Massively parallel solution of the Lambert problem

V S Kravchenko\textsuperscript{1,2,*} and A V Ivanyukhin\textsuperscript{1,2}

\textsuperscript{1}Research Institute of Applied Mechanics and Electrodynamics, Moscow Aviation Institute (National Research University), 5 Leningradskoye shosse, 125080, Moscow, Russian Federation

\textsuperscript{2}Academy of Engineering, Peoples Friendship University of Russia (RUDN University), 6 Miklukho-Maklaya street, 117198, Moscow, Russian Federation

*E-mail: kravchenko_vser@pfur.ru

Abstract. This paper considers the possibility of massively parallel solution to the Lambert problem on graphics processing units. Several of the most popular solution algorithms were used for this problem. Software implementation of algorithms under consideration has been developed using Compute Unified Device Architecture technology that allows performing calculations on graphics processing units. A distinctive feature of this project is an attempt to use parallel programming within an iterative scheme. The developed algorithms have been tested on a number of typical tasks: contouring of Earth – Mars direct flight isolines and solution of the problem of a flight to a group of asteroids for the given start date and flight duration. For such tasks we give estimates of execution time and paralleling efficiency.

1. Introduction

The Lambert problem is a classical celestial mechanics problem. It can be stated as follows: find a Keplerian orbit passing through two points of space with radius-vectors \( r_1 \) and \( r_2 \) at time moments \( t_0 \) and \( t_1 \), i.e. as a boundary value problem for the restricted two-body problem.

Initially, the purpose of solving this problem was to determine the orbits of the solar system bodies from two observations. However, its statement can be considered as an elementary version of the spacecraft flight problem. And now in astrodynamics this problem is often used at the initial stage of mission analysis to determine suitable launch dates and flight duration. Besides, more complex methods of solving trajectory problems with \( n \) velocity impulses, multiple gravitational slingshots and visiting several objects are based on it [1-3]. The same task is often used to solve the most difficult tasks in the Global Trajectory Optimization Competition (GTOC) [6,7].

In practice, it is not necessary to solve the boundary-value problem (1) to solve the Lambert problem, since a complete set of first integrals for this problem is known. They allow you to reduce the solution to a single nonlinear equation. The form of this equation depends on the form of the Kepler equation for the orbital position-time relation.

\[
\dot{r} = \frac{\mu}{|r|^3} r, \quad \begin{cases} r(t_1) = r_1, \\ r(t_2) = r_2, \end{cases}
\]

where \( r \) is the radius-vector, \( \mu \) is the gravitational parameter of the central body.
Over the long history of the problem, several dozens of methods to solve it have been proposed. The human space flight became a powerful impulse for the development of methods, marking the beginning of the space age in the mid-20th century.

However, as the complexity and intensity of space missions grow, more and more different problems need to be solved efficiently. Modern approaches to the organization of performant computer-based systems give an opportunity to solve such problems by parallel methods, since most computationally time-consuming problems can be divided into a sufficiently large number of conditionally independent subproblems, which can be solved independently of each other and subsequently merged into a single solution. The Lambert problem can be an elementary unit of such a division.

Such approaches, with the advent of substantially multicore systems and computational co-processors, are being increasingly used in various fields of science, while requiring adaptation of the known methods to parallel architecture and development of new parallel solution methods.

At present, there are several dozens of methods for solving the Lambert problem, all of which differ in the type of nonlinear equation to be solved and the method of its solving. There are three main groups of equation types:

- methods based on the universal form of the Kepler equation (with a universal variable),
- methods based on the classical form of the Kepler equation and using iterations along one of the Keplerian elements (usually, it is a semi-major axis, eccentricity, focal parameter or anomaly),
- equations in Kustaanheimo-Stiefel regular variables (K-S variables).

All traditional methods for solving a nonlinear equation can be applied to above equations. This classification is shown in table 1.

**Table 1. Classification of methods for solving the Lambert problem.**

| The equation solution method | Halley iterations of the second order | Secant method | Newton's method | Sequential substitution algorithm |
|-----------------------------|--------------------------------------|---------------|----------------|---------------------------------|
| Methods based on a universal variable | Gooding [8] | Izzo [9] | Bate [10], Lancaster [11], Sukhanov [12] | Battin [22] |
| Methods based on the elements of the Keplerian orbit | | | Avanzini [13], Prussing [14], Chen [15], Boltz [16], Throne [17,18] |
| Methods based on K-S variables | | | Kriz [19], Simo [20], Jezewski [21] |

1.1. Universal variable

This formulation is very attractive because it is the same for all types of orbits including those that are rectilinear. For this reason, the variables in this formulation are commonly referred to as universal.

Lancaster [11], Bate [10] and Battin [23] were the first who proposed to solve the problem using universal variables. Then other authors were successively transforming their solutions. Gooding [8] and Izzo [9] proposed their own solutions as the extensions of Lancaster method [11]. Different schemes for solving the equations derived by Lancaster are used in their methods.

Stumpff devised a method to represent the solution of the two-body problem at any time $t$ from the position $(r_0)$ and velocity $(v_0)$ at some reference epoch $t_0$ [24]. Then, a unique solution of the generalized form of Kepler’s equation for any type of orbit can be written in terms of the motion constants and Stumpff’s functions $c_n$. The Sukhanov’s method [12] relying on the universal variables and Stumpff’s functions $c_n$ is nowadays one of the most efficient.
1.2. Keplerian orbital elements
Methods based on Keplerian orbital elements can be divided by three different approaches. The first approach is based on semi-latus rectum ($p$). Liu was the first who proposed such solution [15]. Liu’s follower was Boltz [16]. The next approach is based on semi-major axis ($a$). The first and the only one, who proposed the unique solution, was Lagrange. Further there were additions and modifications to the solution of derived equation by Throne [17,18], Prussing [14], Chen [15], and Waillez [24]. The last approach belonging to this group included solutions based on the eccentricity vector ($e$). Just as in the previous case, the only one who proposed a solution was Avanzini [13]. His followers were He [25], Zhang [26], and Wen [27].

1.3. K-S variables
This is the smallest group of methods. The first ones were Kriz [19] and Simo [20]. Kriz also had a follower – Jezewsky [21].

Thus, there are a large number of solution methods. Some of them have already been used by other authors to speed up the computations through parallel implementation. For example, N Russel and N Arrora [28], J Shimoun, etc. [29], and N. Parrish [30]. These authors based their solutions on the computational capacity of the GPU, which helps to accelerate the speed of solving the problem by several times. However, most of methods offer a massively parallel solution; in other words, computation of a number of problems in parallel. In this paper, we will consider several methods of paralleling a single task and examine the results of execution on CPU and GPU.

2. Methods and materials

2.1. Compute Unified Device Architecture on graphics processing units
The first GPUs were designed as graphics accelerators, culminating in the first Nvidia Corporation (NVIDIA) a graphics processing units (GPU) in 1999. Researchers and scientists quickly began to apply the superior performance of this GPU to general purpose computing. In 2003, a group of researchers led by Ian Buck introduced Brook, the first widely-used coding model, which extended C with data-parallel constructs. Later, in 2006, Ian Buck joined NVIDIA that resulted in bringing about Compute Unified Device Architecture (CUDA), the world’s first general purpose computing technology on GPU.

Typically, in GPU-accelerated applications, the sequential portion of the workload is executed on the CPU – which is optimized for a single-thread performance – while the resource-intensive portion of the application runs on thousands of GPU cores in parallel. With CUDA, developers code in popular languages such as C, C++, Fortran, Python and MATLAB, and express parallelism through extensions in the form of a few keywords.

When constructing a task, it is important to create a dependency on the solutions and the number of blocks and activity in it. This is done in order to optimize graphics card resources.

The standard program should contain an implementation of basic data on the CPU. Then it allocates memory on the CPU, copies data from the global CPU memory to global GPU memory. If identical variables are to be used in each core, they can be defined in constant memory. Next, the hardware core is started, and after the processing is completed, the results are transferred to CPU memory.

For example, the acceleration of graph algorithms [31], non-linear optimization [32], and numerical integration of ODE systems [33] are utilized.

2.2. Approaches to parallelization
In general, there are three main methods of running parallel solutions: massively parallel running, running individual elements in parallel, and running using dynamic parallelism. These three types are schematically shown in figure 1.

Since there are many solutions to the principle of mass parallelism, we will not dwell on this solution scheme, e.g. Russell [34] (figure 1a).
Let’s focus on the options for paralleling individual elements in the methods under consideration (figure 1b). After that, we use the developed algorithms with individual parallelism in the massive parallelism mode (figure 1c). Therewith we will increase the speed of calculating the function value at each iteration.

In equations (2-10) for the Suhanov method we see the necessity to calculate Stumpf function. Since we need to compute seven different Stumpf functions for one iteration, we can compute them simultaneously in parallel mode.

In Gooding method, we will compute in parallel the value of the function $F(x)$ and its derivatives $F'(x), F''(x)$, used in (11-19).

Since the Izzo method uses secant method to solve the equation, the most obvious paralleling option is to compute two values of the function $F(x)$ at points $x_1, x_2$ simultaneously, i.e. the block of formulas (5).

![Figure 1. Parallel run types: massively parallel launch (a), parallel launch of individual elements (b), dynamic concurrency (c).](image)

2.3. Solution methods

The methods chosen for developing the program were those proposed by Suhanov [12], Izzo [9] and Gooding [8] and based on solving the Kepler equation in universal variables (expressed through Stumpf functions) [23] and Lancaster equation [11]. To solve these equations, the following methods are used: the secant method (false-position method) [9], Newton method [12] and Halley method [9]. Further, programs will be written for these methods on the central and graphic processors, and running of these programs will be analyzed.

Below we will try to briefly describe the methods and show the equations to the solution of which they are reduced.

2.3.1. Sukhanov's method. As previously noted, the Stumpf functions are used to solve the problem by this method, and the rules for their evaluation are given in table 2 below.

| $x > 0$ | $C_0(x)$ | $C_1(x)$ | $C_2(x)$ |
|---------|----------|----------|----------|
| $x > 0$ | $\cos \sqrt{x}$ | $\sin \sqrt{x}$ | $1 - \cos \sqrt{x}$ |
| $x < 0$ | $\text{ch} \sqrt{-x}$ | $\text{sh} \sqrt{-x}$ | $\text{ch} \sqrt{x} - 1$ |

![Table 2. Stumpf functions.](image)
\[
C_n = \frac{1}{n!} - xc_{n+2}^2; \quad \frac{dc_n}{dx} = \frac{nc_{n+2} - c_{n+1}}{2}
\]

Algorithm:

\[
\sigma = \frac{\Delta t}{(r_0 + r_1)^2}; \quad \rho = \frac{\sqrt{2r_0 r_1}}{r_0 + r_1} \cos \frac{\varphi}{2};
\]

\[
if \sigma \leq \sigma_{\text{par}}: x_0 = 0; \text{ else } x_0 = 4(\pi - \epsilon_0)^2;
\]

\[
u = \sqrt{1 - \rho} \cdot \frac{c_1}{\sqrt{c_2}}; \quad \frac{du}{dx} = \left(-0.5 - \frac{\rho}{\sqrt{c_2 u}}\right) \cdot 0.5(c_3 - c_2) \cdot \left(\rho \cdot \frac{c_1}{4c_2^{3/2} u}\right) \cdot (c_4 - 0.5c_3)
\]

\[
F(x, \rho) = \frac{c_1}{2} u^3 + \rho u;
\]

\[
F_x = \frac{0.5(3c_3 - c_4)c_2^{3/2} - 1.5c_1\sqrt{c_2(c_4 - 0.5c_3)}}{c_2^2} u + \left(2 \cdot \frac{c_3}{c_2^{3/2}} u^2 + \frac{c_1}{c_2^{3/2}} u^2 + \rho\right) \frac{du}{dx}
\]

While $|\Delta t| > E$:

\[
\Delta x = \frac{F(x_{n+1}, \rho) - \sigma}{F_x(x_{n+1}, \rho)}
\]

\[
x_{n+1} = x_n - \Delta x
\]

The Izzo method is an extension of the Gooding method. These methods are reduced to solving the system of equations obtained by Lancaster. Later Gooding reduced the system to a single equation.

General algorithm: inputs, $r_1$, $r_2$, $\Delta t$

\[
y = \sqrt{[E]; z = \sqrt{1 + KE}};
\]

\[
f = y(z - qx); \quad g = xz - qE
\]

\[
d = \begin{cases} a \tan(f / g), E < 0 \\ a \tanh(f / g), E > 0 \end{cases}
\]

\[
F(x) = \frac{2}{E} \left(x - qz - \frac{d}{y}\right) - T = 0
\]

2.3.2. Gooding's method. To solve the Lancaster equation, Gooding proposes to use the second-order iterative Halley method, which corresponds to the following algorithm. Gooding algorithm:

\[
F' = -\frac{3x(F + T) + 4q^3}{E}; \quad F^* = -\frac{3(F + T) + 5xF' + 4\left(\frac{q}{z}\right)^3 (1 + q^2)}{E}
\]
While $|\Delta x| > E$:

$$\Delta x = \frac{2F_n F'_n}{2(F'_n)^2 - F_n F''_n}$$

(16)

$$x_{n+1} = x_n - \Delta x$$

(17)

2.3.3. *Izzo's method*. Later, Izzo proposed new initial approximations for the iterative process of solving the Lancaster equation and used the Secant method.

Izzo algorithm:

$$x_1 = -0.5233; x_2 = 0.5233$$

(17)

While $|\Delta x| > E$:

$$x^{n+1}_2 = x^{(n)}_1 F(x^{(n)}_2) - x^{(n)}_2 F(x^{(n)}_1)$$

$$\Delta x = x^n_1 - x^{n+1}_2$$

(18)

(19)

3. Results

Let’s take a few typical problems as an example. All calculations have been carried out on a station with specifications given in Table 3.

**Table 3.** Computing system characteristics.

| GPU                | Nvidia Tesla K80 |
|--------------------|------------------|
| CUDA blocks        | 4992             |
| CPU                | Intel core i9-10900X |
| Number of Cores   | 10               |

3.1. Test one. *Earth – Mars Isolines*

Let’s consider the problem of contouring isolines of the total hyperbolic excess of velocity for the Earth – Mars interplanetary flight in one synodic period (2023.08.29 – 2025.10.05) with flight duration from 100 to 700 days. Let us take a grid of initial flight data with a step for the launch date of one day and the transfer time of one day. The planet positions and velocities have been determined with the help of NASA DE/LE 405. As a result, we obtain 462 169 problems. The results of calculations are given in figure 2, and the estimates for the execution time (Table 4). Program execution time does not include the cost of reading data from NASA DE / LE 405.

**Table 4.** Results of massively parallel implementation, test 1.

| Solution methods | CPU time, ms (sequentially) | GPU time, ms (in parallel) | GPU blocks / threads |
|------------------|-----------------------------|----------------------------|----------------------|
| Gooding          | 36.445                      | 1001.2430                  | 1204/384             |
| Izzo             | 1.364                       | 37.0856                    | 1204/384             |
| Suhanov          | 4.271                       | 115.7835                   | 1204/384             |
3.2. Test two. Flight to the asteroid group
Let’s consider a flight from the Earth to a group of 1000 asteroids taken in a row from Minor Planet Center Orbit (MPCORB) database of minor planets having orbital elements in the following range: \(0.7 \leq a \leq 3.5, 0 \leq e \leq 0.4, -15^\circ \leq i \leq 15^\circ\). Launch date – 2022.04.29, flight duration – 350 days. Some of the obtained trajectories are shown in figures 2-4, and runtime estimates are given in table 5.

![Figure 2](image2.png)

**Figure 2.** Isolines of \(V_\infty\) (km/sec) for the Earth-Mars transfer trajectories.

| Solution methods | CPU time, ms (sequentially) | GPU time, ms (in parallel) | GPU kernels/threads |
|------------------|----------------------------|---------------------------|---------------------|
| Gooding          | 79.2280                    | 24.2395                   | 3/384               |
| Izzo             | 2.9460                     | 8.4783                    | 3/384               |
| Suhanov          | 8.3485                     | 9.4787                    | 3/384               |

![Table 5](image3.png)

**Table 5.** Results of massively parallel implementation, test 2.

![Figure 3](image4.png)

**Figure 3.** Top 100 trajectories to asteroids.

![Figure 4](image5.png)

**Figure 4.** Top 5 trajectories to asteroids.

3.3. Test three. Dynamic parallelism
In the final test we compare the time to solve one problem on a CPU with sequential execution and on a GPU with sequential and (dynamically) parallel execution. Programs with dynamic paralleling have been implemented as written in Part 5. Estimates of program execution time are given in table 6.
Table 6. Results of using dynamic parallelization, test 3.

| Solution methods | CPU time, ms | GPU time, ms | GPU blocks / threads | GPU time, ms (in dynamically parallel) | GPU blocks / threads |
|------------------|--------------|--------------|----------------------|----------------------------------------|---------------------|
| Gooding          | 0.1280       | 6.4061       | 1/3                  | 5.4608                                 | 2/4                 |
| Izzo             | 0.0984       | 4.9246       | 1/2                  | 4.5537                                 | 2/3                 |
| Suhanov          | 0.1090       | 5.4552       | 1/7                  | 5.0443                                 | 2/8                 |

4. Conclusion
Overall, the paper considered the possibility of parallel solving the Lambert problem on GPUs using CUDA technology. As evident from the given examples, solving one Lambert problem on GPU requires more time than on CPU, even taking the dynamic parallelism into account. According to the results of test one and test two, massively parallel running can be very efficient, but only when the number of problems to be solved is more than 1000. For efficient use of the GPU, it is desirable that the number of tasks to be solved be comparable to the number of GPU cores. However, a solution with dynamic parallelism can reduce program execution time by 15-20 percent at least. Software implementations of available on GitHub [35].

Acknowledgments
This study was supported by the grant of the Government of the Russian Federation allocated from the federal budget for state support of scientific research conducted under the guidance of leading scientists in Russian educational institutions of higher education, research institutions and state research centers of the Russian Federation (Contest 7, Resolution No.220 of the Government of the Russian Federation of April 9, 2010). Agreement No. 075-15-2019-1894 of December 3, 2019.

References
[1] Izzo D, Becerra V M, Myatt D R, Nasuto S J and Bishop J M 2007 Search space pruning and global optimisation of multiple gravity assist spacecraft trajectories. J. Global Optim. 38(2) 283 https://doi.org/10.1007/s10898-006-9106-0
[2] Shen H X, Casalino L and Luo Y-Z 2015 Global search capabilities of indirect methods for impulsive transfers. J. Aeronaut Sci. 62(3) 212 https://doi.org/10.1007/s40295-015-0073-x
[3] Luo Y-Z, Tang G-J, Lei J and Li H-Y 2007 Optimization of multiple-impulse, multiple-revolution, rendezvous-phasing maneuvers. J. Guid. Control Dyn. 30(4) 946 https://doi.org/10.2514/1.25620
[4] Konstantinov M S and Petukhov V G 2012 The analysis of manned Mars mission with duration of 1000 days. Acta Astronaut. 73 122 https://doi.org/10.1016/j.actaastro.2011.11.010
[5] Mingcheng Z, Guangming D, Lei P, Maocai W and Jinlian X 2016 Multiple gravity assist spacecraft trajectories design based on BFS and EP_DE algorithm. Int. J. Aerospace Eng. Article ID 3416046 1 https://doi.org/10.1155/2016/3416046
[6] Petukhov V G, Konstantinov M S and Fedotov G G 2007 1st ACT global trajectory optimisation competition. Acta Astronaut. 61(9) 775 https://doi.org/10.1016/j.actaastro.2007.03.006
[7] Grigoriev I S and Zapletin M P 2013 Selection of prospective asteroid sequence. Automat. Rem. Contr. 74(8) 1284 https://doi.org/10.1134/S0005117913080055
[8] Gooding R H 1990 A procedure for the solution of Lambert orbital boundary-value problem. Celestial Mechanics and Dynamical Astronomy 48(2) 145 https://doi.org/10.1007/BF00049511
[9] Izzo D 2015 Revisiting Lambert’s problem. Celest. Mech. Dyn. Astr. 121(1) 1 https://doi.org/10.1007/s10569-014-9587-y
[10] Bate R R, Mueller D D and White J E 1971 Fundamentals of Astrodynamics, (New York: Dover) p 470
[11] Lancaster E R and Blanchard R C 1969 A Unified Form of Lambert's Theorem. (Washington: National Aeronautics and Space Administration) p 18
[12] Sukhanov A A 1989 Universal solution of Lambert's problem. Cosmic Res. 26(4) 415
[13] Avanzini G 2008 A simple Lambert algorithm. J. Guid. Control Dynam. 31(6) 1587 http://dx.doi.org/10.2514/1.36426
[14] Trussing J E 2000 A class of optimal two-impulse rendezvous using multiple-revolution Lambert solutions. J. Astronaut. Sci. 48(2-3) 131 https://doi.org/10.1007/BF03546273
[15] Chen T, Kampen E, Yu H and Chu Q P 2013 Optimization of Time-open constrained Lambert rendezvous using interval analysis. J. Guid. Control Dyn. 36(1) 175 https://doi.org/10.2514/1.56773
[16] Boltz F W 1984 Second-order p-iterative solution of the Lambert/Gauss problem. J. Astronaut. Sci. 32 475
[17] Thorne J D 2014 Convergence behavior of series solutions of the Lambert problem. J. Guid. Control Dyn. 8 1 https://doi.org/10.2514/1.000701
[18] Thorne J D 1995 Series reversion/inversion of Lambert’s time function. J. Astronaut. Sci. 43(3) 45 https://doi.org/10.2514/6.1990-2886
[19] Kriz J 1976 A uniform solution of the Lambert problem. Celestial. Mech. 14(4) 509 https://doi.org/10.1007/BF01229061
[20] Simo C 1973 Solucion del problema de Lambert mediante regularizacion. Collect. Math. 24(3) 231
[21] Jezewski D J 1976 K/S two-point-boundary-value problems. Celestial. Mech. 14(1) 105 https://doi.org/10.1007/BF01247136
[22] Battin R H and Vaughan R M 1984 An elegant Lambert algorithm. J. Guidance. 7(6) 662 https://doi.org/10.2514/3.19910
[23] Stumpff K 1962 Calculation of ephemerides from initial values, (Washington: National Aeronautics and Space Administration) p 25
[24] Wailliez S E 2014 On Lambert’s problem and the elliptic time of flight equation: A simple semi-analytical inversion method. Adv. Space Res. 53(5) 890 https://doi.org/10.1016/j.asr.2013.12.033
[25] He Q, Li J and Han C 2010 Multiple-Revolution solutions of the transverse-eccentricity based Lambert problem. J. Guid. Control Dynam. 33 (1) 265 https://doi.org/10.2514/1.45041
[26] Zhang G, Mortari D and Zhou D 2010 Constrained multiple-revolution Lambert’s problem. J. Guid. Control Dynam. 33(6) 1779 https://doi.org/10.2514/1.49683
[27] Wen C, Zhao Y, and Shi P 2014 Derivative analysis and algorithm modification of transverse eccentricity-based Lambert problem. J. Guid. Control Dynam. 37(4) 1195 https://doi.org/10.2514/1.62351
[28] Arora N and Russell R P 2010 A GPU accelerated multiple revolution Lambert solver for fast mission design. Adv. Astronaut. Sci. 1477
[29] Shimoun J, Taheri E, Kolmanovsky I, and Girard A 2018 A study on GPU-enabled Lambert's problem solution for space targeting missions. Annual American Control Conf. (Milwaukee: Wisconsin Center) p 6
[30] Parrish N 2018 Accelerating Lambert's problem on the GPU in Matlab, PhD thesis, California Polytechnic State University
[31] Bisson M, Bernaschi M and Mastrostefano E 2014 Parallel distributed breadth first search on the Kepler architecture. IEEE T. Parall. Distr. 27(7) 2091 https://doi.org/10.1109/TPDS.2015.2475270
[32] Fei Y, Rong G, Wang B and Wang W 2014 Parallel L-BFGS-B algorithm on GPU Comput. Graph. 40 1 https://doi.org/10.1016/j.cag.2014.01.002
[33] Morozov A Y and Reviznikov D L 2018 Adaptive Interpolation algorithm based on a kd-tree for numerical integration of systems of ordinary differential equations with interval initial conditions. Differential Equations 54(7) 945 https://doi.org/10.1134/S001226618070121
[34] Arora N, Vittaldev V and Russell R P 2015 Parallel computation of trajectories using graphics processing units and interpolated gravity models. *J. Guid. Control Dyn*. **38**(8) 1345 https://doi.org/10.2514/1.G000571

[35] *Solution Methods*, available at: https://github.com/evilgenie-code?tab=repositories