do not use the second derivative value.

Equations 11 - 13 summarize the expressions for obtaining the quadratic coefficients. Separate coefficients are necessary for both the long period and short period transfers. Figure 3 illustrates a diagram of the necessary points, where t_{bot} is equal to t_{bi} for a ith revolution transfer and z_{bot} is the corresponding z value. Given a value of n and direction of motion, t_{aux} represents the TOF for a heuristically chosen auxiliary point (z_{aux}, see Eq 14) close to the bounds of z on either side of z_{bot}. The suggested values in Eq 14 are based on an averaged performance over a wide range of input geometries.

$$a = \frac{\log(t_{aux}) - \log(t_{bot})}{z_{aux} - z_{bot}} \tag{11}$$

$$b = -2az_{bot} \tag{12}$$

$$c = \log(t_{bot}) + az_{bot}^2 \tag{13}$$

$$z_{aux} = \begin{cases} 0.3z_{bot} + 0.74n^2\pi^2, & \text{long period} \\ 0.3z_{bot} + 0.74(n+1)^2\pi^2, & \text{short period} \end{cases} \tag{14}$$

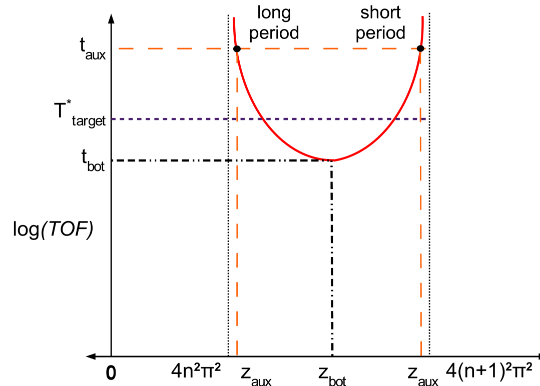


Figure 3. Multi-rev LUF approximation: necessary points

Once we obtain the polynomial coefficients the formula is inverted to achieve the required form $z(T^*)$. We emphasize that the quadratic approximation is obtained using only one additional LUF call.

Zero revolution Approximation

Figure 4 shows the $log_e()$ transformed TOF for the zero-rev transfer case. We observe that the $\log(TOF)$ function behaves linearly for much of the domain in z. We start the approximation by dividing the whole LUF region into three smaller regions (see Fig. 4) given in Table 1.

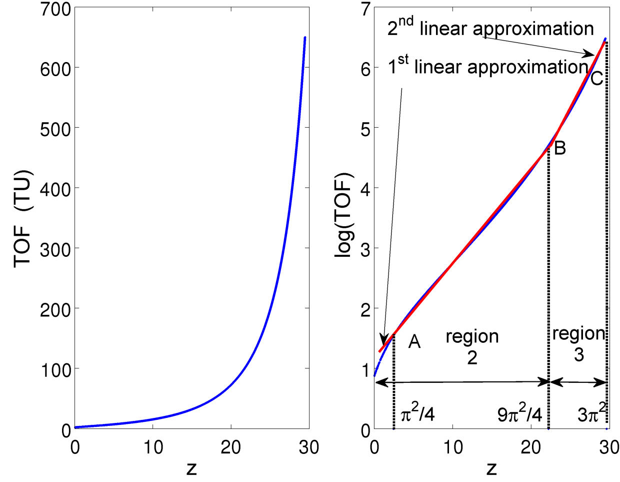


Figure 4. Effect of log transformation: 0 revolution case

Table 1. LUF regions: 0 revolution case

Region 1	$z < 0$	Hyerbolic region approximated by 0 initial guess
Region 2	$0 < z < B$	1st Elliptical region approximated by linear interpolation
Region 3	$B < z < 4 * \pi^2$	2nd Elliptical region approximated by linear interpolation

For regions 2 and 3, three transition points A, B and C are identified. These points mark the approximate boundary at which the behavior of the $\log(TOF)$ diverts from near-linear (Fig. 4). This can be attributed to the behavior of the geometry independent functions $k1$ and $k2$. The auxiliary function $k1$ behaves almost linearly between points A and B while $k2$ starts growing exponentially after point C. Hence, as $k1$ and $k2$ are only functions of z, we can use their precomputed values for any transfer geometry. This considerably reduces the computational effort of the extra function evaluation since $C(z)$ and $S(z)$ are expensive to compute and $k1$ and $k2$ require sqrts. Values of z at the transition points along with $k1$ and $k2$ are given in Table 2.

Table 2. Transition points summary

Transition Point	z	$k1$	$k2$
A	$\pi^2/4$	1	5.712388980384687E+0
B	$9*\pi^2/4$	-1	5.707963267948968E-1
C	$3*\pi^2$	-1.290786939662207E+0	3.207245260435367E+1

For the hyperbolic transfer case an initial guess of $z = 0$ is adopted, noting that this case is interesting for many applications, but peripheral to our main interest of tour design. The patched linear interpolation approximation provides reliable initial guesses for z except in cases very near the limits (0 and $4\pi^2$). Care should be taken to ensure that the initial guess of z is always within these bounds.

Performance

The improved initial guess leads to a significant reduction in the number iterations required over a static guess (generally the mid-point over the valid transfer range of z). A monte-carlo run with $\vec{r_1}$ and $\vec{r_2}$ uniformly varying between $0 \ldots 2$ LU, and TOF varying between 0 - 100 TU, is performed. The range of TOF is selected to allow for an excessively high number of revolutions. To avoid the singular case of $\theta = 2n\pi$, we ignore the geometry inputs when the magnitude of the transfer angle is less than $1E-2$ degrees. The LUF is not singular for the $(2n+1)\pi$ case as long as the f and g series are not used to obtain $\vec{v_1}$ and $\vec{v_2}$. Both cases $2n\pi$ and $2(n+1)\pi$ result from the introduction of degrees of freedom to the solution and are special cases that should be handled separately in practice.

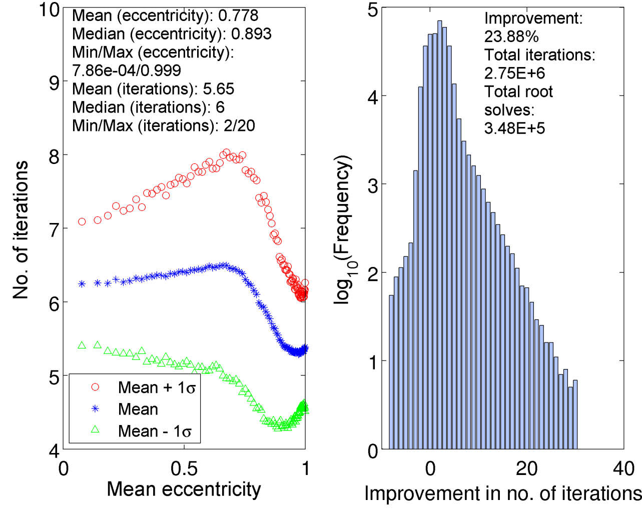


Figure 5. LUF approximation performance

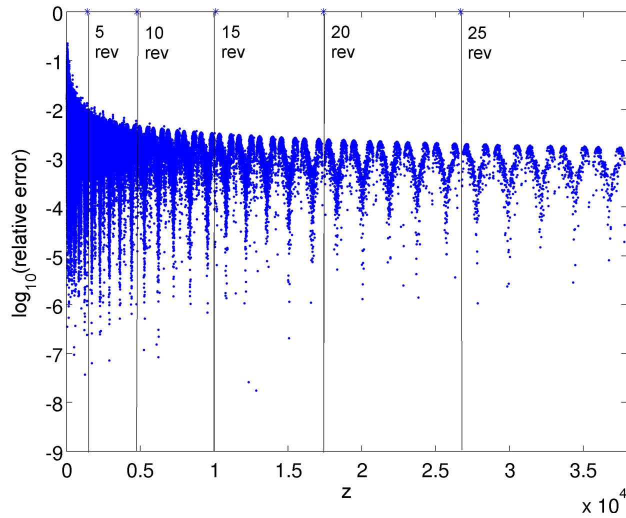


Figure 6. Relative error in z for LUF approximation

Figure 5 shows the improvement in number of iterations achieved corresponding to each successful root solve of the LUF. We achieve a significant (23.88%) reduction in number of iterations which results in approximately 12.5% reduction in computation time. In the "$Iterations$ vs. $Mean\ Eccentricity$" each set

of markers represents the mean eccentricity and 1 sigma variations for 5569 neighboring solutions ordered on the basis of eccentricity. We deliberately test our LUF approximation against highly eccentric transfers. These solutions are harder to converge and generally take more iterations to reach the desired numerical accuracy. Remarkably, the LUF approximation leads to fewer iterations, on average, at the very high eccentricity values. Note that the hyperbolic solutions are not in this figure as the LUF approximation is only valid for ellipses.

Figure 6 shows the relative error of the approximation in z for all possible transfers (up to 30 revolutions in this simulation). The largest relative error in z is less than 0.18, while the average error over all solutions is $< 1E-2$. Hence, the LUF approximation algorithm works well over a large range of TOF and transfer geometry. This robustness of the LUF approximation is important for the safe implementation of parallel Lambert computations on the GPU.

Before highlighting our GPU implementation of the multi-rev Lambert algorithm, some details about the NVIDIA GPU architecture and CUDA technology are presented next.

NVIDIA GPU ARCHITECTURE AND CUDA

Recent advances in programmable GPU has lead to the development of small, yet highly parallel and multi-threaded processors with many cores. Given the GPU high computational throughput and its ability to tap fine grain parallelism, researchers are now mapping non-graphical applications to this hardware with a wide range of success.[32] This field is generally called GPGPU (General Purpose Computing on GPU) programming. With the development of the Nvidia TESLA architecture (in late 2006[33]) along with the recent introduction of Nvidia CUDA * technology, there has been tremendous growth in wide scale GPGPU programming applications on the TESLA architecture. Most of these applications have witnessed a performance boost of 5 to 500 times, thereby outperforming many mid-range supercomputers with just one graphics processor. The latest TESLA G200 architecture (the C1060 series) consists of 240 cores, 4 GB of device memory, and is capable of full IEEE compliant floating point arithmetic.

The CUDA computing architecture is a C-like programing language with keywords for labeling data-parallel functions (kernels), and their associated data structures. Kernels generally execute a large number of threads (on the order of tens of thousands) in parallel. A thread is basically a fork which results from concurrent execution of computation on the GPU. Typically, thousands of threads perform the same set of computation instructions with some limited cooperation over a different set of data.

The NVIDIA C Compiler (NVCC) is responsible for compiling the CUDA code. The part of the code which runs on the GPU is called the device code and the part of the code that runs on the CPU is called the host code. The host and device codes can be compiled using different compilers and linked at runtime. For our implementation, we compile the host code with the Intel Fortran compiler and link it with the device code compiled with NVCC.

There are many algorithmic challenges faced by the programmer (especially due to the memory constrained GPU hardware) while developing algorithms which map effectively to the GPU. Often non-intuitive techniques are developed to map conventionally serial algorithms to the GPU †. Very high performance boosts are possible only if the algorithm maps efficiently to the GPU hardware and a sufficient amount of optimization has been performed. Hence algorithm development is the major activity for consideration when programing in CUDA.

GPU ACCELERATED MULTI-REV LAMBERT SOLVER: CULAM

We implement the universal variable multi-rev Lambert algorithm enhanced with the LUF approximation technique on the GPU. Our implementation (CULAM) also utilizes precision control techniques to achieve consistent single precision accuracy for a wide range of TOF values. The basic structure of CULAM can be divided into three parts as follows:

*http://www.nvidia.com/object/cuda_what_is.html - cited Feb 25 2010

†http://developer.download.nvidia.com/compute/cuda/sdk/website/projects/scan/doc/scan.pdf - cited Feb 25 2010

1. Dynamic grid allocation and inital data transfer to the GPU
2. Multi-rev Lambert kernel implementation
3. Transfer of solution data from the GPU to the CPU

The multi-rev Lambert implementation on the GPU is similar to its CPU counterpart. The zero-rev implementations on the GPU and CPU are essentially the same. During the minimization phase we require a second derivative to minimize the LUF. As shown in Eq 9, the derivative requires multiple transcendental function evaluations. The GPU contains fast hardware functions but they are less accurate and not always rapidly available. Too many transcendental evaluations per thread can lead to loss of precision and higher per thread execution time. Hence, for the minimization phase, we alleviate this problem by using a precision safe version of the secant method.

CULAM starts by evaluating the number of threads and the amount of memory required for allocation on the GPU. We introduce the parameter N^* which is an upper limit on the number of revolutions we seek for any given transfer geometry, direction and TOF value. This N^* value is different from $nmax$, the maximum possible number of revolutions. Currently, data volume must be pre-allocated to pass information to and from the GPU. Therefore, N^* which is a direct measure of the GPU data traffic, plays a major role in the performance of CULAM in terms of memory copy time. The input data for CULAM consists of $\vec{r_1}$, $\vec{r_2}$, TOF and direction of motion values arranged in an array of elementary data types known as a "Structure of Arrays (SOA)". In GPU programming, SOAs typically lead to improved access to the global memory.

During the kernel execution each thread is responsible for loading a complete set of inputs, calculating all possible transfers and storing these solutions back in the GPU main memory in an aligned manner. Finally, once the GPU finishes computing all possible transfers, the solutions ($\vec{v_1}$, $\vec{v_2}$) are copied back to the host CPU code for further processing.

The GPU works like a vector processor capable of handling a large number of threads in multiple groups of 16. It is essential that 16 consecutive threads perform approximately the same volume of computation to achieve peak performance on the GPU. This may be seen as a problem for CULAM, as at any given time each thread may have a different set of input data which may lead to different numbers of Newton-Raphson iterations. Fortunately, this problem is alleviated by our LUF approximation as discussed in earlier sections. The approximation not only reduces the number of iterations but also leads to approximately the same number of iterations for a varying range of input data.

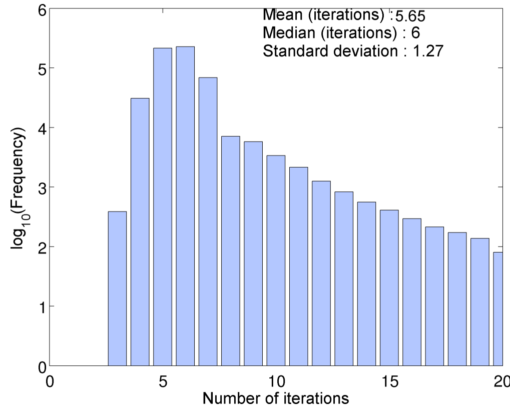


Figure 7. LUF approximation performance

To investigate the workload distribution, the iteration results from the previous monte carlo are plotted in Fig. 7. We can see that the average number of iterations required using the inital guess algorithm is 5.65

with a standard deviation of 1.27. Further, from manual inspection we find that over 85% of the threads take approximately the same number of iterations in groups of 16. Hence the LUF approximation acts as a workload balancer thereby increasing the runtime performance of CULAM.

A detailed performance profile using NVIDIA's profiler shows that CULAM is computationally "balanced", meaning that given enough threads the GPU-CPU memory transfers take approximately the same amount of time as that taken by GPU Lambert kernel computations.

DIRECT LAMBERT CALL COMPARISON: CULAM VS. CPU

In this section we evaluate the performance of CULAM against our tuned CPU implementation. Table 3 gives the specifications of the test hardware. We note that only one GPU at a time is used for the simulations, yet we benchmark the performance of two models for comparison purposes.

The TESLA architecture based GTX 285 GPU is the fastest NVIDIA card available and currently costs around $400. The same card but made specifically for parallel computing using higher quality components and more on chip memory is the TESLA C1060 which costs around $1600. We use the TESLA C1060 as one of the GPUs for this comparison. Realizing that most common laptops and desktops may not have a high-end GPU, we also evaluate the performance of CULAM on the most basic compute GPU (the QUADRO series). They are approximately 25 times slower compared to the TESLA C0160 GPU. Most new laptops have a GPU that is better than the QUADRO, hence our performance comparison charts should approximate the current upper and lower limits of speedup achievable using CULAM.

Table 3. Test hardware specifications

Component type	Component
CPU	Intel Core 2 Duo E6550 @ 2.33 Ghz
Operating system	Linux X86_64
GPU 1	TESLA C1060
GPU 2	QUADRO NVS-295
RAM (Memory)	4.0 GB

The CPU code is compiled with the Intel Fortran compiler version 11.1 with optimization level set to "*-fast*". This enables auto vectorization and inter-procedural optimization. These optimizations result in a 2 times improvement in performance over the unoptimized CPU code. Apart from compiler optimization the CPU code is tuned for high performance Fortran 2003. The CUDA code is compiled using the NVCC compiler version 3.0. Numerical accuracy on the order of approximately $< 1E-7$ is maintained throughout all computations both on the CPU and GPU. We emphasize that double precision is currently available on the TESLA card (not QUADRO) but is ~ 8 times slower that of single precision. Fortunately, single precision is sufficient for most Lambert applications, especially in the global search phase of design.

We perform two kinds of Lambert call comparisons. One based on random geometry and TOF and the other based on random TOF but a specific repeating geometric pattern, like transfers between two planets.

Random Geometry

In this comparison, vectors $\vec{r_1}$, $\vec{r_2}$ vary between 0 - 2 LU, while TOF varies between 0 - 100 TU, unless otherwise specified. Random values are uniformly distributed between the bounds. The value of μ is normalized to unity and N^* is fixed at 5, that is we do not search for any solutions even if they exist above $n = 5$. The timing on the GPU code includes the time taken to transfer memory back and forth between the host and device.

Figure 8 gives the performance for random Lambert calls to CULAM varying in numbers from 16 to approximately 2 million at a time. The amount of Lambert calls which can be concurrently launched on the

GPU is limited by device memory allocated before the kernel call. For the TESLA C1060 we are able to achieve speedups of up to 250 times over the optimized CPU implementation. Even for a modest 30,720 Lambert calls, we achieve a speedup of more than 200 times. The second chart in Fig. 8 shows the speedups using a low end QUADRO series GPU. CULAM still achieves an order of magnitude of speedup over the CPU implementation. We emphasize that many if not most modern laptops are able to match or outperform the QUADRO.

The third chart gives the execution time vs. number of Lambert calls. The point at which the execution times for the CPU and GPU cross is called the "break-even point". Hence, for TESLA C1060 we need just 16 Lambert calls while on QUADRO we need 64 Lambert calls to begin outperforming the CPU implementation. We emphasize that most mission design Lambert applications require thousands of solutions or more.

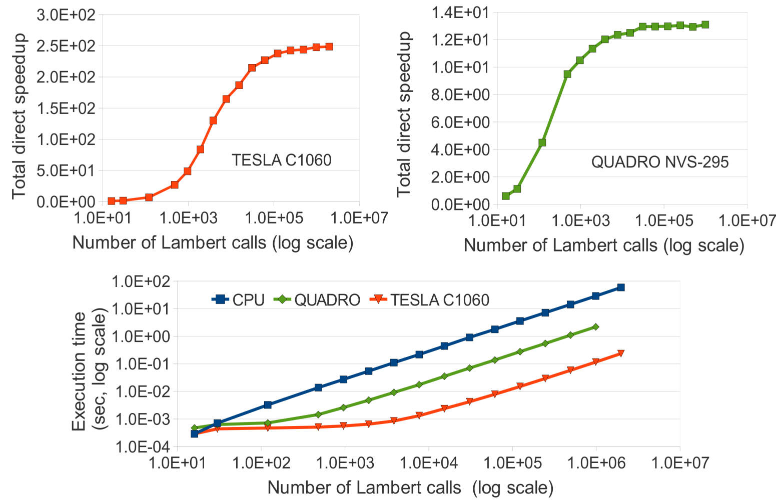


Figure 8. Random CULAM performance

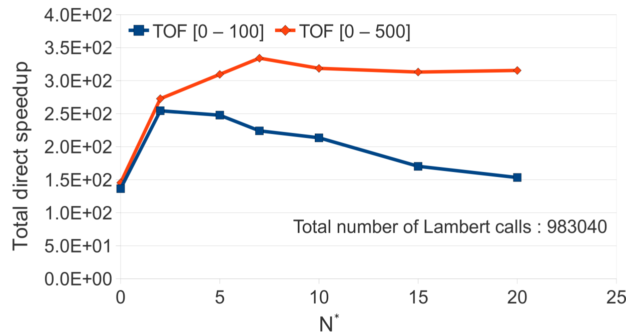


Figure 9. Random CULAM performance (varying N^*)

To examine speedup behavior of CULAM with varying number of s/c revolutions we vary N^* between 0 to 20 and plot the result for two ranges of TOF in Fig. 9. When the TOF varies between 0 to 100 TU, we observe a maximum speedup of 247 times at $N^* = 2$. The speedup decreases as we increase the value of N^* because the amount of computation relative to the amount of memory transfer decreases. While the CPU has no memory transfer overhead, the GPU code has significant overhead for increasing values of N^*. This behavior indicates the transition of CULAM from being computationally balanced to memory bound.

On the other hand when TOF varies between 0 to 500 TU we are able to achieve a maximum speedup of 335 times for $N^* = 7$. Further as we increase N^* up to 20 we are able maintain a speedup of over 300 times.

This is the case because CULAM continues to remain computationally balanced due to the large TOF range that accommodates more high revolution transfers, hence more computations

Table 4 gives the raw computational throughput of CULAM for a N^* value of 5 and TOF range of 0 to 100 TU. CULAM achieves a throughput of 8.5 million Lambert calls per second. This number will be even higher for a larger value of N^* with a wider TOF range.

Table 4. Computational throughput (N^* = 5)

Compute card	$\frac{Lambert\ solutions}{second}$
Intel Core 2 Duo	3.4E+4
TESLA C1060	8.5E+6
QUADRO NVS-295	4.5E+5

Repeating Geometry

For the second direct comparison we use initial inputs in the form of position vectors $\vec{r_1}$, $\vec{r_2}$ obtained from a planetary ephemeris. The states of the planets are obtained using the SPICE ephemeris system * and the states are retrieved using a fast interpolation tool called FIRE.[34] The launch and arrival bodies are Earth and Venus respectively, and the flight times vary between 1 and 50 years. Normalized units are used with a value of μ equal to unity. Instead of fixing a maximum value of revolutions, we seek all possible solutions. For this we estimate the value of N^* so as to capture all of the solutions yet minimize the required memory transfer.

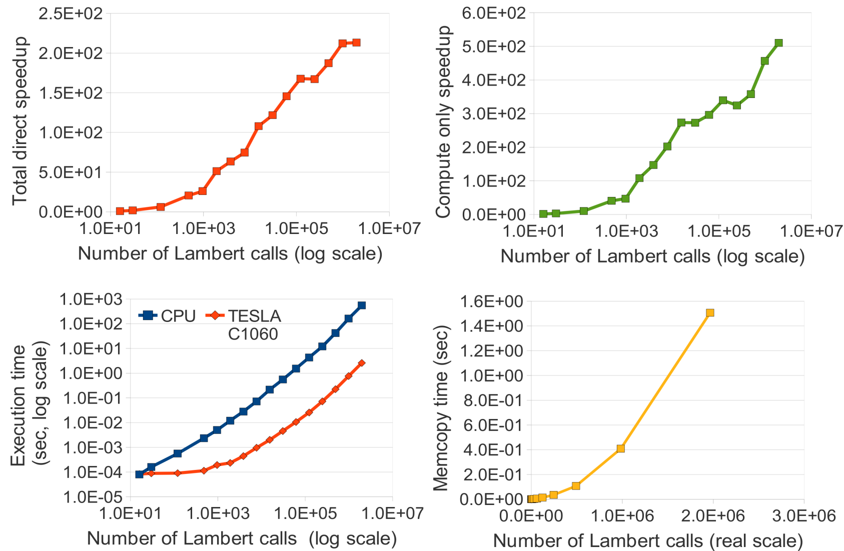


Figure 10. Pattern based CULAM performance

For benchmarking CULAM we use two timing counters. The first timing counter measures the "total direct speedup" over the CPU implementation and includes the time taken to transfer memory back and forth between the host and the device. The second timing counter measures the "compute only speedup" where we only take into account the time taken for CULAM to compute all solutions. This latter time is the upper bound for speedup in the case where memory transfers are either absent or overlapped with necessary CPU calculations. The compute only speedup is relevant when we apply CULAM to a problem like a grand tour

*http://hdl.handle.net/2014/11748 - cited Feb 25 2010

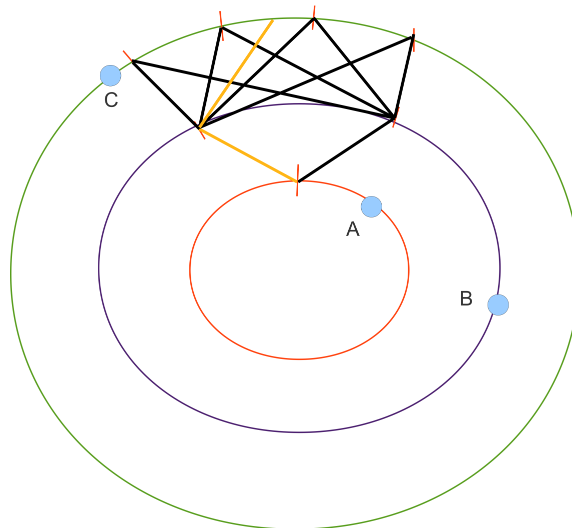
mission design, where there exists a potential to hide the host to device memory transfers by overlapping the CPU and GPU computation.

Figure 10 gives the performance of CULAM for number of Lambert calls varying from 16 to approximately 2 million. CULAM on the TESLA C1060 is able to reach an total direct speedup value of 220 times for 1 million Lambert calls. For 65,000 Lambert calls we achieve a speedup of more than 150 times. The second sub-chart gives us the direct compute speedup. We achieve an impressive speedup of 500 times for approximately 2 million calls. The number of simulations were limited by the memory limits of the CPU code structure. The third sub-chart helps us to locate the "break-even" point which occurs again at 16 Lambert calls. Finally, we observe that the memory copy time increases superlinearly with increasing number of Lambert calls. This superlinear increase is typical for the GPU - CPU PCIe-2.0 hardware connection as data transfer speed stagnates after a certain amount data transfer size is reached.

Out of the 2 million Lambert calls only 109 were found to be partially converged ($\delta z > 1E-6$ but still close to the solution). The primary reason for failure is a maximum iterations limit that is arbitrarily set to 25. We note that 100% of the solutions converged to a tolerance $< 3e-4$. These tests demonstrate the robustness of CULAM which is essential for problems having a large design space. The improved performance over the randomized geometry case is primarily because the $|\vec{r_1}|$ and $|\vec{r_2}|$ are more consistently the same order of magnitude, leading to better behaved numerics. Further, the neighboring calls to CULAM are more closely related, which provides iteration consistency across neighboring thread computations. Fortunately, most mission design applications are similar in nature to this latter more favorable example.

FLYBY TRAJECTORY PROBLEM EXAMPLE

In this section we examine the performance of CULAM applied to a two leg, flyby trajectory design problem. To consider a large variation in distance and in time scales, we consider trajectories in two different systems: an interplanetary trajectory from Earth to Mercury via a flyby from Venus; and a Uranus moon trajectory from Oberon to Ariel via a flyby at Titania. Before showing the results we give a brief overview of our parallelization strategy. We emphasize that the purpose of this section is to show that CULAM is an appropriate workhorse of the "inner loop" in this common mission design problem. The "outer loop" is not a main focus here although we do consider it ripe for future work.



Each branch represents on complete Lambert solution

The transfer solution is shown in yellow

Figure 11. Design Space

Consider a simple three orbit transfer problem as shown in Fig. 11 with three bodies A, B and C. The objective is to find all ballistic transfers from body A to body C with an intermediate, above surface flyby at body B. To solve this problem, the orbits of all bodies are discretized (nodes in Fig. 11) and all possible combinations (edges in Fig. 11) are evaluated for feasible transfers. Each evaluation requires one complete multi-rev Lambert call ($2nmax$+1 transfers for each direction of motion).

There is abundant parallelism in the above problem as all the Lambert problems are independent once the time discretization is complete. Further there is a second level of parallelism present where solutions corresponding to different directions of motion and number of revolutions can be computed in parallel. After all possible transfers for both the legs are computed, any potential solutions are patched on the CPU using a v_∞-matching procedure and constraint checks on flyby altitudes. In our current implementation, only the Lambert calls are calculated with the aid of the GPU.

Interplanetary flyby trajectories

An Earth to Mercury with a single flyby at Venus (EVM) transfer trajectory is considered. The mission design parameters are given in table 5. The resolution of both Earth and Venus is set at 32 points per revolution while Mercury's is set to 16. Table 6 summarizes the results.

Table 5. Interplanetary trajectory design parameters

Parameters	Value
Transfer type	Earth-Venus-Mercury
v_∞ launch range	0-3.5 (km/sec)
Minimum launch date	JAN 1 2015
Maximum launch date	JAN 1 2020
Maximum Arrival date	JAN 1 2025

Table 6. Interplanetary trajectory design solution

Parameter	Value
Number of Lambert calls	2.09868E+5
Number of ballistic transfers with flyby altitude $>$ 200 km	2.92E+2
Number of failed transfers	0
Ephemeris lookup time	5.0E-4 seconds
CPU Lambert calls	7.4E+0 seconds
CULAM Lambert calls (TESLA)	1.0E-1 seconds
CULAM Lambert calls (QUADRO)	5.2E-1 seconds
v_∞ root solving time	3.2E-1 seconds
Lambert speedup (TESLA)	74 times
Lambert speedup (QUADRO)	14.2 times
v_∞root solving time / Lambert time (TESLA)	3.2
v_∞root solving time / Lambert time (CPU)	.04

CULAM is able to compute all Lambert calls (209,868) in one tenth of a second. The v_∞ root solve takes more than three times the amount of time taken by the GPU.

Uranus moon flyby trajectories

The tour design problem at planetary satellite systems is much more computationally difficult due the short periods of the bodies. A common planetary system mission might include on the order of 100 revolutions around the primary. Therefore, we choose a planetary satellite example that highlights this computational burden. As a moon tour example we consider Uranus system transfers from Oberon to Ariel with a single flyby at Titania. Table 7 gives the mission design parameters adopted for the search. The resolutions of Oberon, Titania, and Ariel are set to 64, 64, and 32 points per revolution respectively. Table 8 summarizes the results.

Table 7. Uranus intermoon trajectory design parameters

Parameters	Value
Transfer type	Oberon-Titania-Ariel
v_∞ launch range	5.9-6.0 (km/sec)
Minimum launch date	JAN 1 2020
Maximum launch date	FEB 1 2020
Maximum arrival date	MAR 1 2020

Table 8. Uranus intermoon trajectory design solution

Parameter	Value
Number of Lambert calls	5.06116E+5
Number of ballistic transfers with flyby altitude $>$ 20 km	2.1798E+4
Number of failed transfers	0
Ephemeris lookup time	9.0E-4 seconds
CPU Lambert calls	2.457E+1 seconds
CULAM Lambert calls (TESLA)	3.30E-1 seconds
CULAM Lambert calls (QUADRO)	1.27E+0 seconds
v_∞ root solving time	1.396E+1 seconds
Lambert speedup (TESLA)	74.5 times
Lambert speedup (QUADRO)	19.4 times
v_∞ root solving time / Lambert time (TESLA)	42.30
v_∞ root solving time / Lambert time (CPU)	1.76

The problem is solved in just under 15 seconds as we are able to compute approximately ~ 0.5 million Lambert calls in 0.33 seconds. After pruning this solution set we are left with 21,798 resulting feasible transfers with matching v_∞ and non-impacting flybys. As we can see from the results, the Lambert calls using CULAM no longer dominate the execution time. Infact for this problem CULAM accounts for less than 2.4% of the computation time meaning that the v_∞ root solving phase is ~ 42 times slower than the calls to CULAM. This result provides an impetus to build a v_∞ root solving algorithm on the GPU.

CONCLUSIONS

In this paper, after a short review of the history and state of the art of the celebrated Lambert problem, we propose CULAM as a practical and fast solution method for tackling large scale combinatorial Lambert problems. CULAM is a GPU accelerated, universal variable-based, multi-rev Lambert algorithm. It includes a simple and efficient initial guess generation technique to improve robustness and reduce interations.

Speedups of up to 250 times are reported over an optimized CPU implementation. To demonstrate the practical use of CULAM we implement examples in the interplanetary and intermoon design space. The capability of CULAM to rapidly compute transfers can be harnessed in a transparent and readily scalable manner using a wide range of existing computational resources.

As GPUs are a common part of modern desktop computers, a broad community stands to benefit from CULAM and the associated GPU computing philosophy. The ability to provide such dramatic performance improvements with a single computer provides for unprecedented mission design capabilities that were previously reserved for supercomputers. The design times for complex searches can be potentially reduced from the currently impractical days or weeks to minutes or seconds, greatly improving the mission design process.

ACKNOWLEDGEMENTS

The authors thank Yanping Guo, Jim McAdams, Damon Landau, Nathan Strange, and Anastassios Petropoulos for their interest and useful inputs. The authors also thank NVIDIA for technical and hardware support.

NOTATION

z	Universal variable
n	Revolution number of the transfer
i	ith revolution transfer under consideration
M	Mass of the body
G	Standard gravitational parameter
μ	$G \times M$ of the primary body
δz	z converged - z approximated
v_{∞}	Hyperbolic excess velocity
s/c	Space craft
T^{*}	Target time of flight
N^{*}	Maximum allowed number of s/c revolutions
$multi-rev$	Multiple revolution
TOF	Time of flight
t_{bi}	Minimum TOF for any revolution i
t_{bot}	Minimum TOF corresponding to the current revolution transfer
t_{aux}	TOF for the auxiliary point (applicable to multi-rev transfers only))
z_{bot}	Universal variable corresponding to the current revolution transfer
$\vec{r_1}$	Starting body position vector
$\vec{r_2}$	Ending body position vector
$\vec{v_1}$	Starting body velocity vector
$\vec{v_2}$	Ending body velocity vector
θ	Angle between the two position vectors
$C(z)$ & $S(z)$	Stumpff functions
$C(z)'$ & $S(z)'$	First derivative of $C(z)$ & $S(z)$
$C(z)''$ & $S(z)''$	Second derivative of $C(z)$ & $S(z)$
$nmax^{D}$	Maximum number of revolutions possible (direct)
$nmax^{r}$	Maximum number of revolutions possible (retrograde)
$nmax$	Total number of revolution transfers for a Lambert call
CPU	Central Processing Unit
GPU	Graphics Processing Unit
$CUDA$	Compute Unified Device Architecture
$NVCC$	NVIDIA CUDA compiler
$CULAM$	CUDA accelerated multi-rev Lambert solver
$GPGPU$	General purpose computing on graphics processing units

REFERENCES

[1] E. R. Lancaster, R. C. Blanchard, and R. A. Devaney, "A Note on Lambert's Theorem," *Journal of Spacecraft and Rockets*, Vol. 3, No. 9, 1995, pp. 1436–1438.

[2] R. H. Gooding, "A procedure for the solution of Lambert's orbital boundary-value problem," *Celestial Mechanics and Dynamical Astronomy*, Vol. 48, No. 2, 1990, pp. 145–165.

[3] R. H. Battin, "An Introduction to the Mathemat-ics and Methods of Astrodynamics," *AIAA Education Series*, 1999.

[4] D. Vallado and D. W. McClain, *Fundamentals of astrodynamics and applications*. 2001.

[5] P. R. Escobal, *Methods of Orbit Determination*. John Wiley & Sons, 1965.

[6] S. I. Ochoa and J. E. Prussing, "Multiple Revolution Solutions to Lambert's Problem," *Advances in the Astronautical Sciences*, Vol. 79, No. 2, 1992, pp. 1989–2008.

[7] V. L. Coverstone and J. E. Prussing, "A Class of Optimal Two-Impulse Rendezvous Using Multiple-Revolution Lambert Solutions," *The Journal of the Astronautical Sciences*, Vol. 48, Apr. 2000, pp. 131–148.

[8] G. Avanzini, "A Simple Lambert Algorithm," *Journal of guidance, control, and dynamics*, Vol. 31, No. 6, 2008, pp. 1587–1594.

[9] Q. He, J. Li, and C. Han, "Multiple-Revolution Solutions of the Transverse-Eccentricity-Based Lambert Problem," *Journal of Guidance, Control and Dynamics*, Vol. 33, Jan. 2010.

[10] S. Herrick and A. Liu, "Two Body Orbit Determination from Two Positions and Time of Flight," *Appendix A, Aeronutronic*, Vol. C-365, 1959.

[11] C. F. Gauss, *Theory of Motion of the Heavenly Bodies Revolving about the Sun in Conic Sections*, Vol. 1. Dover, 1963.

[12] J. D. Thorne, "Lambert's Theorem - A Complete Series Solution," *Advances in the Astronautical Sciences*, Vol. 119, No. 3, 2004, pp. 3061–3074.

[13] D. J. Jezewski, "K/S two-point-boundary-value problems," *Celestial Mechanics and Dynamical Astronomy*, Vol. 14, No. 1, 1976, pp. 105–111.

[14] E. R. Lancaster and R. C. Blanchard, "A unified form of Lambert's theorem," tech. rep., NASA, Sept. 1969.

[15] R. R. Bate, D. D. Mueller, and J. E. White, *Fundamentals of astrodynamics*. Dover, 1971.

[16] J. Kriz, "A uniform solution of the Lambert problem," *Celestial Mechanics and Dynamical Astronomy*, Vol. 14, No. 4, 1976, pp. 509–513.

[17] R. H. Battin, "Lambert's problem revisited," *AIAA Journal*, Vol. 15, No. 5, 1977, pp. 707–713.

[18] C. Uphoff, P. H. Roberts, and L. D. Friedman, "Orbit Design Concepts for Jupiter Orbiter Missions," *Journal of Spacecraft and Rockets*, Vol. 13, No. 6, 1976, pp. 348–355.

[19] J. M. Longuski and S. N. Williams, "Automated design of gravity-assist trajectories to Mars and the outer planets," *Celestial Mechanics and Dynamical Astronomy*, Vol. 52, 1991, pp. 207–220.

[20] R. P. Russell and C. A. Ocampo, "Global Search for Idealized Free-Return Earth-Mars Cyclers," *Journal of Guidance, Control, and Dynamics*, Vol. 28, No. 2, 2005, pp. 194–208.

[21] A. V. Labunsky, O. V. Papkov, and K. G. Sukhanov, *Multiple gravity assist interplanetary trajectories*. ESI book series, 1998.

[22] A. F. Heaton, N. J. Strange, J. M. Longuski, and E. P. Bonfiglio, "Automated Design of the Europa Orbiter Tour," *Journal of Spacecraft and Rockets*, Vol. 39, No. 1, 2002, pp. 17–22.

[23] O. Abdelkhalik and D. Mortari, "N-Impulse Orbit Transfer Using Genetic Algorithms," *Journal of Spacecraft and Rockets*, Vol. 44, 2007.

[24] R. P. Russell and C. A. Ocampo, "Geometric Analysis of Free-Return Trajectories Following a Gravity-Assisted Flyby," *Journal of Spacecraft and Rockets*, Vol. 42, No. 1, 2005, pp. 138–151.

[25] B. G. Marsden, "Initial orbit determination - The pragmatist's point of view," *Astronomical Journal*, Vol. 90, 1985, pp. 1541–1547.

[26] K. B. Beley, "Analytic Determination of Perigee Passage Using Lambert's Therorem," *Journal of Spacecraft and Rockets*, Vol. 4, No. 8, 1967, pp. 1101–1103.

[27] A. E. Petropoulos, J. M. Longuski, and E. P. Bonfiglio, "Trajectories to Jupiter via Gravity Assists from Venus, Earth and Mars," *Journal of Spacecraft and Rockets*, Vol. 37, Nov. 2000.

[28] N. Arora, R. P. Russell, and R. W. Vuduc, "Fast Sensitivity Computations for Trajectory Optimization," *AAS/AIAA Astrodynamics Specialist Conference and Exhibit*, 2009.

[29] NVIDIA, "NVIDIA CUDA Programming Guide 2.0," *Documentation*, 2008.

[30] L. Nyland, M. Harris, and J. Prins, *Fast N-Body Simulation with CUDA*.

[31] J. E. Prussing and B. A. Conway, *Orbital mechanics*. Oxford University Press, 1993.

[32] N. Arora, A. Shringarpure, and V. R.W, "Direct N-body Kernels for Multicore Platforms," *International conference on parallel processing*, 2009.

[33] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, "NVIDIA Tesla: A Unified Graphics and Computing Architecture," *Micro IEEE*, 2008.

[34] R. P. Russell and N. Arora, "FIRE: A Fast, Accurate, and Smooth Planetary Body Ephemeris Interpolation System," *AAS/AIAA Astrodynamics Specialist Conference and Exhibit*, 2008.