ARTICLE IN PRESS

Available online at www.sciencedirect.com

ScienceDirect

Advances in Space Research xxx (xxxx) xxx

ADVANCES IN SPACE RESEARCH
(a COSPAR publication)

www.elsevier.com/locate/asr

Efficient computation of the geopotential gradient in graphic processing units

Carlos Rubio [a,*], Jesús Gonzalo [a], Jan Siminski [b], Alberto Escapa [a]

[a] Dept. of Aerospace Engineering, Universidad de León, León, Spain
[b] European Space Agency – European Space Operations Centre (ESA/ESOC), Darmstadt, Germany

Received 16 January 2024; received in revised form 25 March 2024; accepted 27 April 2024

Abstract

Efficient computation of the geopotential gradient is essential for numerical propagators, particularly in scenarios involving low Earth orbits. Conventional geopotential calculations are based on spherical harmonics series, which become computationally demanding as the degree/order increases. This computational burden can be mitigated by means of parallelized algorithms. Additionally, certain situations lend themselves to high parallelization, such as the propagation of space debris catalogs, satellite mega-constellations, or the dispersion of particles resulting from a space collision event. This paper introduces an optimized Graphics Processing Unit (GPU) implementation designed to facilitate extensive parallelization in the geopotential gradient calculation. The formulation developed in this study is not specific to any GPU. However, to illustrate the low-level optimizations necessary for an efficient implementation, we selected the Compute Unified Device Architecture (CUDA) as the dominant and *de facto* standard in parallel computing. Nevertheless, most of the concepts and optimizations presented in this paper are also valid for other GPU architectures. Built upon the spherical harmonic expansion using the Cunningham formulation, which is well-suited for GPU computations, our implementation offers several variants with different tradeoffs between speed and accuracy. Besides GPU double precision, we introduced a mixed precision arithmetic –a hybrid between single and double precision– that exploits GPU capabilities with a low penalty in accuracy. The proposed algorithm was implemented as a software reusable module, and its performance was evaluated against GMAT, GODOT, and Orekit astrodynamic codes. The algorithm's accuracy in double precision is comparable to such codes. The mixed precision version showed enough accuracy for LEO satellite propagation, with around 1 m difference in four days. Testing across different CUDA architectures revealed very high speed-up factors compared to a single CPU, reaching a speed-up of 645 for the mixed precision variant and 450 for the double precision one in the propagation of about 3200 objects with a geopotential of degree/order 126 × 126 using an A100 GPU device.
© 2024 COSPAR. Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Keywords: Geopotential gradient computation; LEO orbit propagation; Cunningham formulation; GPU parallelization; CUDA

1. Introduction

The demand for more efficient and faster orbital propagation is increasing with the advent of satellite mega-constellations and the space debris awareness needs. The space debris catalogs are constantly growing not only due to the increment of the objects in space but also because of the improvement in the sensor's sensitivity (Adushkin et al., 2020). It is also of particular interest the propagation of the cloud of particles generated by a collision event in space (Pardini and Anselmo, 2007) (Kaplinger et al., 2013), and the orbital covariance of space debris (Hoogendoorn et al., 2018). Many of those propagations are performed in low Earth orbit (LEO) where a more accurate representation of the Earth gravity field is necessary (Vallado and McClain, 2013).

* Corresponding author.
E-mail address: carlos.rubio@unileon.es (C. Rubio).

https://doi.org/10.1016/j.asr.2024.04.056
0273-1177/© 2024 COSPAR. Published by Elsevier B.V.
This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

Please cite this article as: C. Rubio, J. Gonzalo, J. Siminski et al., Efficient computation of the geopotential gradient in graphic processing units, Advances in Space Research, https://doi.org/10.1016/j.asr.2024.04.056

The Earth's gravity field is customarily derived from its gravitational potential function given as a sum of spherical harmonics of different degrees and orders, i.e., the geopotential. A comprehensive list of the existing gravity models is available at ICGEM (2023). In their evolution, the degree n and order m considered in the models have been progressively increased, from tens to hundreds or even thousands.

In orbital propagation, that representation is truncated according to the characteristics of the mission. For example, Vallado notes that it is usual to truncate at degree/order 70×70 for LEO applications (Vallado and McClain, 2013), and 30 × 30 for Space Situational Awareness (SSA) services (Vallado et al., 2013). Other scenarios, different from the construction of the geopotential models themselves, require higher representations like LEO precise orbit determination with degree/order 120 × 120 (Mao et al., 2021; Schreiner et al., 2023); or re-entry precise orbit determination of GOCE with 200 × 200 (Gini et al., 2015).

From a practical point of view, the computational burden associated with such gravitational potential is high. One way to optimize this demanding task is parallelizing algorithms, models, or the entire propagations using Graphics Processor Units (GPUs). Although other methods to compute the gravity field are suitable for a high level of parallelization, for example, the mass concentration (mascon) models (Russell and Arora, 2012), the standard in the astrodynamic software is to use spherical harmonics to represent the gravity potential, due to the quasi-spherical structure of the Earth. The spherical harmonic expansion needed to calculate the geopotential gradient is computationally expensive, with algorithmic complexity $\mathcal{O}(n^2)$.

A few authors addressed this problem by proposing Computer Processor Unit (CPU) and Graphics Processor Unit (GPU) parallelization alternatives. In particular, GPU devices are specially designed to perform massively parallel computations. Isupov et al. (2016) focused their work on the calculation of the Legrende polynomials in GPU for ultra-high degrees and orders. They tested their implementation based on extended-range arithmetic up to degree 53200. Also, Xiao and Lu (2007) proposed a parallelized mechanism for very high degrees but, in this case, using quadruple-precision floating-point arithmetic in GPU. Both developments add extra complexity required for those high degrees but not needed in common LEO propagations.

Another interesting work is that of Martin and Schaub (2020). They propose a parallel implementation in GPU based on the Pines formulation (Pines, 1973) using Vulkan (Khronos-Group, 2022), which is a parallel computing cross-platform API. Using that API, they obtain low-moderate speed-up ratios even for multiple simultaneous computations. In the same line, we can mention the work of Bai and Junkins (2010) about solving initial value problems by the Picard-Chebyshev method with NVIDIA GPUs, also reaching low speed-up ratios.

Regarding CPU parallelization, a very interesting proposal is made by Fukushima (2012) that equalizes the computation effort between p processors, obtaining a speed-up factor equivalent to the number of processors for moderate degrees. This is an intelligent solution for speeding up single geopotential gradient computations. However, scaling to hundreds or thousands of simultaneous propagations is more costly than with a GPU-based solution.

This study aims to propose an efficient implementation for the geopotential gradient computation in GPU devices. As a concrete GPU architecture is needed to illustrate the implementation details and to evaluate its performance, the Compute Unified Device Architecture (CUDA) was selected (NVIDIA-Corporation, 2022a). CUDA was chosen due to its dominance in the parallel computing field. It is the architecture and programming model that allows the use of the NVIDIA GPUs to execute parallel programs.

This work is tailored for its use in LEO satellite propagations. Our approach is based on the Cunningham formulation (Cunningham, 1970) that is stable, singularity-free, and low demanding in shared memory. Moreover, carefully implemented, it has enough accuracy even when executing part of the computations in single precision arithmetic.

The next sections present the base Cunningham formulation (Section 2), its adaptation and implementation details for CUDA GPUs (Sections 3 and 4), and the accuracy and performance analysis (Sections 5 and 6). The performance of the implemented reusable module was compared against the production-grade implementations used in the GMAT (NASA, 2022), GODOT (ESA, 2022), and Orekit (CS-Group, 2022) astrodynamic codes. Finally, the specifications of the hardware used in the tests are listed in Appendix A; the results of the accuracy and the performance tests in Appendix B and Appendix C; and in Appendix D how to integrate this GPU specific module into existing astrodynamic software, providing some insights about the performance gains that can be expected.

With the purpose of enabling reproducibility and promoting its use, the complete source code is made public in Rubio (2023) under a free software license.

2. Base formulation and recursions

The acceleration $\ddot{\boldsymbol{r}}$ of a space debris or satellite expressed in a quasi-inertial geocentric reference system is given by

$$\ddot{\boldsymbol{r}} = -\mu \frac{\boldsymbol{r}}{r^3} + \boldsymbol{a_p}, \quad (1)$$

where μ is the Earth gravitational parameter, $\boldsymbol{r} = (x, y, z)$ the satellite position, r its modulus, and $\boldsymbol{a_p}$ is the force per unit of mass, i.e., specific force, due to the perturbation forces such as the effect of the non-spherical Earth, atmospheric drag, solar radiation pressure, attraction of third bodies, tides, etc. (Vallado and McClain, 2013).

The total gravitational force, per unit of mass, that the Earth exerts on the satellite is given by the gradient of the geopotential at the satellite location

$$\nabla U = -\mu \frac{\boldsymbol{r}}{r^3} + \boldsymbol{a_g}. \tag{2}$$

The geopotential U is commonly given as an expansion in spherical harmonics relative to an Earth-fixed reference system (Montenbruck and Gill, 2012). Its expression is[1]

$$U = \frac{\mu}{r} \sum_{n=0}^{\infty} \sum_{m=0}^{n} \frac{R_\oplus^n}{r^n} P_{n,m}(\sin\phi) \times [C_{n,m}\cos(m\lambda) + S_{n,m}\sin(m\lambda)], \tag{3}$$

where ϕ is the geocentric latitude, λ the longitude, $R_\oplus$ the Earth reference radius, and $P_{n,m}$ is the associated Legendre polynomial of degree n and order m. The $C_{n,m}$ and $S_{n,m}$ terms are called the Stokes coefficients. Their numerical value is given by the corresponding geopotential model (JGM-2, EGM2008, etc.). The coefficients with $m = 0$ are called zonal coefficients, the ones with $n = m$ sectorial coefficients, and those with $n \neq m \neq 0$ tesseral coefficients.

The former expression of the geopotential, Eq. (3), and consequently the derived expression for the gradient has an infinite number of terms. In practice, the series is truncated to some degree N depending on the mission, as stated in the Introduction. If, as in Eq. (3), the maximum order is also given by the maximum degree N, the expansion is referred to as a square gravity model. It is also feasible to derive non-square gravity models, as indicated in Eckman et al. (2011), by employing distinct values for the degree N and order M. In this study, we exclusively focus on square gravity models, leaving the analysis of non-squared models for future research.

The evaluation of Eq. (3) to compute the geopotential and its gradient presents some difficulties. First, its expression in terms of spherical coordinates suffers singularities at the poles when calculating the geopotential gradient. Second, the direct evaluation of the associate Legendre functions is computationally intensive for high degrees and orders. Thus, different procedures have been developed to avoid the singularities and to compute the geopotential recursively. One alternative is moving the polar singularity, rotating the whole expansion (Fukushima, 2017), other is using a singularity-free formulation. Members of this last group are the Pines formulation (Pines, 1973) used in GMAT, the variant of the Clenshaw summations (Clenshaw, 1955) named *modified forward row method* and described in Holmes and Featherstone (2002) used in Orekit, and the Cunningham formulation (Cunningham, 1970) used in GODOT and in the present article.

The Cunningham formulation follows a recursion sequence that allows maintaining a minimum amount of intermediate memory, as detailed in Section 4, making it very appropriate for a GPU implementation. Other recursion sequences can be used; a summary of them can be consulted in Holmes and Featherstone (2002) and Fantino and Casotto (2009) where one can find forward and reverse methods by rows (degree) or by columns (order). However, none of them enables a greater degree of parallelization.

The gradient of the geopotential ∇U can be computed according to Cunningham (1970). Following the notation by Montenbruck and Gill (2012), its contribution to the acceleration $\boldsymbol{a} = (\ddot{x}, \ddot{y}, \ddot{z})$ is given by

$$\ddot{x} = \sum_{n=0}^{N} \sum_{m=0}^{n} \ddot{x}_{n,m}, \quad \ddot{y} = \sum_{n=0}^{N} \sum_{m=0}^{n} \ddot{y}_{n,m}, \quad \ddot{z} = \sum_{n=0}^{N} \sum_{m=0}^{n} \ddot{z}_{n,m}, \tag{4}$$

where $\ddot{x}_{n,0}$ and $\ddot{y}_{n,0}$ are

$$\ddot{x}_{n,0} = \frac{\mu}{R_\oplus^2} [-C_{n,0} V_{n+1,1}], \tag{5}$$

$$\ddot{y}_{n,0} = \frac{\mu}{R_\oplus^2} [-C_{n,0} W_{n+1,1}], \tag{6}$$

and

$$\ddot{x}_{n,m} = \frac{\mu}{2R_\oplus^2} \Big[-C_{n,m} V_{n+1,m+1} - S_{n,m} W_{n+1,m+1} + \frac{(n-m+2)!}{(n-m)!} (C_{n,m} V_{n+1,m-1} + S_{n,m} W_{n+1,m-1})\Big], \tag{7}$$

$$\ddot{y}_{n,m} = \frac{\mu}{2R_\oplus^2} \Big[-C_{n,m} W_{n+1,m+1} + S_{n,m} V_{n+1,m+1} + \frac{(n-m+2)!}{(n-m)!} (-C_{n,m} W_{n+1,m-1} + S_{n,m} V_{n+1,m-1})\Big], \tag{8}$$

for $m > 0$. The third component contributions are calculated with

$$\ddot{z}_{n,m} = \frac{\mu}{R_\oplus^2} (n - m + 1)(-C_{n,m} V_{n+1,m} - S_{n,m} W_{n+1,m}). \tag{9}$$

The factors $V_{n,m}$ and $W_{n,m}$ are obtained using recursion formulas. Starting with $W_{0,0} = 0, W_{1,0} = 0$, and

$$V_{0,0} = \frac{R_\oplus}{r}, \tag{10}$$

$$V_{1,0} = \sqrt{3} \frac{R_\oplus^2}{r^2} \sin\phi = \sqrt{3} \frac{R_\oplus^2}{r^3} z, \tag{11}$$

$$V_{1,1} = \sqrt{3} \frac{R_\oplus^2}{r^2} \cos\phi \cos\lambda = \sqrt{3} \frac{R_\oplus^2}{r^3} x, \tag{12}$$

$$W_{1,1} = \sqrt{3} \frac{R_\oplus^2}{r^2} \cos\phi \sin\lambda = \sqrt{3} \frac{R_\oplus^2}{r^3} y, \tag{13}$$

then calculating the other factors when $n = m$ using

$$V_{n,m} = (2m-1) \left[\left(\frac{xR_\oplus}{r^2} \right) V_{m-1,m-1} - \left(\frac{yR_\oplus}{r^2} \right) W_{m-1,m-1} \right], \tag{14}$$

$$W_{n,m} = (2m-1) \left[\left(\frac{xR_\oplus}{r^2} \right) W_{m-1,m-1} - \left(\frac{yR_\oplus}{r^2} \right) V_{m-1,m-1} \right], \tag{15}$$

and

$$V_{n,m} = \frac{2n-1}{n-m} \left(\frac{zR_\oplus}{r^2} \right) V_{n-1,m} - \frac{n+m-1}{n-m} \left(\frac{R_\oplus^2}{r^2} \right) V_{n-2,m}, \tag{16}$$

$$W_{n,m} = \frac{2n-1}{n-m} \left(\frac{zR_\oplus}{r^2} \right) W_{n-1,m} - \frac{n+m-1}{n-m} \left(\frac{R_\oplus^2}{r^2} \right) W_{n-2,m}, \tag{17}$$

when $n \neq m$.

[1] Hence, the resulting gravitational specific force derived from Eqs. (2) and (3) is given in an Earth-fixed reference system. To incorporate it into the equations of motion, Eq. (1), it must be transformed to the quasi-inertial reference system (Montenbruck and Gill, 2012).

Fig. 1 shows the sequences needed to calculate recursively the $V_{n,m}$ and $W_{n,m}$ terms. For the sectorial terms ($n = m$), only the previous sectorial term with index $[m-1, m-1]$ is required. For the non sectorial terms ($n \neq m$) the two previous degree terms with indexes $[n-1, m]$ and $[n-2, m]$ are needed.

3. GPU formulation

In this section, we present the initialization and recursions that allow efficient computation of the gradient of the geopotential, as well as the more appropriate strategy of summing the different spherical harmonics. Such algorithms are not specific to any particular GPU architecture.

3.1. Initialization and recursions

The first step to achieve an efficient implementation is to pre-compute all terms that do not depend on the satellite position x, y, z, and r. Those terms can be calculated at startup in the CPU and reused in the GPU for each individual position. Eqs. (5)–(9) and (14)–(17) can be factorized and rewritten as

$$\ddot{x}_{n,0} = F_n^{(0)} V_{n+1,1}, \tag{18}$$

$$\ddot{y}_{n,0} = F_n^{(0)} W_{n+1,1}, \tag{19}$$

$$\begin{aligned} \ddot{x}_{n,m} &= F_{n,m}^{(1)} V_{n+1,m+1} + F_{n,m}^{(2)} W_{n+1,m+1} \\ &+ F_{n,m}^{(3)} V_{n+1,m-1} + F_{n,m}^{(4)} W_{n+1,m-1}, \end{aligned} \tag{20}$$

$$\begin{aligned} \ddot{y}_{n,m} &= F_{n,m}^{(1)} W_{n+1,m+1} - F_{n,m}^{(2)} V_{n+1,m+1} \\ &- F_{n,m}^{(3)} W_{n+1,m-1} + F_{n,m}^{(4)} V_{n+1,m-1}, \end{aligned} \tag{21}$$

$$\ddot{z}_{n,m} = F_{n,m}^{(5)} V_{n+1,m} + F_{n,m}^{(6)} W_{n+1,m}, \tag{22}$$

$$V_{m,m} = F_{n,m}^{(7)} \left[Q_x V_{m-1,m-1} - Q_y W_{m-1,m-1} \right], \tag{23}$$

$$W_{m,m} = F_{n,m}^{(7)} \left[Q_x W_{m-1,m-1} - Q_y V_{m-1,m-1} \right], \tag{24}$$

$$V_{n,m} = F_{n,m}^{(8)} Q_z V_{n-1,m} - F_{n,m}^{(9)} Q_r V_{n-2,m}, \tag{25}$$

$$W_{n,m} = F_{n,m}^{(8)} Q_z W_{n-1,m} - F_{n,m}^{(9)} Q_r W_{n-2,m}. \tag{26}$$

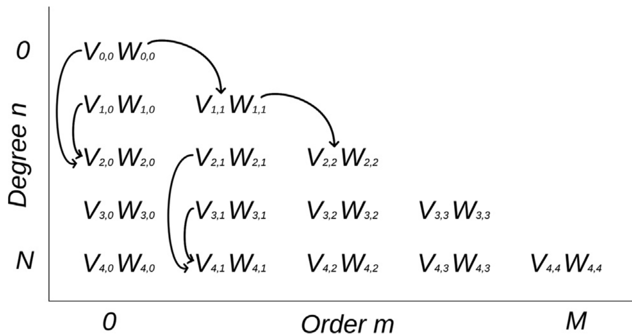

Fig. 1. Schema of the recursions used to calculate $V_{n,m}$ and $W_{n,m}$. To calculate the terms where $n = m$ only the previous sectorial term $V_{m-1,m-1}/W_{m-1,m-1}$ is required. To calculate the non sectorial terms where $n \neq m$, the two previous degree terms $V_{n-1,m}/W_{n-1,m}$ and $V_{n-2,m}/W_{n-2,m}$ are needed.

The reusable coefficients $F_{n,m}^{(i)}$ do not depend on the satellite position $\boldsymbol{r}$. They depend on the degree n, order m, and the geopotential model through $\mu, R_\oplus$, and the normalized Stokes coefficients $\overline{C}_{n,m}$ and $\overline{S}_{n,m}$, which are related to the non-normalized ones $C_{n,m}$ and $S_{n,m}$ used in Eq. (3) by (Montenbruck and Gill, 2012)

$$\left\{ \begin{array}{c} \overline{C}_{n,m} \\ \overline{S}_{n,m} \end{array} \right\} = \Pi_{n,m} \left\{ \begin{array}{c} C_{n,m} \\ S_{n,m} \end{array} \right\}, \tag{27}$$

$$\Pi_{n,m} = \sqrt{\frac{(n+m)!}{(2-\delta_{0,m})(2n+1)(n-m)!}}, \tag{28}$$

where $\delta_{0,m}$ is the Kronecker delta. As an example, the transformation for Eq. (5) into Eq. (18) is

$$\ddot{x}_{n,0} = \frac{\mu}{R_\oplus^2} \left[-C_{n,0} V_{n+1,1} \right] = F_n^{(0)} V_{n+1,1}, \tag{29}$$

$$\begin{aligned} F_n^{(0)} &= -\frac{\mu}{R_\oplus^2} \frac{\Pi_{n+1,1}}{\Pi_{n,0}} \overline{C}_{n,0} \\ &= -\left[\frac{\mu}{R_\oplus^2} \sqrt{\frac{(n+1)(n+2)(2n+1)}{2(2n+3)}} \right] \overline{C}_{n,0} \end{aligned} \tag{30}$$

Using the same procedure, we can obtain the other reusable coefficients $F_{n,m}^{(1)} \ldots F_{n,m}^{(9)}$ as

$$F_{n,m}^{(1)} = -\left[\frac{\mu}{2R_\oplus^2} \sqrt{\frac{(n+m+2)(n+m+1)(2n+1)}{2n+3}} \right] \overline{C}_{n,m}, \tag{31}$$

$$F_{n,m}^{(2)} = -\left[\frac{\mu}{2R_\oplus^2} \sqrt{\frac{(n+m+2)(n+m+1)(2n+1)}{2n+3}} \right] \overline{S}_{n,m}, \tag{32}$$

$$F_{n,m}^{(3)} = \left[\frac{\mu}{2R_\oplus^2} \sqrt{\frac{k(2n+1)(n-m+2)(n-m+1)}{2n+3}} \right] \overline{C}_{n,m}, \tag{33}$$

$$F_{n,m}^{(4)} = \left[\frac{\mu}{2R_\oplus^2} \sqrt{\frac{k(2n+1)(n-m+2)(n-m+1)}{2n+3}} \right] \overline{S}_{n,m}, \tag{34}$$

$$F_{n,m}^{(5)} = -\left[\frac{\mu}{R_\oplus^2} (n-m+1) \sqrt{\frac{(n+m+1)(2n+1)}{(n-m+1)(2n+3)}} \right] \overline{C}_{n,m}, \tag{35}$$

$$F_{n,m}^{(6)} = -\left[\frac{\mu}{R_\oplus^2} (n-m+1) \sqrt{\frac{(n+m+1)(2n+1)}{(n-m+1)(2n+3)}} \right] \overline{S}_{n,m}, \tag{36}$$

$$F_{n,m}^{(7)} = (2m-1) \sqrt{\frac{2n+1}{(n+m)(n+m-1)(2n-1)}}, \tag{37}$$

$$F_{n,m}^{(8)} = \sqrt{\frac{(2n-1)(2n+1)}{(n+m)(n-m)}}, \tag{38}$$

$$F_{n,m}^{(9)} = \sqrt{\frac{(n-m-1)(n-m-1)(2n+1)}{(n+m)(n-m)(2n-3)}}, \tag{39}$$

with $k = 1 + \delta_{m,1}$.

When computing the former expressions in double precision, it is necessary to recall that multiplication is not associative in floating-point arithmetic (Dahlquist and Björck, 2003). A very small but noticeable improvement

in accuracy is obtained if in Eqs. (30)–(36) we compute first the division $\mu/R_\oplus^2$ or $\mu/\left(2R_\oplus^2\right)$, then the terms inside the brackets, and last, the multiplication by the normalized Stokes coefficient.

The position-dependant part includes the initialization of Q_x, Q_y, Q_z, and Q_r, along with the initialization of the first values of the recursion $V_{0,0}, V_{1,0}, V_{1,1}$, and $W_{1,1}$. To minimize the number of operations and avoid using transcendental functions, we introduce the following auxiliary variables

$$H_1 = \frac{R_\oplus}{r}, \quad H_2 = \sqrt{3}H_1, \quad H_3 = \frac{H_1}{r}, \tag{40}$$

and computing Q_x, Q_y, Q_z, and Q_r from

$$Q_x = H_3 x, \quad Q_y = H_3 y, \quad Q_z = H_3 z, \quad Q_r = H_3 R_\oplus. \tag{41}$$

The first values for the recursions are obtained with

$$V_{0,0} = H_1, \quad V_{1,0} = H_2 Q_z, \quad V_{1,1} = H_2 Q_x, \quad W_{1,1} = H_2 Q_y. \tag{42}$$

Once computed these initial values and the $F^{(i)}$ factors, the recursions can be calculated in GPU using Eqs. (18)–(26).

3.2. Single and double summation

The last step is to implement the summations of Eqs. (4). The same as with multiplications, summations are not associative with respect to the resulting accuracy (Dahlquist and Björck, 2003). Indeed, the effect of an incorrect summation order leads to a significant difference in the error magnitude.

We propose two different summation strategies; the first one is called double summation method

$$a = \sum_{m=N}^{0} \left(\sum_{n=N}^{m} a_{n,m} \right), \tag{43}$$

and the second one called single summation method

$$a = \sum_{m=N}^{0} \left(\sum_{n=0}^{N} a_{n,m} \right), \tag{44}$$

where a is one of the acceleration components $\ddot{x}, \ddot{y}$, or $\ddot{z}$.

Regarding accuracy, the correct summation strategy is to sum in order from the smaller terms to the bigger ones. The bigger differences between terms occur in colatitudes near zero, i. e., at the North Pole. As an illustration, Fig. 2 shows the $\ddot{z}_{n,m}$ values for a geopotential gradient calculation with colatitude zero. The best summation order according to Fig. 2 is the double summation method, that is, first sum the elements of each column starting with the element of larger degree, and then sum the column totals from bigger to smaller. This optimal summation order requires summing the terms in a different order than the recursions provide. Therefore, it needs two memory vectors to store the intermediate results.

Degree n \ Order m	0				M
2	-2.35e-02				
	-6.81e-05	4.96e-22			
	-5.05e-05	-2.90e-22	-4.54e-40		
	-7.89e-06	-6.05e-23	-1.46e-40	-3.49e-58	
N	2.03e-05	-4.69e-23	4.75e-40	4.41e-59	8.84e-76

Fig. 2. First values of $\ddot{z}_{n,m}$ for a geopotential gradient calculation with latitude 90 degrees (zero colatitude), longitude 55 degrees, and altitude 500 km. The geopotential model is the EGM2008 (NGA, 2022), as in other parts of this work.

The recursions provide the terms in order from low to high degree, as shown in Fig. 1. In GPU, for the Cunningham formulation, we parallelize using one thread to calculate each column. Consequently, the terms of the same degree (row) are available at the same time. Based on this, we can invert the order of the column summations, which leads to the simpler but still accurate single summation method. This method requires only one memory vector to store the results of the inner summations for each column, thus optimizing the use of the memory with respect to the double summation method.

4. GPU implementation

While CPUs are optimized to process data in sequence by means of heavy cores with very high processor speeds, GPUs exploit parallelism using a great number of lightweight but slower cores. Hence, special care should be taken to optimize the GPU implementation to obtain a competitive version for low geopotential orders. An efficient implementation needs to account for the specific GPU architecture details. Ignoring these details can lead to more than one order of magnitude penalty in the execution speed. This section discusses the key aspects to consider for an efficient algorithm version based on the CUDA programming guides (NVIDIA-Corporation, 2022b). As stated in the Introducction, CUDA is the dominant and *de facto* standard in parallel computing in GPUs. The complete source code is available in Rubio (2023) for reference.

According to the CUDA programming guides, the key aspects to consider for the Maxwell, Pascal, Turing, Volta, and Ampere CUDA architectures are:

- Minimize the use of arithmetic instructions with low throughput
- Maximize parallel execution
- Optimize for high occupancy
- Avoid intra-warp divergence
- Perform coalesced access to global memory

- Avoid shared memory bank conflicts
- Optimize memory copies between host and device

4.1. Minimize the use of arithmetic instructions with low throughput

The great difference between the throughputs of single and double precision units in GPU devices makes the use of single precision, when possible, of special interest. In the latest CUDA architectures, this disparity is approximately twofold. For instance, in the A100 device corresponding to the Ampere architecture, the maximum throughput in double precision is 9.7 TFLOPS, while in single precision, it reaches 19.5 TFLOPS (NVIDIA-Corporation, 2023). This difference is even more pronounced in older CUDA architectures, making mixed precision usage more advantageous and enabling the utilization of older GPUs. Additionally, the memory allocation for double precision is also doubled, imposing a constraint on the achieved performance for certain memory-bound algorithms.

With careful implementation, the Cunningham formulation provides enough accuracy even when the majority of the computation is performed in single precision arithmetic. Specifically, the precomputed factors can be obtained in double but stored in single precision; the Eqs. (19)–(26) can be computed in single precision; and the final summations of Eqs. (43) or (44) and the input derived values ($H_1, H_2, H_3, Q_x, Q_y, Q_z, Q_r$) computed in double precision. In the sequel, we will refer to this combination as the mixed precision version, in contrast to the double precision version that performs all computations in double precision arithmetic.

To illustrate the differences between the double and the single precision versions, Fig. 3 represents the function unit utilization for 78 simultaneous geopotential gradient calculations using double precision and Fig. 4 the same for the mixed precision. In the first case, the double precision GPU unit is saturated, setting a clear limit in performance. In the second case, the utilization of the double unit is in the mid-range, and the work is more evenly shared between both single and double precision units.

4.2. Maximize parallel execution

For one geopotential gradient calculation, the limit in parallelization is given by the dependencies in the recursion scheme. As some terms depend on others, we need to calculate them following a strict serial sequence.

Except for the sectorial terms, all the dependencies are limited to their column, so we can parallelize assigning of one computation thread per column, as shown in Fig. 5.

The total number of terms to calculate for a geopotential model of order and degree n is $n^2/2$. Hence, we expect an algorithm complexity of $\mathcal{O}(n^2)$ for a single-core CPU and of $\mathcal{O}(n)$ for the GPU implementation.

Beyond the parallelization limit imposed by the recurrence relations, we can launch several geopotential gradient computations at the same time. This can be useful for one single satellite propagation when using a numeric integrator that allows some degree of parallelization like, for example, the Bulirsch-Stoer integrator (Stoer and Bulirsch, 1996) and other extrapolation and alternative methods (Fukushima, 1999). It can also be useful to launch several simultaneous propagations of different objects, as explained in the Introduction.

4.3. Optimize for high occupancy

A CUDA device is composed of an array of Streaming Multiprocessors (SMs). Each SM can execute several thread blocks at the same time. A block is an aggregate of threads with the capability of sharing information between them. Therefore, each geopotential gradient computation is circumscribed to one block with one thread per column, as represented in Fig. 5. If we want to maximize the calculations throughput, we need to launch enough blocks and threads to achieve a high level of hardware utilization. Usually, the greater number of simultaneous calculations performed in the device gives the lowest average time per individual calculation. The threads inside a block are grouped in warps, which are simply groupings of 32 threads. Occupancy is the ratio of active warps to the maximum number of possible active warps per one SM. The theoretical occupancy is restringed by the following limits:

- The maximum number of blocks per SM
- The maximum number of threads per SM
- The maximum number of registers per SM
- The maximum shared memory per block

The number of threads per block is fixed by the number of columns of $V_{n,m}$ and $W_{n,m}$ terms. As can be shown in Eqs. (20)–(22), it is needed to access terms with index $m-1$ and $m+1$; therefore, the number of threads is the desired geopotential order plus 2 (from 0 to $m+1$). The threads of a warp can issue each instruction simultaneously; in consequence, there is some performance benefit if the number of threads is multiple of 32, for example, by using a geopotential degree/order of 126 (128 threads).

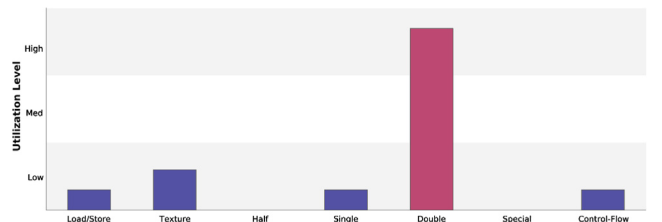

Fig. 3. Function unit utilization for 78 simultaneous calculations of gravity order 126×126 with the double precision version, executed in a GeForce GTX 1050 Ti GPU.

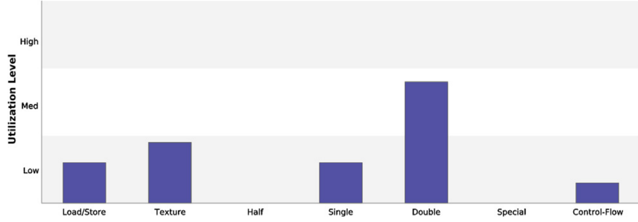

Fig. 4. Function unit utilization for 78 simultaneous calculations of gravity order 126×126 with the mixed precision version, executed in a GeForce GTX 1050 Ti GPU.

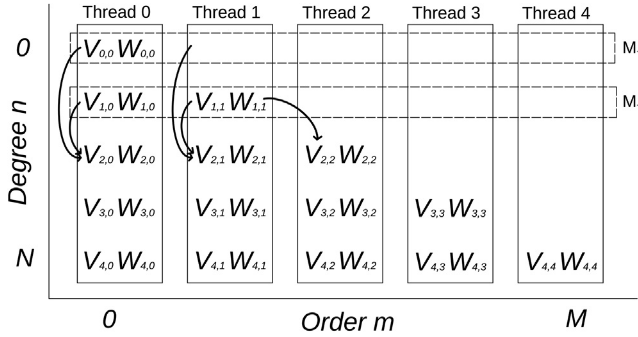

Fig. 5. Thread assignment and shared memory vectors for the $V_{n,m}, W_{n,m}$ terms.

The maximum number of registers that a thread can use is configurable at compilation time. If not limited, the required registers can exceed the maximum of the SM, and fewer blocks are executed simultaneously. Limiting the number of registers means that local memory is used in exchange for the registers, causing some performance penalty. Still, this penalty is preferable to the decrease in the achieved occupancy.

Another limit is the amount of shared memory per SM. Shared memory is the common memory used by all the threads of the same block. Each thread needs to keep in memory the two previous $V_{n,m}$ and $W_{n,m}$ terms due to the recursions scheme. Also, as shown Eqs. 20,21, each thread needs to access to the neighbour threads $V_{n,m}$ and $W_{n,m}$ terms in order to compute the acceleration contributions. Therefore, this information should be in shared memory. Another shared memory buffer is required to store the $\ddot{x}, \ddot{y}$, and $\ddot{z}$ accumulations of Eq. (44) (single summation). If the more accurate summation order of Eq. (43) is chosen, as the calculation order is different from the recursions, another intermediate buffer for $\ddot{x}, \ddot{y}$, and $\ddot{z}$ is needed (double summation). Finally, it is worth copying the input data structure to shared memory to avoid several accesses to global memory, which is much slower than shared memory. The total usage of shared memory is summarized in Table 1.

Table 1
Shared memory sizes depending on the geopotential degree n.

Buffer name	Elements	Bytes
Input (mixed precision)	7	28
Input (double precision)	7	54
V_{nm} (mixed precision)	$2(n+2)$	$8(n+2)$
V_{nm} (double precision)	$2(n+2)$	$16(n+2)$
W_{nm} (mixed precision)	$2(n+2)$	$8(n+2)$
W_{nm} (double precision)	$2(n+2)$	$16(n+2)$
Single summation	$3(n+2)$	$24(n+2)$
Double summation	$3(2n+1)$	$24(2n+1)$

4.4. Avoid intra-warp divergence

All the threads of a warp can issue one common instruction at a time. Performance is degraded if some threads need to execute different instructions according to a data-dependent condition. Because each thread processes one column, we have divergence when we need to compute different equations depending on the order m. This situation occurs when calculating the $V_{n,m}$ terms with Eqs. (10), (11), (12), (23), and (25) and $W_{n,m}$ terms with Eqs. (13), (24), and (26). It is not possible to elude the problem completely. However, we can extract the part with the costly floating-point operations out of the divergent branches. For example, for $W_{n,m}$ the code without extracting those operations looks like:

```
if(m > n or (n = 0 and m = 0) or (n = 1 and
  m = 0))
  Wnm = 0
else if(n = 1 and m = 1)
  Wnm = H2 * Qy
else if(n = m)
  Wnm = F7*(Qx*W[n-1][n-1]+Qy*V[n-1][n-
  1])
else
  Wnm=(F8*Qz)*W[n-1][m]-(F9*Qr)*W[n-2]
  [m]
```

Extracting the floating-point arithmetic out of the divergent branches looks like:

```
a,d = 0
b,c,e,f = 1
if(n = 1 and m = 1)
  a = H2, b = Qy
else if(n>=2 and n<=m)
  if(n = m)
    a = Qx, b = W[n-1][n-1], c = F7
    d = Qy, e = V[n-1][n-1], f = F7
  else
    a = F8, b = Qz, c = W[n-1][m], d = F9
    e = Qr, f = W[n-2][m]
Wnm = (a*b)*c - (d*e)*f
```

The performance effect with intra-warp divergent branches is similar to the case when all the threads execute all the lines. The first implementation needs two additions and eight floating-point multiplications per degree itera-

tion, and the second one only one addition and four multiplications. Notice that full divergence occurs only for the first warp that processes orders with m from 0 to 31. The other warps do not enter the branch of $m = 1$.

Tests for orders 94×94 and 126×126 show that using this option, there is a decrease in the use of the double precision unit from 85% to 75%. However, the extra load and store operations and the additional registers needed conduct to a total increment in the execution time. Hence, we do not recommend implementing this optimization.

The other divergence circumstance occurs when summing the x and y gradient contributions with Eqs. (18)–(21). The required code for $\ddot{x}$ is:

```
if(m = 0) x[m] += F0*V[n + 1][1]
else x[m] += F1*V[n + 1][m + 1]+F2*W[n + 1]
  [m + 1]
           + F3*V[n + 1][m-1]+F4*W[n + 1]
  [m-1]
```

where $x[m]$ is the x coordinate accumulator for order m, and the $+=$ operator means that $a \mathrel{+}= b$ equals $a = a + b$. The overload caused by the divergence is only one addition and one multiplication and occurs only for the first warp. The addition is always performed in double precision, while the multiplication is performed in double or single precision, depending on whether we are using the double or the mixed precision version of the algorithm. A good compromise solution is to extract only the addition from the divergent branch, especially to improve the mixed precision version as it is its only double precision repeated operation. The improved and recommended version for $\ddot{x}$ is:

```
if(m = 0) tmp  = F0*V[n + 1][1]
else tmp  = F1*V[n + 1][m + 1]+F2*W[n + 1]
  [m + 1]
           + F3*V[n + 1][m-1]+F4*W[n + 1]
  [m-1]
x[m] += tmp
```

4.5. Perform coalesced access to global memory

The precomputed factors $F_n^{(0)}, F_{n,m}^{(1)}, \ldots, F_{n,m}^{(9)}$, and the input data should be accessed by the program from global memory. The GPU second-level cache (L2) stores one cache line per each 128 aligned bytes of global memory. The first recommendation is to mark all pointers to factors and input data with *__restrict__ const*. This indicates to the compiler the read-only condition and allows using the read-only cache. The second optimization is to assure coalesced access, which is that all threads in a warp issue read operations for contiguous memory addresses. This requires indexing by m (order/thread), striding by n (degree), and setting the 128 bytes alignment for all of the $F^{(i)}$ arrays using a struct align keyword:

```
struct __align__(128) {
float f0[(N + 2)];
float f1[(N + 2)*(N + 2)];
...
float f9[(N + 2)*(N + 2)];
};
```

Implemented in this form, for the mixed precision version, the 32 threads of the warp peek a precomputed factor in one single operation. In the case of double precision, the request is split into two separate requests, one per each half-warp.

4.6. Avoid shared memory bank conflicts

The shared memory is organized in 32 parallel banks where successive 32-bit words are mapped to consecutive banks. When a warp accesses words in different banks, there is no conflict, and the request can be delivered in a single operation. Moreover, this is also true if several threads access the same memory location. On the other hand, when more than one thread in a warp has access to different addresses of the same bank, a conflict occurs, and the request must be split.

This situation occurs when performing the summations of Eqs. (43) and (44). In the single summation version, Eq. (44), the inner summation is performed during iteration from degree 0 to N while are computed the recursive terms. At the end of the iterations, we have a vector with the sums for each order m that must be reduced to a single total. Regarding accuracy, we need to sum the terms in a specific order, starting with the smaller ones. This prevents us from using sequential addressing in the reduction process and forces us to use interleaved addressing instead. In the interleaved addressing scheme, represented in Fig. 6, each thread sums two consecutive values. Thus, we sum terms of similar magnitude, avoiding numeric errors.

The drawback of interleaved addressing is that shared memory bank conflict occurs. The alternative is to include extra code to store the totals in a different order and use sequential addressing. However, the impact of these conflicts on the total execution time is so small that the optimization is not worth it.

For the case of the double summation version, Eq. (43), the situation is the same with the only difference that there are two reduction processes instead of one.

4.7. Optimize memory copies between host and device

The guidelines for these copies are to compact small transfers into one larger and minimize the amount of data transferred. The first one is straightforward, grouping all

the inputs needed by one execution in one array and doing the same for the output. The execution times of these transfers are quite short (1.15 μs for the input and 1.28 μs for the output in the example of the previous section and for the first GPU listed in Appendix A). Therefore, regarding input, it is preferable to add extra information, even if this implies a size increment. This extra information saves computation time on the GPU side. The input contains the following pre-calculated (in double precision) values: $H_1, H_2, H_3, Q_x, Q_y, Q_z$, and Q_r. The pre-computed factors $F^{(i)}$ are copied from host to device in a separate transfer but only once at startup. These factors are reusable because they do not depend on the satellite's position.

5. Accuracy analysis

Next, the accuracy of the algorithm is assessed for the four implementations discussed in Section 3 and Section 4, i.e., the single and double summations, and the mixed precision and the double precision arithmetics. It leads to the following four cases of study:

- Mixed precision and single summation
- Mixed precision and double summation
- Double precision and single summation
- Double precision and double summation

5.1. Accuracy evaluation

Accuracy was evaluated for each implementation by means of a spherical grid of one-degree step in latitude and ten-degree step in longitude. The gravity field was recreated with degree/order 100 × 100 from the EGM2008 Global Gravitational Model (NGA, 2022),

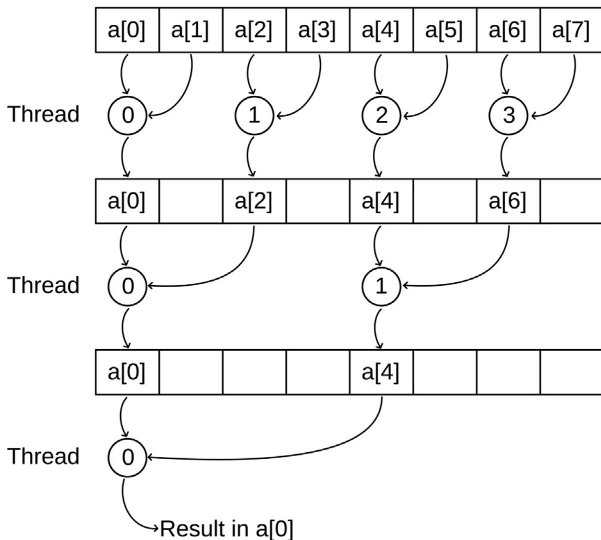

Fig. 6. Parallel reduction using interleaved addressing. The $a[i]$ variables correspond to the individual contributions to each acceleration component $\ddot{x}, \ddot{y}$, or $\ddot{z}$.

and a 500 km satellite height was used. The error was evaluated against the same computations performed in CPU in quadruple precision as

$$e = \left\| \frac{\ddot{x} - \ddot{x}_q}{a_q}, \frac{\ddot{y} - \ddot{y}_q}{a_q}, \frac{\ddot{z} - \ddot{z}_q}{a_q} \right\|_\infty, \tag{45}$$

where $\boldsymbol{a_q} = (\ddot{x}_q, \ddot{y}_q, \ddot{z}_q)$ is the quadruple precision acceleration coming from the gradient of the geopotential (in the Earth-fixed reference system) and a_q its modulus. This reference computation in quadruple precision was also used to evaluate the accuracy of the GMAT, GODOT, and Orekit geopotential implementations, whose results are included in Appendix B.

In Fig. 7 it is represented the upper bound of the errors for each colatitude, while in the Appendix B are represented all the grid points. In the mixed precision version, the accuracy differences between the single and double summation versions are so small that they overlap in the graph. In the double precision version, both kinds of summations can be distinguished, although their difference is minimal. This suggests the recommendation of exclusively utilizing the single summation version due to its lower shared memory requirement and best performance.

The accuracy achieved in double precision, with a relative error in the order of 10^{-15}, is comparable to that of production-grade astrodynamic software, as can be viewed in Appendix B. The mixed precision version achieves a relative error in the range of 6.3×10^{-8} to 4×10^{-7}, which can be enough for many LEO propagations, as shown in the next subsection.

It is worth noting that the underflow associated with the Cunningham formulation does not significantly affect the final accuracy, as appreciated in Fig. 7. The Cunningham formulation does not present singularities or overflow problems, although underflow in the $V_{n,m}$ and $W_{n,m}$ can occur even for moderate degree orders at low colatitudes. A common countermeasure for underflows is introducing a scaling factor, making all the computation scaled up several orders of magnitude and scaling down the final result. Fig. 8 shows the underflow condition in any of the $\ddot{x}_{n,m}, \ddot{y}_{n,m}$, or $\ddot{z}_{n,m}$ terms for several colatitudes. These terms follow the same underflow pattern as $V_{n,m}$ and $W_{n,m}$ according to Eqs. (20)–(22). The graph also represents the underflow limits for single and double precision.

The effect of the underflows in the computation of the $V_{n,m}$ and $W_{n,m}$ terms is that once it happens, the following terms in the same column are ignored as they will be multiplied by zero. Even if ignoring these terms, as they are quite small, the effect on the final accuracy is not significant.

5.2. Application of the mixed precision version in LEO

Given the reduction of the accuracy of the mixed precision version with respect to the double precision one, it is of particular interest to assess its potential use in a practical

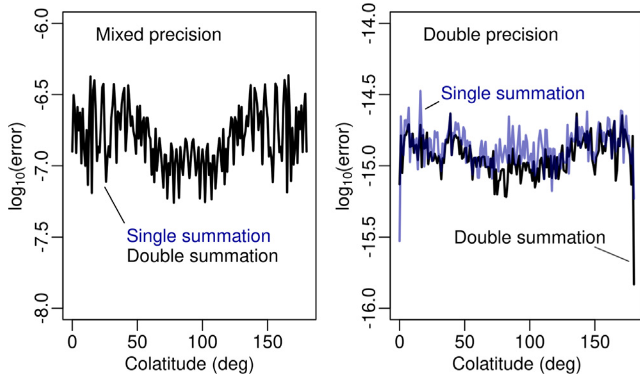

Fig. 7. Maximum relative errors as a function of the colatitude for the mixed and double precision CUDA versions (note the different scales in the OY axes).

scenario. As accurate gravity evaluation is more relevant for lower orbits, we checked the usability by means of low-height satellite propagation. The selected satellite is the IceSat (NORAD ID 27642), propagated from the TLE at epoch Feb 19, 2003. The satellite follows a near-circular orbit at a height of 598.7 km. The most common perturbations were modeled using the Orekit framework. These perturbations include the drag and radiation forces using isotropic models, Sun and Moon attraction, solid and ocean tides effect, relativity corrections, and the different degrees and orders in the employed geopotential model (EGM2008). The Orekit framework is executed on the CPU, but using the POSIX queue mechanism (Kerrisk, 2010), it sends the requests to the reusable module that computes the geopotential gradient on the GPU.

In Fig. 9 we represent the contribution to the distance differences after four days of propagation by the different perturbations. This graph is similar to the ones existing in Vallado and McClain (2013). The double precision geopotential models 12×12, 30×30, 100×100, and 200×200 calculate the difference comparing with its nearest lower-degree geopotential case. The mixed precision models 12×12mp, 30×30mp, and 100×100mp calculate the differences comparing with its counterpart in double

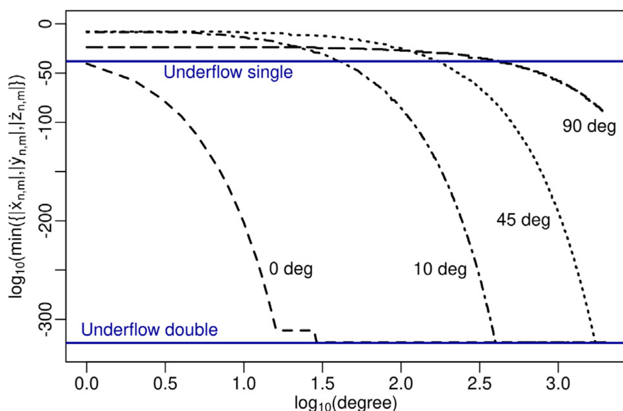

Fig. 8. Smaller acceleration terms for 0, 10, 45, and 90 degree colatitude angles. The horizontal lines indicate the underflow limits for single and double precision.

precision. In other words, the 100×100mp line represents the distance difference between the 100×100 double precision and the 100×100 mixed precision versions.

The differences introduced by the mixed precision versions are lower than the 100×100 geopotential perturbation, which in turn is lower than the relativity effect. This leads to the conclusion that the mixed precision version with order 100×100 is precise enough to use in LEO propagations with this time horizon, giving distance differences in the fourth day of propagation about 1 m when compared to the double precision version. As execution in GPU is performed in warps of 32 threads, the execution time is similar for 100×100 than for 126×126. Therefore, a very interesting choice for GPU calculation is the mixed or double precision version, depending on the desired accuracy, and single summation of degree/order 126×126.

6. Performance

The GPUs are much more effective in the computation tasks that exploit their massive parallelism. However, in the case of serial execution or low parallelism levels, they are at a disadvantage with respect to the CPU.

In Fig. 10 are represented the single (one gradient computation) execution times for the GPU and the GMAT, GODOT, and Orekit CPU implementations. All CPU implementations were executed using a single CPU core. Regarding the GPU, the execution times encompass the entire kernel execution, which reflects the computational time of the algorithm itself. If the module is invoked from an external CPU framework rather than from the GPU, the slight additional latency required for transferring the requests and responses can be computed as indicated in Appendix D.

For the performance tests, the particular satellite position $\boldsymbol{r}$ is irrelevant; therefore, all the computations use a random one. As in Section 4, the details of the hardware used for all the CPU and GPU tests are listed in the Appendix A. The aforementioned algorithm complexities, $\mathcal{O}(n^2)$ for CPU and $\mathcal{O}(n)$ for GPU, are visible as a quadratic curve in the case of CPU and a linear trend in GPU. For low geopotential degree/order, there is not enough parallelism level to overcome the GPU's lower clock speeds, the less floating-point arithmetic throughput, and the extra memory transfers and latencies.

Fig. 11 shows an ampliation for the lower geopotential degrees where we can see the cross points where the GPU starts to be faster in the case of a single computation. Orekit, implemented in the Java language, exhibits significantly lower performance and greater variability compared to GMAT and GODOT. Therefore, it may not be the most suitable reference for time comparisons.

Using the GMAT and GODOT performances as a reference, we can see the cross point around degree/order 120 for the CUDA mixed precision version and around 160 for the CUDA double precision one. Hence, in the sce-

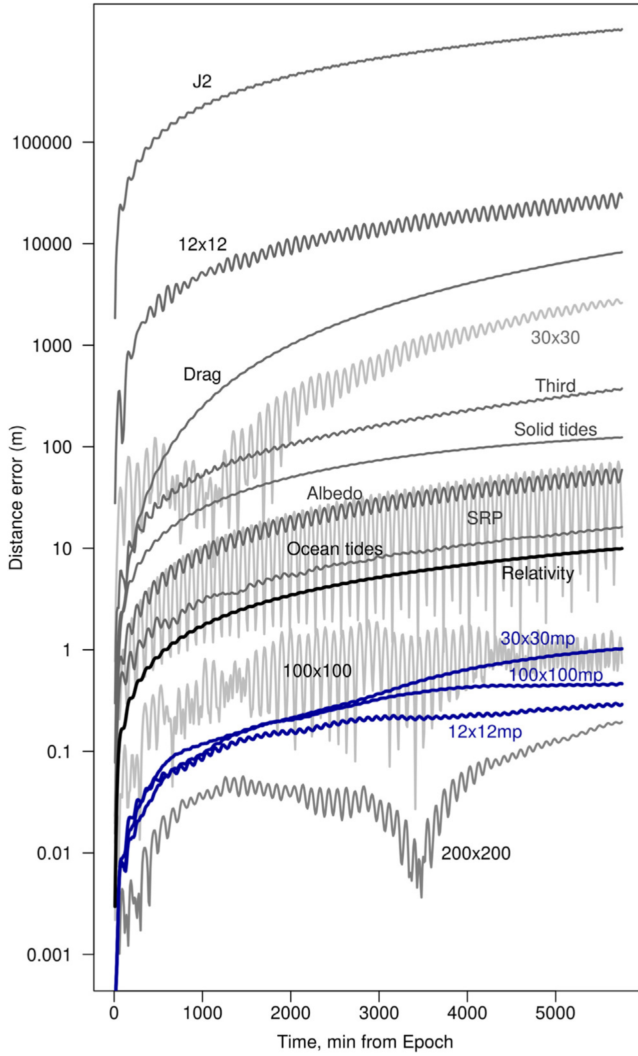

Fig. 9. Distance differences caused by the different perturbations to an IceSat satellite numeric propagation. The 12×12mp, 30×30mp, and 100×100mp represent the difference caused by the mixed precision with respect to the double precision CUDA versions of the same degree.

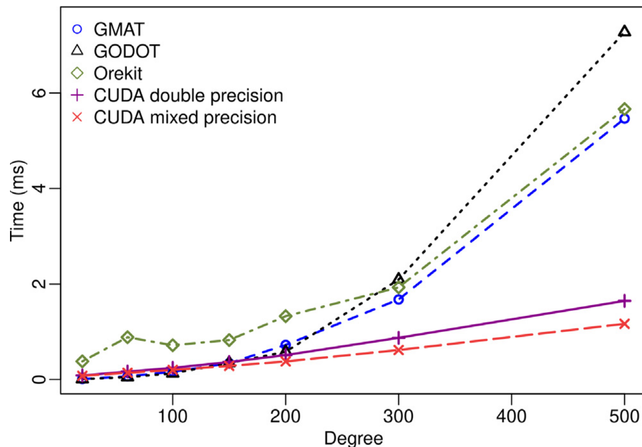

Fig. 10. Single geopotential gradient computation times. Measured as the mean value of ten computations after a warm-up of another ten. The CUDA versions were executed in a NVIDIA GeForce GTX 1050i.

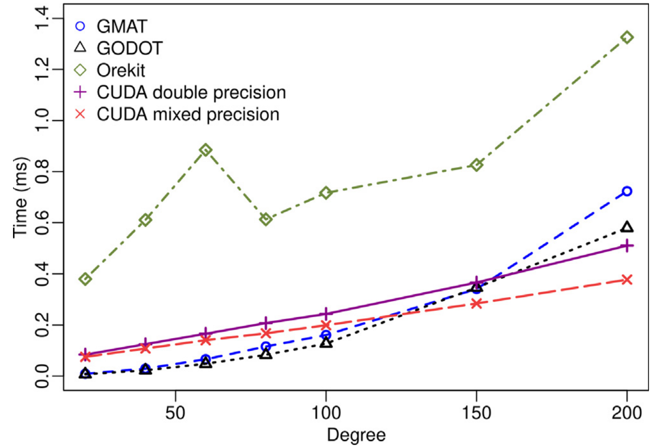

Fig. 11. Single geopotential gradient computation times zoomed in the low-degree region.

nario of a single LEO propagation that needs a moderate geopotential degree, the CUDA versions do not provide performance improvement with respect to the highly optimized CPU implementations. Nonetheless, as mentioned in the Introduction, some missions require a high order in the computation of the geopotential; for these scenarios, our implementation is faster even for one single computation.

The significant difference in performance occurs when the parallelism level and occupancy are increased. This can be made by increasing the degree/order and/or executing several simultaneous calculations. Tables 2,3 show the achieved CUDA speed-up factors with respect to the average time between GMAT and GODOT for different geopotential degrees. Their particular values are device-dependant, although they show similar trends. Thus, in Tables 2,3, we have considered a CUDA device of the Ampere architecture, while in Appendix C, we present similar tables but for the Pascal and Volta architecture devices. The number of simultaneous computations and the degree/

Table 2
Execution times, occupancy, and speed-up factors with respect to CPU for the mixed precision version. NVIDIA Ampere A100.

Degree Order	Simultaneous computations	Total time (ms)	Occupancy	Speed-up factor
62	3456	0.53	0.47	328.3
94	3456	0.81	0.77	464.5
126	3456	1.22	0.84	645.6
158	3348	1.63	0.82	756.2

Table 3
Execution times, occupancy, and speed-up factors with respect to CPU for the double precision version. NVIDIA Ampere A100.

Degree Order	Simultaneous computations	Total time (ms)	Occupancy	Speed-up factor
62	3456	0.77	0.79	224.5
94	3456	1.21	0.82	311.9
126	3240	1.63	0.86	452.3
158	3132	2.15	0.84	537.3

order were selected to maximize the GPU occupancy. Hence, the tables illustrate the maximum expected speed-up using our proposed implementation. For the suggested 126×126 single summation version and optimizing occupancy on an A100 GPU device, we achieve speed-ups of up to 450 for the double precision version and 645 for the mixed precision one compared to the serial execution in a CPU.

7. Conclusion

The Cunningham formulation is well suited to calculate the geopotential gradient in GPU. It allows a forward-column parallelization that requires a low amount of shared memory. The algorithm is numerically stable, and if carefully implemented, it provides enough accuracy even when using mixed precision arithmetic, a hybrid that combines single and double precision. In this paper, we presented the adapted formulation and the details needed to obtain an effective GPU implementation. Additionally, we worked out variants employing mixed and double precision arithmetic, as well as two different summation schemes. Among these variants, the 126×126 single summation version stands out as a well-balanced compromise between accuracy and performance.

We have developed the algorithm as a reusable module that can be seamlessly integrated into a GPU framework or invoked from external CPU software using the POSIX queue mechanism with very minimal interconnection overhead.

The algorithm accuracy using double precision arithmetic is comparable to other CPU implementations, like the ones used in GMAT, GODOT, or Orekit. Regarding the mixed precision version, we checked its usability for the lower orbits by means of a LEO propagation, comparing its induced error with that of the other perturbations. In terms of distance, after four days of propagation, the difference between the double precision and the mixed precision variants is about 1 m, considering the most common perturbations and a geopotential of degree/order 126 × 126.

In terms of execution speed, the GPU version is not competitive with respect to the most efficient CPU implementations for low degree/order and one single computation. For high degree/order (above 120 for double precision and 160 for mixed precision), the GPU version is faster because the algorithmic complexity is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. The use case where the GPU manifests all its potential and clearly outperforms the CPU implementations is when calculating the geopotential gradient for several points at the same time. We tested simultaneous calculations in three different devices, each one corresponding to one different CUDA architecture. The results showed speed-up factors as high as 645 for the mixed precision version and 450 for double precision with only one single device (NVIDIA A100 GPU) in the computation of about 3200 satellite positions with a geopotential degree/order 126 × 126.

8. Code availability statement

In the interest of facilitating further research, promoting its use, and allowing the reproduction of the experiments, we made available the complete CUDA code in Rubio (2023) under the GNU Lesser General Public License version 3 (LGPLv3).

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Appendix A. Benchmark hardware specifications

The CPU tests were executed in a system with the following characteristics:

- CPU: AMD FX-4300 Quad-Core Processor. Clock frequency 3.9 GHz. L2 cache size 4MiB. L3 cache size 4MiB. System memory 16GiB DDR3-1600 MHz.

The GPU performance tests were executed in the following GPU devices:

- GPU 1: GeForce GTX 1050 Ti, rev 6.1. Architecture Pascal. Stream multiprocessors 6. CUDA cores 768. Clock frequency 1.392 GHz. Memory 4 GB GDDR5.
- GPU 2: Tesla V100-SXM2-16 GB, rev: 7.0. Architecture Volta. Stream multiprocessors 80. CUDA cores 5120. Base clock frequency 1.245 GHz. Boost clock frequency 1.38 GHz. Memory 16 GB HBM2.
- GPU 3: NVIDIA A100-SXM4-40 GB, rev: 8.0. Architecture Ampere. Stream multiprocessors 108. CUDA cores 6192. Base clock frequency 1.095 GHz. Boost clock frequency 1.41 GHz. Memory 40 GB HBM2e.

Appendix B. Accuracy plots

This appendix includes the accuracy plots of the four proposed CUDA versions (mixed and double precision and single and double summations). It also includes similar plots for the geopotential gradient computations in GMAT, Orekit, and GODOT. Each point corresponds to a spherical grid with one-degree and ten-degree steps in latitude and longitude angles. The grid points are on a spherical surface that simulates a 500 km height satellite. The geopotential was recreated from the EGM2008 Global Gravitational Model (NGA, 2022) with degree/order 100×100. The relative errors are calculated according to Eq. (45) (see Figs. B.12, B.13, B.14, B.15, B.16, B.17, B.18).

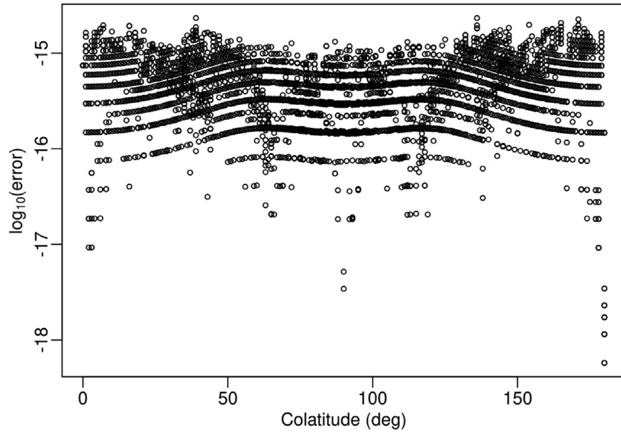

Fig. B.12. Relative errors of the double precision, single summation CUDA version.

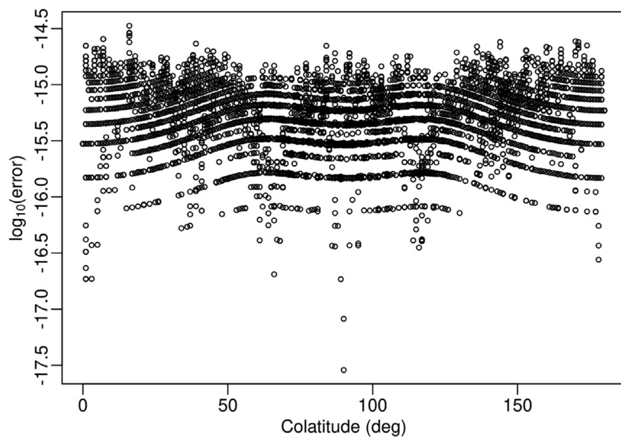

Fig. B.13. Relative errors of the double precision, double summation CUDA version.

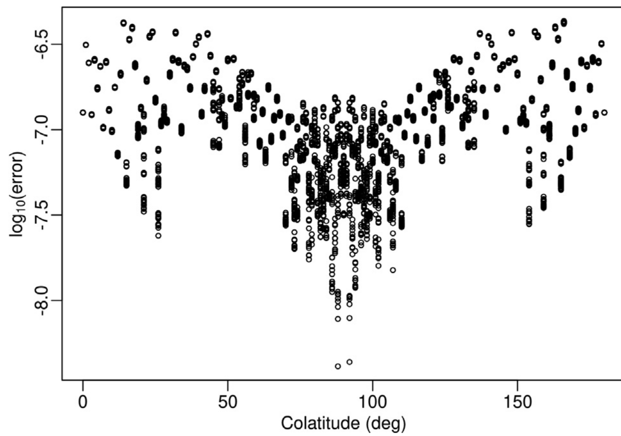

Fig. B.14. Relative errors of the mixed precision, single summation CUDA version.

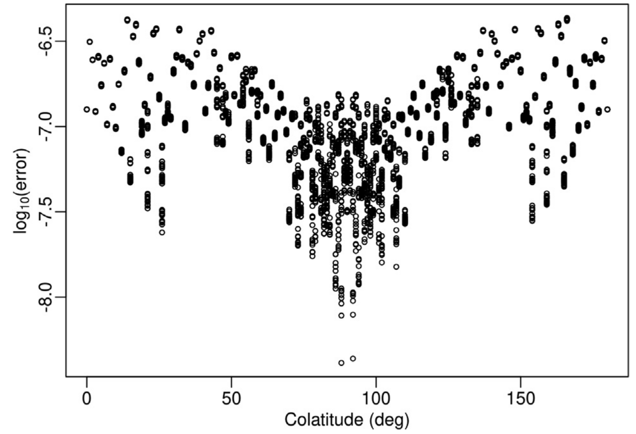

Fig. B.15. Relative errors of the mixed precision, double summation CUDA version.

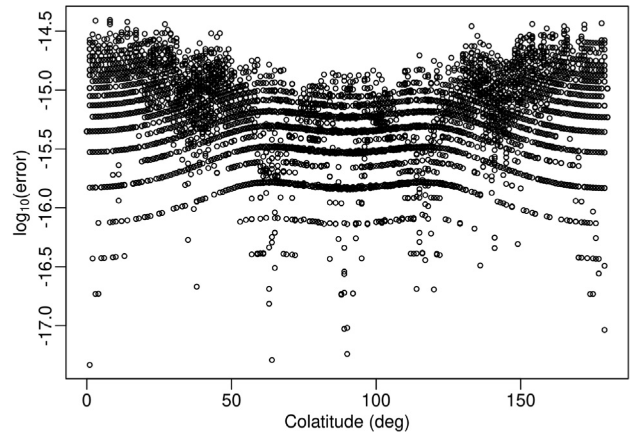

Fig. B.16. Relative errors of the GMAT implementation based in the Pines formulation. Obtained with GMAT version R2020a.

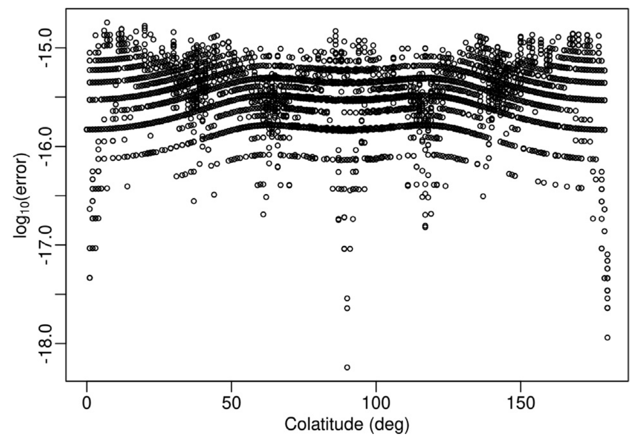

Fig. B.17. Relative errors of the GODOT implementation based in the Cunningham formulation. Obtained with GODOT version 0.8.0.

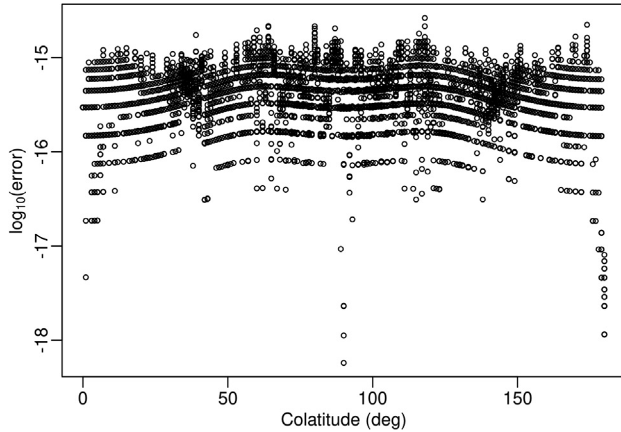

Fig. B.18. Relative errors of the Orekit implementation based in the Clenshaw summations variant described in the Holmes and Featherstone paper. Obtained with Orekit version 10.3.1.

Appendix C. Performance in Pascal and Volta GPUs

This appendix includes the results for simultaneous computations in devices of the Pascal and Volta architectures. The number of simultaneous computations is selected to achieve the maximum occupancy in the GPU. In the case of the GTX 1050i device, occupancy is constrained by the maximum blocks per SM limit. Additionally, for the GTX 1050i, the occupancy of the double precision versions is also restricted by the amount of shared memory per multiprocessor (see Tables C.4, C.5, C.6, C.7).

Table C.4
Execution times, occupancy, and speed-up factors with respect to CPU for the mixed precision version. NVIDIA GeForce GTX 1050i.

Degree Order	Simultaneous computations	Total time (ms)	Occupancy	Speed-up factor
62	192	0.33	0.80	28.6
94	126	0.46	0.88	30.1
126	108	0.70	0.76	34.9
158	84	0.85	0.62	36.6

Table C.5
Execution times, occupancy, and speed-up factors with respect to CPU for the double precision version. NVIDIA GeForce GTX 1050i.

Degree Order	Simultaneous computations	Total time (ms)	Occupancy	Speed-up factor
62	150	0.65	0.71	11.5
94	102	0.89	0.76	12.5
126	78	1.14	0.59	15.6
158	60	1.39	0.44	16.0

Table C.6
Execution times, occupancy, and speed-up factors with respect to CPU for the mixed precision version. NVIDIA Tesla V100.

Degree Order	Simultaneous computations	Total time (ms)	Occupancy	Speed-up factor
62	2560	0.44	0.70	288.4
94	2560	0.68	0.78	409.7
126	1280	0.54	0.81	534.8
158	960	0.60	0.87	594.6

Table C.7
Execution times, occupancy, and speed-up factors with respect to CPU for the double precision version. NVIDIA Tesla V100.

Degree Order	Simultaneous computations	Total time (ms)	Occupancy	Speed-up factor
62	2000	0.48	0.65	207.2
94	2000	0.95	0.69	230.9
126	1360	1.09	0.77	283.0
158	800	0.95	0.75	310.4

Appendix D. Integration aspects

This appendix introduces some aspects related to integrating this specific module into existing astrodynamics software. Calculating the gravitational gradient on a GPU can be used to accelerate orbit integrations where the numeric integration process runs on a CPU, resulting in a hybrid CPU/GPU implementation, or it can be used in software entirely developed on GPU.

The two key aspects to consider are the interconnection of the new module into the existing software and determining in which cases its incorporation offers a better performance.

D.1. Module integration on astrodynamic software

In the case of integrating the module into software already developed on GPU, the solution is straightforward, either by incorporating the proposed source code into existing CUDA code or by making a kernel-to-kernel call. The kernel-to-kernel launch overhead is minimal, on the order of a few microseconds.

For the hybrid CPU/GPU solution, a highly effective method is the utilization of POSIX queues (Kerrisk, 2010). This approach was employed in this study to execute the propagations of Section 5.2 from Orekit calling to the CUDA gravity module. The source code for managing the queues from CUDA is also available in the repository (Rubio, 2023) in the files *middleware.c* and *middleware.h*. The latency added by this method is very small and almost proportional to the number of inputs. This delay can be approximated by a linear regression as illustrated in Fig. D.19.

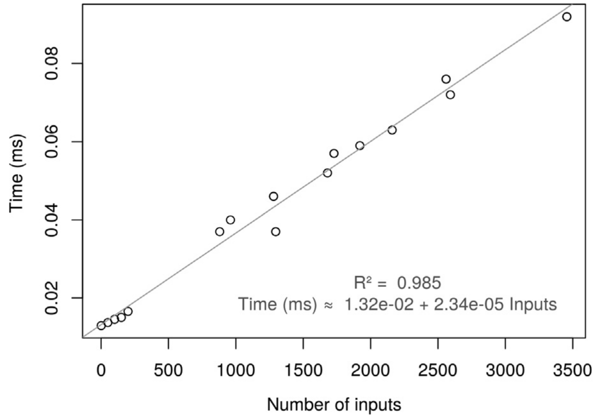

Fig. D.19. POSIX queue bi-directional latency vs number of inputs. Each individual input represents a position for computing the gravity gradient. The latency time includes the two necessary transfers, one for the request and one for the response. This data was obtained with the CPU and the GPU 1 described in Appendix A.

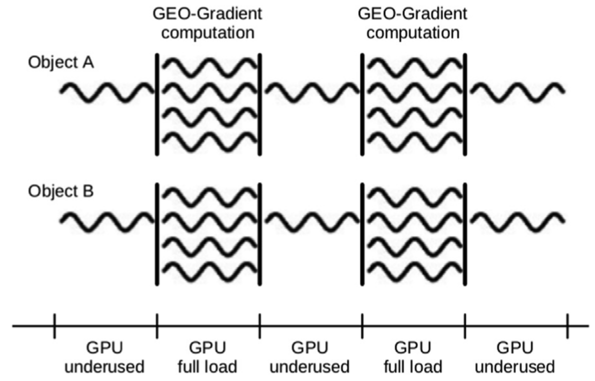

Fig. D.20. Thread splitting due combine parallelization levels 1 and 3.

D.2. Expected performance gain

The performance gain depends greatly on the use case. First, we must define the different levels of parallelization that we can employ. The first level is object-level parallelization, assigning a separate and independent execution thread for each object propagation. The second level is the integrator parallelization, where we can evaluate the acceleration of the object at different points independently and, therefore, in parallel during the integration of the same trajectory. Finally, the third level is at the level of the force/acceleration models, where we use multiple execution threads to evaluate the acceleration of the object at the same point. This article describes the parallelized computation of the acceleration due to the gravitational field, meaning it is parallelization at the third level. Another type of third-level parallelization occurs in the case of multi-faceted models for drag or SRP. In these cases, the contribution to the total acceleration of each face can be computed in parallel.

The three levels of parallelization can be combined simultaneously. In the case of combining all of them, we would have a series of m threads, one for the propagation of each trajectory, which at some time during the numeric integration process are splitted into n threads to calculate the acceleration simultaneously at various points, and then further separated into k threads when we reach a parallelized force model at the third level. This splitting process is illustrated in Fig. D.20 for a combination of the first and third parallelization levels.

The strategies and performance improvements associated with integrating a third level parallelized module can vary significantly depending on whether we are dealing with a pure GPU implementation or a hybrid CPU/GPU software.

D.2.1. Performance considerations for a pure GPU implementation

The key factor here is the aforementioned thread splitting. Orbital propagation software implemented on GPUs, such as CUDAjectory (Geda, 2019), is usually based on the first parallelization level, meaning massive propagation of independent trajectories. If the number of trajectories we want to propagate is such that all available execution threads on the GPU are utilized, we wouldn't obtain any performance increase due to the inability to perform the required thread splitting. In contrast, for cases where we have a smaller number of trajectories to propagate, we would indeed achieve better performance using the proposed module. Compared to results provided by other authors of equivalent GPU implementations of geopotential gradient calculation (Martin and Schaub, 2020), the gain with the proposed method is very significant, as detailed in Section 6.

D.2.2. Performance considerations for a hybrid CPU/GPU implementation

In the case of a hybrid CPU/GPU implementation, we typically start with non-parallelized astrodynamics software. In this scenario, the geopotential gradient calculation module works as an accelerator for a specific computation within the integration process. To achieve a performance increase, as mentioned in Section 6 and illustrated in Figs. 10 and 11, we either need a high degree/order or have to compute multiple calculations simultaneously. Regardless of being able to have that multiplicity of calculations propagating multiple objects simultaneously, for the hybrid CPU/GPU solution, it is very beneficial to incorporate a second-level parallelization, i.e., a parallelized numeric integrator. Picard iterative integrators (Fukushima, 1999), for example, allow for many parallel evaluations of acceleration, thus allowing the exploitation of the performance improvement offered by the proposed algorithm.

References

Adushkin, V., Aksenov, O.Y., Veniaminov, S., et al., 2020. The small orbital debris population and its impact on space activities and ecological safety. Acta Astronaut. 176, 591–597. https://doi.org/10.1016/j.actaastro.2020.01.015, URL: https://linkinghub.elsevier.com/retrieve/pii/S0094576520300175.

Bai, X., Junkins, J.L., 2010. Solving initial value problems by the Picard-Chebyshev method with NVIDIA GPUs. Adv. Astronaut. Sci. 136, 1459–1476.

Clenshaw, C., 1955. A note on the summation of the Chebyshev series. Math. Comput. 9, 118–120. https://doi.org/10.1090/S0025-5718-1955-0071856-0.

CS-Group (2022). About Orekit. URL: https://www.orekit.org/ accessed on 17 April 2022.

Cunningham, L.E., 1970. On the computation of the spherical harmonic terms needed during the numerical integration of the orbital motion of an artificial satellite. Celest. Mech. 2 (2), 207–216. https://doi.org/10.1007/BF01229495, URL: http://link.springer.com/10.1007/BF01229495.

Dahlquist, G., Björck, A., 2003. Numerical methods. Dover Publications, Mineola, N.Y.

Eckman, R.A., Brown, A.J., Adamo, D.R., 2011. Normalization of Gravitational Acceleration Models. Technical Report JSC-CN-23097 NASA.

ESA (2022). GODOT documentation. URL: https://godot.io.esa.int/docs/0.7.0/# accessed on 17 April 2022.

Fantino, E., Casotto, S., 2009. Methods of harmonic synthesis for global geopotential models and their first-, second- and third-order gradients. J. Geodesy 83 (7), 595–619. https://doi.org/10.1007/s00190-008-0275-0, URL: http://link.springer.com/10.1007/s00190-008-0275-0.

Fukushima, T., 1999. Parallel/vector integration methods for dynamical astronomy. Celest. Mech. Dynam. Astron. 73 (1/4), 231–241. https://doi.org/10.1023/A:1008311500582, URL: http://link.springer.com/10.1023/A:1008311500582.

Fukushima, T., 2012. Parallel computation of satellite orbit acceleration. Comput. Geosci. 49, 1–9. https://doi.org/10.1016/j.cageo.2012.07.009, URL: https://linkinghub.elsevier.com/retrieve/pii/S0098300412002348.

Fukushima, T., 2017. Rectangular rotation of spherical harmonic expansion of arbitrary high degree and order. J. Geodesy 91 (8), 995–1011. https://doi.org/10.1007/s00190-017-1004-3, URL: http://link.springer.com/10.1007/s00190-017-1004-3.

Geda, M., 2019. Massive Parallelization of Trajectory Propagations Using GPUs. Delft University of Technology, Ph.D. thesis.

Gini, F., Otten, M., Springer, T., et al., 2015. Precise orbit determination of the GOCE re-entry phase. In 5th International GOCE User Workshop, volume 728. ESA, Paris.

Holmes, S.A., Featherstone, W.E., 2002. A unified approach to the Clenshaw summation and the recursive computation of very high degree and order normalised associated Legendre functions. J. Geodesy 76 (5), 279–299. https://doi.org/10.1007/s00190-002-0216-2, URL: http://link.springer.com/10.1007/s00190-002-0216-2.

Hoogendoorn, R., Mooij, E., Geul, J., 2018. Uncertainty propagation for statistical impact prediction of space debris. Adv. Space Res. 61 (1), 167–181. https://doi.org/10.1016/j.asr.2017.10.009, URL: https://linkinghub.elsevier.com/retrieve/pii/S0273117717307305.

ICGEM (2023). ICGEM Inernational Center for Global Gravity Field Models. URL: http://icgem.gfz-potsdam.de/home accessed on 27 November 2023.

Isupov, K., Knyazkov, V., Kuvaev, A., et al., 2016. Parallel Computation of Normalized Legendre Polynomials Using Graphics Processors. In: Voevodin, V., Sobolev, S. (Eds.), Supercomputing, volume 687. Springer International Publishing, Cham, pp. 172–184. https://doi.org/10.1007/978-3-319-55669-7_14, series Title: Communications in Computer and Information Science.

Kaplinger, B., Wie, B., Dearborn, D., 2013. Nuclear fragmentation/dispersion modeling and simulation of hazardous near-Earth objects. Acta Astronaut. 90 (1), 156–164. https://doi.org/10.1016/j.actaastro.2012.10.013, URL: https://linkinghub.elsevier.com/retrieve/pii/S0094576512003918.

Kerrisk, M., 2010. The Linux programming interface: a Linux and UNIX system programming handbook. No Starch Press, San Francisco.

Khronos-Group (2022). Vulkan Cross platform 3D Graphics. URL: https://www.vulkan.org/ accessed on 17 April 2022.

Mao, X., Arnold, D., Girardin, V., et al., 2021. Dynamic GPS-based LEO orbit determination with 1 cm precision using the Bernese GNSS Software. Adv. Space Res. 67 (2), 788–805. https://doi.org/10.1016/j.asr.2020.10.012, URL: https://linkinghub.elsevier.com/retrieve/pii/S0273117720307237.

Martin, J.R., & Schaub, H. (2020). GPGPU Implementation of Pines' Spherical Harmonic Gravity Model. In Proceedings of the AAS/AIAA Astrodynamics Specialist Conference (p. 94). Virtual Event: American Astronautical Society volume 175.

Montenbruck, O., Gill, E., 2012. Satellite orbits: models, methods and applications, 1st ed. Springer, Berlin.

NASA (2022). GMAT. URL: https://sourceforge.net/projects/gmat/ accessed on 17 April 2022.

NGA (2022). National Geospatial-Intelligence Agency - Office of Geomatics. URL: https://earth-info.nga.mil/ accessed on 17 April 2022.

NVIDIA-Corporation (2022a). About CUDA. URL: https://developer.nvidia.com/about-cuda accessed on 17 April 2022.

NVIDIA-Corporation (2022b). CUDA Programming Guides. URL: https://docs.nvidia.com/cuda/index.html#programming-guides accessed on 15 April 2022.

NVIDIA-Corporation (2023). Nvidia Ampere architecture in depth. URL: https://developer.nvidia.com/blog/nvidia-ampere-architecture-in-depth/ accessed on 28 November 2023.

Pardini, C., & Anselmo, L. (2007). Evolution of the debris cloud generated by the FENGYUN-1C fragmentation event. In Proceedings of the 20th International Symposium on Space Flight Dynamics. Annapolis, MD, USA.

Pines, S., 1973. Uniform Representation of the Gravitational Potential and its Derivatives. AIAA Journal 11 (11), 1508–1511. https://doi.org/10.2514/3.50619, URL: https://arc.aiaa.org/doi/10.2514/3.50619.

Rubio, C. (2023). CUDA Gravity Gradient. URL: https://bitbucket.org/carlos-rubio/cuda-cunningham accessed on 26 November 2023.

Russell, R.P., Arora, N., 2012. Global point mascon models for simple, accurate, and parallel geopotential computation. J. Guid., Control, Dynam. 35 (5), 1568–1581. https://doi.org/10.2514/1.54533, URL: https://arc.aiaa.org/doi/10.2514/1.54533.

Schreiner, P., König, R., Neumayer, K.H., et al., 2023. On precise orbit determination based on DORIS, GPS and SLR using Sentinel-3A/B and -6A and subsequent reference frame determination based on DORIS-only. Adv. Space Res. 72 (1), 47–64. https://doi.org/10.1016/j.asr.2023.04.002, URL: https://linkinghub.elsevier.com/retrieve/pii/S027311772300265X.

Stoer, J., Bulirsch, R., 1996. Introduction to numerical analysis. Number 12 in Texts in applied mathematics, 2nd ed. Springer, New York.

Vallado, D.A., McClain, W.D., 2013. Fundamentals of astrodynamics and applications. Number 21 in Space technology library, 4th ed. Microcosm Press, Hawthorne, Calif.

Vallado, D.A., Virgili, B.B., Flohrer, T., 2013. Improved SSA through orbit determination of two-line element sets. In Proceedings of the 6th European Conference on Space Debris (pp. 22–25). Darmstadt, Germany.

Xiao, H., Lu, Y., 2007. Parallel computation for spherical harmonic synthesis and analysis. Comput. Geosci. 33 (3), 311–317. https://doi.org/10.1016/j.cageo.2006.07.005, URL: https://linkinghub.elsevier.com/retrieve/pii/S0098300406001452.