ESA GNC & ICATT Conference 2026
28 Sep. – 2 Oct. 2026, Sevilla, Spain
https://atpi.eventsair.com/gncicatt/

Onboard orbit propagation on a low-precision space-graded GPU

C. Matarrese[1,*], G. Battaglia[1], A. Nicali[1], C. Giordano[1], G. Meoni[2], G. Giuffrida[3], and P. Panicucci[1]

[1]*Department of Aerospace Science and Technology, Politecnico di Milano, Via Giuseppe la Masa, 34, 20156 Milan, Italy*
[2]*European Space Agency, Frascati, 00044, Italy*
[3]*IngeniArs s.r.l., Pisa, 56121, Italy*
[*]*Corresponding author: claudio.matarrese@polimi.it*

ABSTRACT

Large-scale trajectory propagation is essential for uncertainty quantification, space situational awareness, GNC, mission planning. Growing satellite numbers and mission complexity make ground-based computation increasingly impractical, motivating edge computing onboard. Although radiation-tolerant GPUs enable in-orbit parallel propagation, their limited performance requires significant adaptation of classical orbit propagation algorithms. This work presents a parallel orbit propagator suite for the Class-1 space-qualified GPU@SAT, which offers only 32-bit integer arithmetic. Three parallel integrators — Runge–Kutta 4, Dormand–Prince 8, and Picard–Chebyshev — are adapted to the board using software-emulated 32-bit posit, adimensionalized dynamics, round-off compensation, and force models restricted to elementary operations. This co-design reduces the propagation error by six orders of magnitude with respect to a naive fixed-point implementation. Validated against a high-fidelity reference over 1000 dispersed LEO orbits, the suite achieves position errors around one hundred metres, with different integrators offering distinct accuracy–runtime trade-offs. Deployment on GPU@SAT becomes competitive with state-of-the-art space-grade processors as the number of trajectories grows; the posit emulation, instead, increases runtime fivefold, yet enables accurate propagation. These results show that accurate ensemble propagation is feasible on low-precision space hardware, enabling future onboard applications. A Monte Carlo uncertainty-propagation scenario is proposed as a demonstration.

1 Introduction

Propagating large ensembles of trajectories is a recurring need across a wide range of astrodynamics applications, from uncertainty quantification and space situational awareness (SSA) to guidance, navigation, and control (GNC) and mission planning. Tasks as diverse as Monte Carlo uncertainty propagation, reachability analysis for maneuver planning, conjunction assessment, and large space-object catalogs propagation share a common trait: they do not rely on a single orbit, but require many trajectories to be propagated, frequently over several orbital revolutions.

Owing to their computational cost, batch propagation is traditionally carried out on ground, where commercial-grade computing resources, including GPUs, are readily available. Ground-based operations, however, introduce latencies of hours to days between a request and its execution, and scale poorly as the number of resident space objects and orbiting assets keep increasing. Such delays are especially limiting in time-critical settings: for instance, a reactive Earth-observation satellite that must re-plan an acquisition in response to a sudden request, or the planning of a collision avoidance maneuver. At the same time, the growing traffic threatens to saturate the existing ground infrastructure.

© *Politecnico di Milano, IngeniArs*

A natural way to overcome these limitations is to move computation to the edge, letting the spacecraft propagate trajectories directly onboard. Orbit propagation is particularly well suited to this shift, as an ensemble of trajectories is a set of independent integration problems that naturally map to the massively parallel architecture of a GPU, which can propagate thousands of trajectories in the time a CPU would take for a single one [1]. Commercial GPUs, however, are unsuitable for institutional and safety-critical missions, and only recent advances in radiation-tolerant electronics have made space-graded onboard GPUs a viable option for in-orbit parallel propagation. These space-qualified, Class-1 platforms still lag well behind their terrestrial counterparts in throughput, memory bandwidth and, most critically, supported numerical formats. As a consequence, classical orbit-propagation routines cannot be ported as they are: they demand a substantial reformulation to comply with the constraints of the target hardware.

This paper tackles this challenge by presenting the design, implementation, and validation of a parallel orbital propagator suite tailored for onboard execution on a low-precision space-graded GPU. The considered application scenario is LEO propagation, which is challenging enough due to the variety of orbital perturbations involved. In turn, the following issues are addressed: the selection of parallelization-friendly integration schemes, the adaptation of numerical arithmetic to available formats, and the simplification of force models.

The reference platform for this work is GPU@SAT, a Class-1 space-qualified soft-GPU developed by IngeniArs [2]. Being a soft-GPU, its architecture is not fixed in hardware but obtained by configuring the logic of an FPGA, and it is conveniently operated through an OpenCL-based framework. The board executes up to 512 threads in parallel, and it supports integer (fixed-point) arithmetic, with no floating-point unit; it does not provide trigonometric or other transcendental functions, and the local memory available to each thread is limited. These characteristics, rather than raw performance, are what set space-graded GPUs apart from ground platforms and drive the numerical and modeling choices central to this work: number format, error compensation, force-model reformulation, and memory-aware kernel design.

The remainder of the paper follows the reasoning that led to the final propagator. Section 2 reviews the sources of numerical integration error and the families of integration schemes, retaining those best suited to parallel execution. Section 3 presents a preliminary propagator design with the dynamical model and the force-model simplifications imposed by the hardware; ultimately, it shows through a first validation that a direct transposition to the low-precision arithmetic of GPU@SAT is not accurate enough for orbital propagation. Section 4 introduces the two techniques that overcome this limitation, namely round-off error compensation and the posit number format, and quantifies the resulting accuracy gain. Section 5 describes the deployment of the propagator on GPU@SAT and profiles its onboard execution time, and Section 6 demonstrates the suite in a Monte Carlo uncertainty-propagation scenario. Finally, Section 7 summarizes the findings and lessons learned and outlines possible improvements.

2 Orbital propagation on GPU

The propagator design and implementation performed in this work are influenced by three main factors: numerical integration error, types of integration schemes, and GPU architecture.

Numerical integration is affected by several error types, each impacting the final accuracy for different reasons:

- **Truncation errors** represent the difference between the numerical solution computed by the integration scheme and the exact solution of continuous dynamics. They reflect the error introduced by propagating the system using finite time steps.
- **Round-off errors** arise from the finite-precision arithmetic of digital computers. Their magnitude is typically on the order of machine precision.
- **Modeling errors** are generated by the simplifications and assumptions adopted in the dynamical model with respect to the real physical system.

Numerical integration methods can be broadly divided into two main families: methods for solving Initial Value Problems (IVP) and collocation methods.
Among IVP methods, integration schemes can be further classified according to several high-level characteristics:

- **Order**: a numerical integrator is said to be of order p if its local truncation error is proportional to h^{p+1}, where h is the integration step size.
- **Single-step / Multi-step**: single-step methods use only the previous state to compute the next one, whereas multi-step methods exploit information from multiple previously computed states.

- **Fixed-step / Adaptive-step**: fixed-step methods employ a constant step size throughout the integration, while adaptive-step methods adjust the step size based on an estimate of the truncation error.
- **Explicit / Implicit**: in explicit methods, the next state is obtained directly as a function of previously known states. Implicit methods, instead, require the solution of a nonlinear algebraic system at each integration step.
- **First-order / Second-order**: first-order integrators solve systems of first-order ordinary differential equations, whereas second-order integrators can directly handle second-order dynamical systems.

Collocation methods approximate the system state using polynomial functions, enforcing the governing differential equations at a discrete set of points, called collocation points. Among the methods belonging to this class, the Picard–Chebyshev approach has gained significant popularity in the field of GPU-based parallel orbit propagation. This method employs Chebyshev polynomials to obtain a global approximation of the trajectory over large time intervals, starting from an initial guess trajectory and iteratively refining the solution. The classifications presented before will be discussed in the next section to identify state-of-the-art integration schemes that are most suitable for this application.

Lastly, it is important to note that GPU programming differs substantially from the classical CPU paradigm. A CPU devotes most of its resources to control logic and cache in order to execute one or a few sequential threads at low latency; a GPU instead trades this flexibility for throughput, packing a large number of simpler processing units that operate under a Single Instruction, Multiple Data model (SIMD), in which groups of units execute the same instruction in lockstep on different data [1]. The violation of this model causes thread divergence, in which multiple threads are executed sequentially, due to the lower number of control units. The execution is orchestrated by a host processor, which moves the input data to the device, launches the parallel routines, and gathers the results, while the on-device memory is arranged in a hierarchy that, unlike on a CPU, the programmer must manage explicitly to keep frequently accessed data in the fast but small local memory rather than in the large, high-latency global one.

These traits have direct consequences on orbital propagation. Since the processing units advance in lockstep, any data-dependent branching forces the diverging execution paths to be taken sequentially, eroding the parallel speed-up; the propagation must therefore be arranged so that every trajectory performs the same sequence of operations, which discourages adaptive step-size control, iterative stopping criteria, and conditional force evaluations. Moreover, the limited fast memory available to each thread restricts how much state and how many intermediate quantities a trajectory can retain, favoring compact force models and regular memory-access patterns. These principles guide the propagator design presented in the following section, even though the target platform departs from the architecture of mainstream GPUs.

2.1 Integrators trade-off

A starting point for integrator selection was the study by Atallah et al. [3], which compares different state-of-the-art integrators for the propagation of perturbed satellite orbits around Earth. Specifically, the Round-Trip Closure error (RTC) of several integration schemes is analyzed under a 70×70 gravity field and atmospheric drag, highlighting the accuracy–efficiency trade-off. The best-performing methods are Adaptive Picard–Chebyshev [4], Gauss–Jackson 8 (a fixed-step, explicit, multi-step, second-order integrator), and Dormand–Prince 8(7) (an explicit, single-step Runge–Kutta integrator with adaptive step-size control).

Intuitively, the best integrators are high-order schemes that do not suffer from truncation error, enabling longer integration steps, albeit at the cost of more function evaluations per step. However, in GPU-based propagation the SIMD model demands each integration step to perform the same sequence of operations regardless of the selected initial conditions. For this reason, no adaptive-step or adaptive-order routines are allowed, and implicit integration schemes are discarded. Consequently, a fixed step size must be adopted for all IVP integrators. For the same reason, the state-of-the-art Adaptive Picard–Chebyshev scheme [4] is constrained to use fixed user-selected numbers for orbit segmentation and collocation grid points per segment. Still, due to its growing adoption and natural suitability for parallel implementation, the Picard–Chebyshev method is still taken into consideration [1, 5].

Based on these considerations, the selected integrators for the implementation are reported in Table 1. Alongside Dormand–Prince 8 and Picard–Chebyshev, the lower-order Runge–Kutta 4 is deliberately added to the pool: DP8 requires twelve stages per step against the four of RK4, and its higher order pays off only when the arithmetic is precise enough to exploit the additional coefficients — a condition not guaranteed in low precision, where

every extra stage is a further source of round-off. RK4 is therefore retained as a simpler, round-off-parsimonious alternative to DP8. Despite its favorable accuracy–efficiency balance, GJ8 (and multi-step schemes in general) has been discarded for three reasons. First, it needs a single-step start-up procedure to build its initial back-points; second, advancing the state requires storing and repeatedly accessing several past states (rather than only the last one as in single-step schemes), increasing the per-thread memory footprint; last, the dependency on several back-points exposes the integration to round-off accumulation in low precision. In this regard, the integrator analysis performed so far assumed to work in a double-precision format; as this is not possible on the GPU@SAT, most of the efforts of this work focused on the numerical techniques described in the next section.

Method	Type	Explicit/Implicit	First/Second-order	Adaptation
RK4, DP8	Single-step	Explicit	First-order	Fixed step size
PC	Collocation	Explicit, iterative	Second-order	Non-adaptive, segmentation only

Table 1 *Numerical integrators implemented for orbit propagation.*

3 Preliminary propagator design

3.1 Dynamical model and simplification

The dynamical model implementation in the propagator has deep impact on numerical precision, modeling accuracy, and deployment feasibility on the target hardware. Therefore, the force models have been selected and tailored to strike a balance between accuracy and performance, with the constraint of avoiding transcendental and trigonometric operations.

The dynamics are formulated as a perturbed two-body problem using Cowell's method, summing the central gravitational term to the perturbation term [6]. All throughout the formulation, the dynamics are dimensionless with respect to a reference length of around 8000 km, coherent with the semi-major axis of LEO with substantial eccentricity, and the time unit enforces a unitary dimensionless gravitational parameter. It is worth noting that the loss of dimensionality does not remove the several orders of magnitude separating the central gravitational term from the perturbations; rather, it brings the resulting total acceleration $\ddot{\boldsymbol{r}}$ close to unity, i.e. into the range where the posit format is most precise. Most importantly, this allows the integration steps to be carried out in the optimal numerical range.

A regularized formulation, such as the Kustaanheimo–Stiefel one, was taken into consideration but ultimately not adopted: although it linearizes two-body dynamics and tolerates larger steps, it propagates in a fictitious time and would require a per-orbit event-detection stop to precisely reach the prescribed physical epoch, reintroducing thread divergence. Moreover, the conversion from and towards Cartesian coordinates would increase the computational expense.

Even though the most relevant perturbations in LEO are Earth's non-spherical gravity field and atmospheric drag, all main perturbing forces are included in the model in order to enable orbital prediction in a wider range of dynamical regimes. In particular, the following perturbations are considered: spherical harmonics (SH) perturbations up to degree and order 10; Sun and Moon third-body gravitational attractions; solar radiation pressure (SRP); atmospheric drag.

The computation of SRP and third-body perturbation models requires Sun and Moon ephemerides, while spherical harmonics require rotation matrices from ECI to the Earth-Centered Earth-Fixed (ECEF) frame. These quantities are sampled on the integration grid before propagation from the Orekit library, reflecting a realistic flight scenario in which such information would be provided from onboard data. However, the IVP integration schemes (of Runge-Kutta type) require these quantities not only at the nominal integration steps t_n, but also at multiple intermediate points within the interval $[t_n, t_{n+1}]$ for right-hand side function evaluations. Therefore, in the current implementation, ECI-ECEF rotation matrices and ephemerides are assumed to remain constant throughout the integration step. This introduces modeling errors that scale up as the time step increases. Still, this kind of error is negligible compared to round-off errors in low precision. Future work could devise interpolation strategies compatible with the arithmetic operations supported on the GPU@SAT.

Spherical Harmonics Spherical harmonics are used to model the non-spherical component of the Earth's gravitational field. To deploy the propagator on GPU@SAT, which does not support trigonometric functions, the formulation proposed by [7] is adopted. This approach avoids the direct evaluation of sine and cosine functions by exploiting their addition properties and recursively computing the harmonic terms together with the associated Legendre polynomials. The gravitational potential is expressed in spherical coordinates as

$$U(r,\phi,\lambda) = \frac{\mu_e}{R_e}\sum_{n=0}^{\infty}\sum_{m=0}^{n}\left(C_{nm}V_{nm} + S_{nm}W_{nm}\right). \tag{1}$$

where C_{nm} and S_{nm} are the Earth's spherical harmonics coefficients, μ_e and R_e are the Earth gravitational parameter and mean radius, respectively, and

$$V_{nm} = \left(\frac{R_e}{r}\right)^{n+1} P_{nm}(\sin\phi)\cos(m\lambda), \qquad W_{nm} = \left(\frac{R_e}{r}\right)^{n+1} P_{nm}(\sin\phi)\sin(m\lambda). \tag{2}$$

where P_{nm} denotes the associated Legendre polynomial of degree n and order m.

The recursion is initialized exploiting $\cos\lambda = \frac{x}{r}$, $\sin\lambda = \frac{y}{r}$, $\sin\phi = \frac{z}{r}$, and is then given by

$$V_{m,m} = (2m-1)\left(\frac{R_e x}{r^2}V_{m-1,m-1} - \frac{R_e y}{r^2}W_{m-1,m-1}\right), \quad V_{n,m} = \frac{2n-1}{n-m}\frac{R_e z}{r^2}V_{n-1,m} - \frac{n+m-1}{n-m}\left(\frac{R_e}{r}\right)^2 V_{n-2,m}, \tag{3}$$

$$W_{m,m} = (2m-1)\left(\frac{R_e x}{r^2}W_{m-1,m-1} + \frac{R_e y}{r^2}V_{m-1,m-1}\right), \quad W_{n,m} = \frac{2n-1}{n-m}\frac{R_e z}{r^2}W_{n-1,m} - \frac{n+m-1}{n-m}\left(\frac{R_e}{r}\right)^2 W_{n-2,m}. \tag{4}$$

Beyond spherical harmonics, a mascon (mass-concentration) representation [8] was also considered, as it shares the same advantage of avoiding trigonometric operations. However, it was ultimately set aside because reproducing a gravity field of equivalent degree and order requires summing the contributions of a large number of point masses. This makes it not only computationally less efficient but also more sensitive to round-off.

Third-body perturbation Third-body perturbations account for the gravitational attraction of the Sun and the Moon. The acceleration induced by each perturbing body is modeled as in [7]:

$$\ddot{\boldsymbol{r}}_b = \mu_b\left(\frac{\boldsymbol{s}-\boldsymbol{r}}{\|\boldsymbol{s}-\boldsymbol{r}\|^3} - \frac{\boldsymbol{s}}{\|\boldsymbol{s}\|^3}\right) \tag{5}$$

where $\boldsymbol{s}$ is the geocentric position vector of body b.

Solar radiation pressure SRP is modeled assuming isotropic reflection:

$$\ddot{\boldsymbol{r}}_{\text{SRP}} = -\frac{P_{1\,\text{AU}}}{r_{\text{sun}}^2}\,c_r\,\frac{A}{m}\,\frac{\boldsymbol{r}_{\text{sun}}}{r_{\text{sun}}} \tag{6}$$

The vector $\boldsymbol{r}_{\text{sun}}$ denotes the position of the Sun relative to the spacecraft and is approximated by the Earth–Sun vector. To model eclipses, a simple cylindrical shadow model is adopted [9]. This choice avoids trigonometric operations and is therefore compatible with the computational constraints of the onboard hardware, with minimal accuracy losses for this application.

Atmospheric Drag Atmospheric drag is modeled using a simple isotropic formulation:

$$\ddot{\boldsymbol{r}}_{\text{drag}} = -\frac{1}{2}\,c_d\,\frac{A}{m}\,\rho\,v_r^2\,\hat{\boldsymbol{v}}_r \tag{7}$$

where $\boldsymbol{v}_r = \boldsymbol{v} - \boldsymbol{\omega}_E \times \boldsymbol{r}$ is the velocity relative to the atmosphere. Normally, the atmospheric density is obtained through exponential interpolation of tabulated values according to the Harris-Priester model for mean solar activity [7]; in this case, linear interpolation is adopted to avoid transcendental operations. Moreover, the diurnal bulge is neglected.

3.2 Numerical Validation and Low-Precision Performance

To validate the performance of the propagator, a ground-truth simulation environment is adopted. The open-source C&S toolkit Orekit is used as the reference propagator due to its high fidelity and flexibility in modeling different perturbing forces. The validation is performed by sampling a set of initial conditions in the LEO environment, propagating each case with both Orekit and the implemented propagator, and comparing the resulting absolute and relative errors with respect to the reference solution. Force models in Orekit have been set to match the formulations of the implemented propagator (Sec. 3.1), minimizing modeling discrepancies and highlighting other error sources.

The orbit dataset is generated by uniformly sampling 1000 different initial conditions in the Keplerian orbital elements, with perigee altitude between 400 and 800 km and eccentricity between 0 and 0.3, while inclination, RAAN, argument of perigee, and initial true anomaly are sampled uniformly over their full range.

The propagation horizon is equal for all orbits and set to 5 times the average orbital period of the dataset. Relative errors between Orekit and our implemented propagator are reported below for three numerical formats: IEEE double (64 bit) floats, IEEE single (32 bit) floats, and Q16.16 fixed-point quantization. The Qm.n format represents floating point numbers using standard integers, and allocates m bits to the integer part and n bits to the fractional part. Qm.n is a standard method to operate floating point numbers on integer-only systems, such as the GPU@SAT, therefore it is considered for a first deployment attempt.

Results in double precision

To validate the integrator and force implementation, a first test is performed in double precision, shown in Fig. 1.

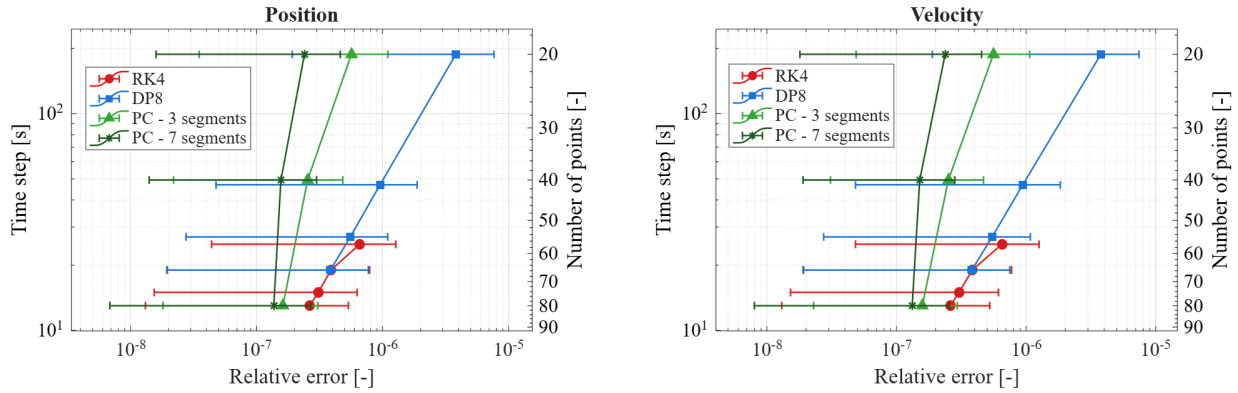

Figure 1 *Relative final position and velocity errors in double precision floating-point.*

The error magnitude exhibits the same trend for all integrators: it decreases as the integration step size is reduced, eventually reaching a minimum value on the order of 10^{-7} for both position and velocity. This behavior indicates that truncation error is the dominant source of inaccuracy at larger step sizes. As the step size decreases, the truncation error is progressively reduced until the total error becomes limited by modeling approximations, such as the coarse retrieval of ephemerides and rotation matrices as well as the cylindrical eclipse model adopted for the SRP computation described in Sec. 3.1. Nevertheless, this level of modeling error is considered acceptable, as it becomes negligible compared to the numerical errors introduced by lower-precision formats.

Results in single precision - no compensation

As a preliminary demonstration of the impact of round-off error in the orbital propagation, the integrators are tested in single precision and shown in Fig. 2. The substantially worse performance with respect to double precision can be immediately noticed, as errors are orders of magnitude larger. The main source of error becomes round-off error rather than truncation error, as can be observed from the behavior of the integrators for different time steps (or, equivalently, different numbers of grid points). For the PC method, the error remains nearly constant as the grid is refined, whereas for RK4 it even increases when smaller time steps are used. In general,

these results suggest that a lower number of repeated arithmetic operations reduces the accumulation of round-off errors.

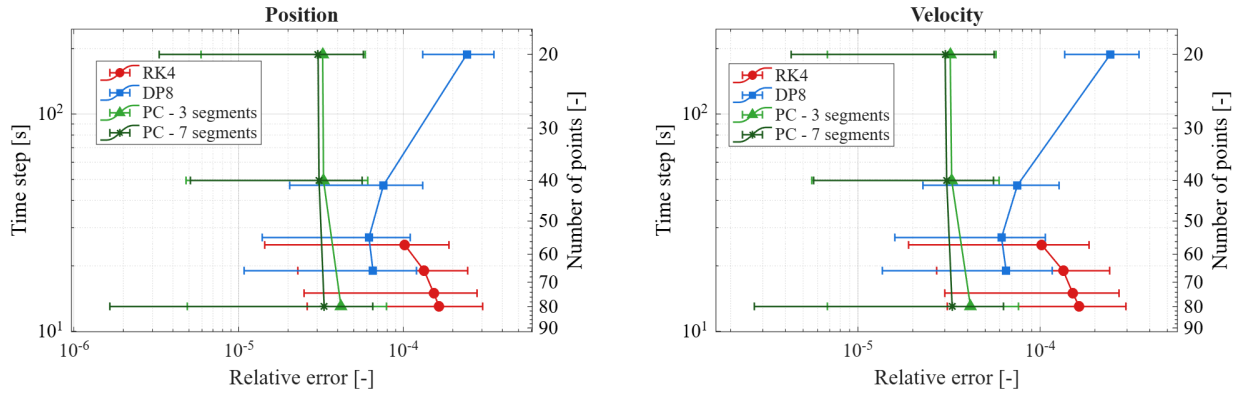

Figure 2 *Relative final position and velocity errors in single precision floating-point.*

Results in Q16.16 fixed point quantization

Preliminary results for Q-Format quantization show that each integrator reaches unacceptably large errors; for instance, RK4 with a 25-second step reaches mean relative errors of 3.2×10^{1} for both position and velocity, proving that classical quantization methods cannot be naively applied for the purpose of this work.

4 Round-off Mitigation and Improved Numerical Arithmetic

Among the three error sources introduced in Sec. 2, round-off is the one that depends directly on the underlying number representation and arithmetic. In conventional ground-based astrodynamics, double-precision floating-point arithmetic keeps it well below the truncation and modeling contributions, so that it can generally be neglected. As anticipated, however, GPU@SAT can only handle a bit-width up to 32 bits; round-off therefore becomes the dominant error source, as shown by the preliminary results in Sec. 3.2. The single precision format confirms a marked accuracy degradation with respect to double precision, consistent with round-off becoming dominant. The loss is most severe for fixed-point, reaching unfeasible levels of relative propagation errors (>10). This outcome calls for two complementary strategies, pursued in the remainder of this subsection: limiting the accumulation of round-off through error-compensation techniques, and replacing the unsuitable fixed-point representation with a more favorable 32-bit number format.

4.1 Error compensation techniques

The first strategy limits the accumulation of round-off over the many additions performed during propagation, independently of the number format adopted. Two complementary techniques are considered: compensated summation and compensated time integration [10]. Compensated summation (Kahan summation) reduces round-off errors in repeated additions by maintaining an auxiliary compensation term, making the overall error largely independent of the number of summed elements. The implementation used here relies on the 2Sum algorithm [11, 12], which computes the exact rounding error of a floating-point addition, thereby increasing numerical accuracy. Compensated time integration, following Klöwer et al. [10], extends this idea to sequential single-step integration. A compensation term is propagated across time steps to account for the error introduced when updating the state from x_k to $x_{k+1} = x_k + dx$, thus preventing the progressive accumulation of round-off errors. In the present propagator, compensated summation is applied to the weighted combination of stages of the Runge–Kutta schemes and to the matrix products of the Picard–Chebyshev iteration, whereas compensated time integration handles the state update of the single-step integrators. The corresponding routines are reported in Alg. 1 to Alg. 3.

Algorithm 1
CompSum($\boldsymbol{v}$): Compensated (Kahan) summation

Require: input vector $\boldsymbol{v} = (v_1, ..., v_n)$
$S \leftarrow 0, \quad C \leftarrow 0$
for i in $1, ..., n$ **do**
  $y \leftarrow v_i + C$
  $(S, C) \leftarrow \text{2Sum}(S, y)$
end for
return S

Algorithm 2
2Sum(a, b): Error-free transformation of a sum

Require: a, b
$S \leftarrow a + b$
$a' \leftarrow S - b$
$b' \leftarrow S - a'$
$\varepsilon \leftarrow (a - a') + (b - b')$
return (S, ε)

Algorithm 3: CompTime($\boldsymbol{x}_0$): Compensated time integration

Require: initial state $\boldsymbol{x}_0$
$\boldsymbol{x} \leftarrow \boldsymbol{x}_0, \quad \boldsymbol{C}_{\text{in}} \leftarrow 0$
for t_k in time grid **do**
  $\boldsymbol{\delta} \leftarrow$ weighted sum of stages
  $\boldsymbol{\delta} \leftarrow \boldsymbol{\delta} - \boldsymbol{C}_{\text{in}}$
  $\boldsymbol{x}_{\text{next}} \leftarrow \boldsymbol{x} + \boldsymbol{\delta}$
  $\boldsymbol{C}_{\text{out}} \leftarrow (\boldsymbol{x}_{\text{next}} - \boldsymbol{x}) - \boldsymbol{\delta}$
  $\boldsymbol{x} \leftarrow \boldsymbol{x}_{\text{next}}$
end for
return $\boldsymbol{x}, \boldsymbol{C}_{\text{out}}$

4.2 The Posit Number Format

The second strategy concerns the representation itself. Since the fixed-point format proved unsuitable, a more favourable one is adopted: the posit number format [13]. A posit with m total bits and n exponent bits, denoted Posit m, n, has its bit string illustrated in Fig. 3. The posit representation consists of four fields:

- a sign bit s;
- a regime field of variable length;
- n exponent bits of fixed length, encoding the integer e;
- a fraction (mantissa) f, of variable length.

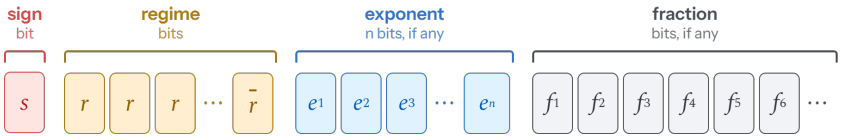

Figure 3 *Bit string composition of a Posit number [13].*

Unlike IEEE floats, which reserve a fixed-width exponent and therefore keep a constant relative precision across their whole range, posits encode the magnitude of a number through the variable-length regime *together with* the n exponent bits. A positive posit p is decoded as [13, 14]:

$$p = \beta^k \, 2^e \, (1 + f), \qquad \beta = 2^{2^n}, \tag{8}$$

where k is the signed regime run length, e the integer encoded by the exponent bits, and f the fraction. The regime is a run-length-encoded field, meaning that its value is determined by the number of consecutive identical bits rather than by a fixed-width exponent field. The regime sets a coarse scale in powers of *useed*, which the exponent then refines, so that numbers close to unity need only a few regime bits. The bits thus freed are reassigned to the fraction, giving posits a *tapered* precision: a higher relative accuracy around unity, gradually traded for dynamic range towards very large or very small magnitudes.

These advantages make posits the format of choice for the propagator developed in this project: the dimensionless equations of motions, which operate on a unitary state, work in synergy with this number format, as detailed

in Section 3.1. Ultimately, posits yield superior numerical precision than standard IEEE floats for the same bit length. The precision of posit formats can be assessed by analyzing their decimal accuracy, defined in Eq. 9.

$$\text{decimal accuracy} = -\log_{10}\left(\left|\log_{10}\left(\frac{x_{\text{true}}}{x_{\text{posit}}}\right)\right|\right). \tag{9}$$

Two posit configurations are considered for this application: Posit 32,0 and Posit 32,3. Their decimal accuracy is compared with that of the single-precision floating-point format, using double-precision floating-point arithmetic as the reference (ground truth). The resulting accuracy profiles over the representable numerical range are shown in Fig. 4.

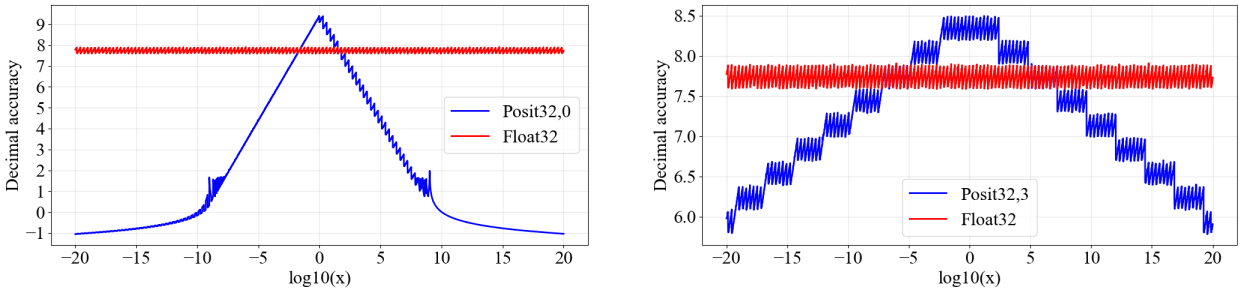

Figure 4 *Decimal accuracy of Posit 32,0 (left) and Posit 32,3 (right) compared with IEEE single-precision floating-point.*

Both posit formats achieve higher precision than single-precision floating-point arithmetic in the vicinity of unity. Posit 32,0 exhibits a higher peak accuracy around 1, but its precision decreases more rapidly as the magnitude of the represented number moves away from unity as no bit is dedicated to the exponent. In contrast, Posit 32,3 attains a lower peak accuracy but maintains an elevated level of precision across a wider dynamic range, thanks to the three exponent bits. This wider range is decisive for orbit propagation, where the small perturbing accelerations must still be resolved accurately alongside the dominant central-gravity term. For this reason, it is anticipated that Posit 32,3 is the adopted reference format for the propagator.

The posit format is implemented as a software emulation, which conveniently uses an integer representation for GPU@SAT compatibility. The emulated operators (the four basic operations and the square root) were validated in precision against IEEE single precision, using double precision as ground truth. It should be stressed that, on the current GPU@SAT hardware, any floating-point format can only be software-emulated.

4.3 Accuracy Improvements and Posit-Single comparison

Results in single precision with compensation

The proposed compensation techniques are applied to the single-precision format, and the resulting errors are shown in Fig. 5. The impact of the compensation techniques varies depending on the integrator. The RK4 method achieves the most significant improvement, reducing the propagation error by more than one order of magnitude and reaching the best overall accuracy among the considered integrators. The PC method exhibits a smaller but consistent improvement, with similar benefits observed for both the 3 and 7-segment configurations. Conversely, the DP8 integrator shows only marginal improvements, with errors remaining close to those obtained without compensation. This phenomenon is attributable to the number of steps and matrix operations involved, with DP8 gaining only a marginal benefit from the time compensation technique due to its coarser integration grid.

Results in Posit

Both Posit 32,0 and Posit 32,3 number formats are tested with the selected integrators. However, the Posit 32,0 configuration exhibits very poor performance, with integration errors largely exceeding acceptable levels. This behavior is mainly due to the poor precision with which this format represents values significantly larger or smaller than 1. In particular, these limitations become critical in the evaluation of spherical harmonics, with coefficients

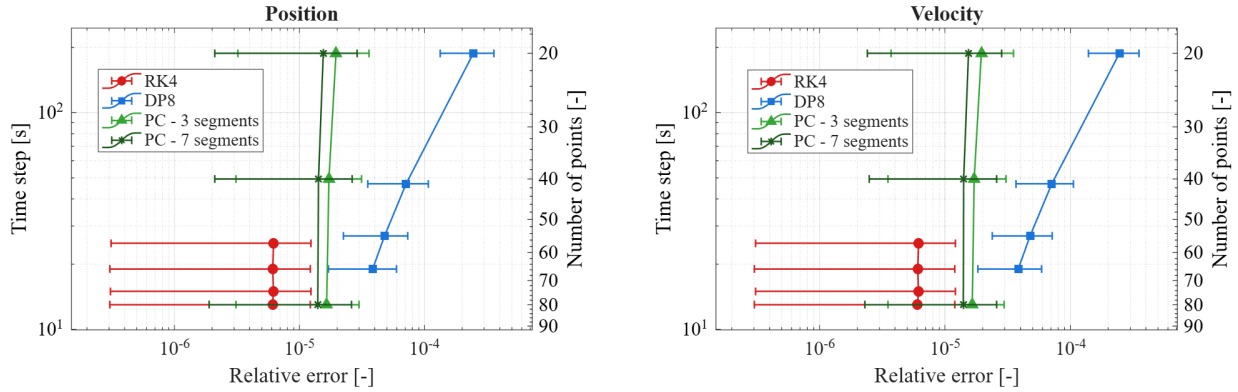

Figure 5 *Relative final position and velocity errors in single precision with numerical compensations.*

of typically very small magnitude. Under these conditions, Posit 32,0 introduces significant distortions, ultimately compromising the entire integration process. For this reason, the results in Fig. 6 report only the Posit 32,3 format.

The Posit 32,3 format generally exhibits performance comparable to single precision in order-of-magnitude. For the RK4 integrator, the resulting error is close to that obtained with single precision, albeit 2-3 times higher, and remains approximately constant across different integration step sizes. Operating on a broader grid of integration steps, DP8 shows instead a marked dependency on the step size. Similarly to RK4, the PC integrator shows marginally worse performance with posits, and the error increases with the number of segments per orbit while being insensitive to the number of points per segment. This behavior can be attributed to the accumulation and propagation of round-off errors between consecutive segments. With fewer segments, the error is propagated a smaller number of times through the initial conditions of subsequent segments, whereas a higher number of segments leads to a progressive amplification of the error. The different precision of RK4 and PC may be reasonably explained by the absence of fused multiply-add operations in the posit implementation, while the standard IEEE 32-bit float features such low round-off operations. Especially for PC, a lower round-off error in multiply-add operation sequences heavily matters for matrix multiplications.

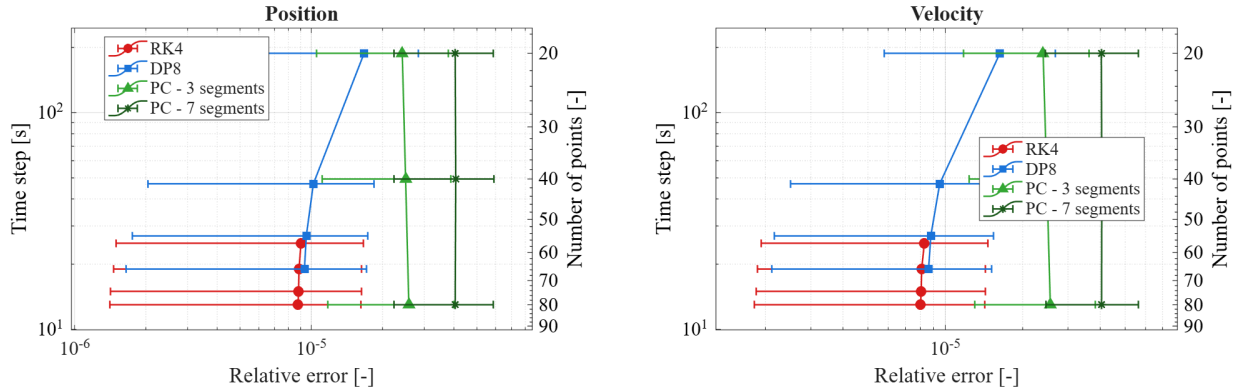

Figure 6 *Relative final position and velocity errors in Posit 32,3 with numerical compensations.*

Conversely, the DP8 integrator benefits significantly from the use of the Posit format. This is allegedly because the increased accuracy around unity can handle better the higher number of Runge-Kutta stages of this integrator which, due to non-dimensional dynamics, feature quantities around 1. In the other integration schemes, the accuracy tradeoff of posits in lower dynamic ranges presumably penalizes the force models computation (especially SH), favoring single-precision numbers.

In all, the results show that compensation techniques are effective in increasing accuracy, especially the compensated time integration applied to IVP integrators; at the same time the adoption of the Posit format provides a significant improvement for the DP8 integrator. The PC method instead exhibits only a marginal improvement with Kahan compensation, whereas the use of the Posit number format slightly degrades its

performance. Dedicated matrix-multiplication compensation techniques may improve PC performance.

Due to its simplicity and performance in low-precision, the RK4 integrator was selected for deployment on GPU@SAT, as detailed in the next section.

5 Deployment framework

As the analyses in Sec. 4.3 suggest, RK4 offers remarkable perforamance with the posit 32,3 format and largely benefits from compensation techniques. However, most relevantly, RK4 is the ideal candidate for a first deployment on the GPU@SAT due to its small code and memory footprint with respect to the other integrators.

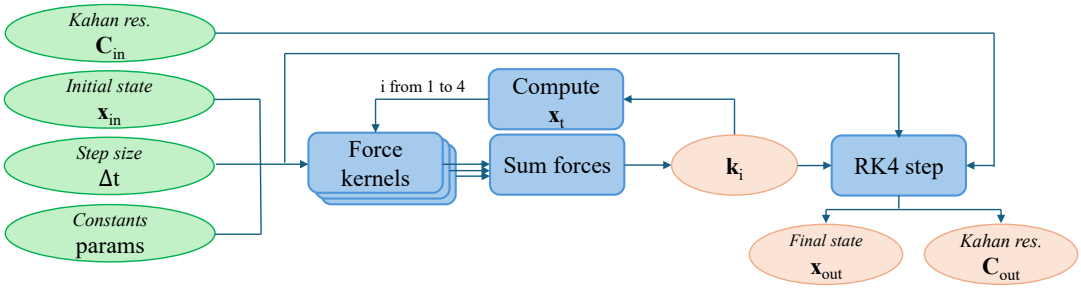

Figure 7 *Diagram of the Runge-Kutta 4 step implemented for GPU@SAT, with each blue box being a separate kernel.*

The GPU@SAT is a Class-1-grade space qualified board developed by IngeniArs, providing a soft-GPU OpenCL-based framework that enables parallel computing for space applications [2]. To comply with this environment and at the same time enable fast prototyping, we developed a custom cross-platform OpenCL interface library [15], allowing the deployment of single kernels and neural networks (also converted from the Open Neural Network Exchange (ONNX) format*) with seamless compatibility with most OpenCL drivers and a specific implementation for GPU@SAT. The propagator is written as a recursive network where each iteration of the recursion represents a single integration step. In turn, each step is divided into multiple kernels instead of using a single one, mainly for memory constraints of the GPU@SAT, as a single kernel would require a much bigger space for instructions and local variables. A diagram is shown in Fig. 7. The integrator implemented is Runge-Kutta-4, hence, the forces are evaluated four times following the algorithm and added together in a separate kernel. Finally, a kernel performs the actual Runge-Kutta-4 step, including compensated sums (see Section 4.1). The following kernels are implemented (Fig. 7):

- "Force kernels": sequence of force kernels that compute their respective related acceleration given the state, time and parameters, one for each environmental force (central gravity, spherical harmonics, atmospheric drag, solar radiation pressure, third body).
- "Sum forces": compensated summation of all the computed accelerations (Alg. 1), producing the intermediate RK4 acceleration term $\boldsymbol{k}_i$, $i = 1, ..., 4$
- "Compute $\boldsymbol{x}_t$": computes the input state for the following Runge-Kutta-4 evaluation as:

$$\boldsymbol{x}_t = \boldsymbol{x}_{in} + r_i \boldsymbol{k}_{i-1} \Delta t \tag{10}$$

  for $i = 2, 3, 4$, where r_i is the Runge-Kutta coefficient: $r_2 = r_3 = \frac{1}{2}, r_4 = 1$
- "RK4 step": computes the integrated state by weighted summation of RK4 stages, using compensated sums (Alg. 1) and compensated time integration (Alg. 3):

$$\begin{cases} \boldsymbol{\delta} = \frac{1}{6}\text{CompSum}\left(\boldsymbol{k}_1, 2\boldsymbol{k}_2, 2\boldsymbol{k}_3, \boldsymbol{k}_4\right) - \boldsymbol{C}_{\text{in}} \\ \boldsymbol{x}_{out} = \boldsymbol{x}_{in} + \boldsymbol{\delta} \\ \boldsymbol{C}_{\text{out}} = \left(\boldsymbol{x}_{out} - \boldsymbol{x}_{in}\right) - \boldsymbol{\delta} \end{cases} \tag{11}$$

*`https://onnx.ai/`

The posit arithmetic is written in a custom soft implementation that emulates all needed operations such as addition, subtraction, multiplication, division, and squared root. All parameters required for propagation are saved in the "__constant" address space and given as input. Finally, all the code-base is written with branchless logic to maximize parallelism and avoid thread divergence. Unfortunately, GPU@SAT has limited local memory for each thread, preventing the storage of V and W coefficients (Eq. 4) from the spherical harmonics recursive computation. Therefore, the terms in question must be allocated in global memory as read/write buffers, severely penalizing performance, as discussed in Section 5.1.

5.1 Profiling Results on GPU@SAT

The propagator is profiled by measuring the mean runtime of a single integration step for the whole ensemble, as a function of the number of propagated trajectories. GPU@SAT is compared against two space-representative processors, the GR740 [16] and the GR712RC [17], accessed through the remote execution facility for flight software of the ESA/ESTEC Flight Software Laboratory. A terrestrial CPU (AMD Ryzen 5 3500U) and an embedded GPU (NVIDIA Jetson Nano [18], programmed through the PoCL OpenCL driver [19]) serve as additional references. Ideally, the CPU runtime should grow linearly with the ensemble size, whereas GPU@SAT should remain nearly constant up to its 512-thread limit. Two hardware limitations, however, shape the measured behaviour.

The first concerns the spherical-harmonics model. The GPU@SAT per-step time grows with the ensemble size because, as anticipated in Section 5, the limited per-thread local memory forces the V and W recursion coefficients (Eq. 4) into global memory. Hence, the resulting read/write traffic dominates the runtime. Figure 8 isolates the effect by comparing the same Q8.8 propagation with and without spherical harmonics: without them the runtime stays essentially flat, whereas with a degree-and-order-10 field it grows steeply with the number of trajectories. To assess the intrinsic parallel scaling of the board, the platform comparison below is therefore carried out on the spherical-harmonics-free configuration.

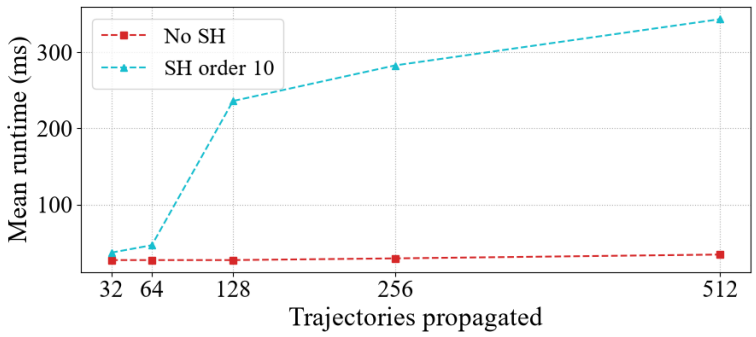

Figure 8 *Step mean runtime with and without spherical harmonics for the Q8.8 format on GPU@SAT.*

The second limitation concerns the number format: at the current stage only integer arithmetic is available in hardware on GPU@SAT, while the software Posit 32,3 unit is still under development. The on-board posit runtime is therefore estimated by scaling the measured Q8.8 time by the posit-to-Q8.8 runtime ratio obtained on the Jetson Nano, whose embedded OpenCL GPU is architecturally closer to GPU@SAT than a desktop CPU. Table 2 reports the resulting single-step times, at 512 trajectories, for the three numerical formats across all platforms. The software emulation of posits is one to two orders of magnitude slower than fixed-point arithmetic, so the estimated on-board posit time (188.85 ms) is dominated by emulation overhead; a dedicated hardware posit unit is expected to bring it back close to the Q8.8 figure.

Figure 9 then compares GPU@SAT (Q8.8) against the GR740 and GR712RC running single-precision floating point, on the spherical-harmonics-free configuration. Single precision is the fairest reference, as it shares the 32-bit word length of Posit 32,3. As expected, the two CPUs are faster for small ensembles, where the parallel overhead of the board is not amortized; as the ensemble grows, however, the near-constant runtime of GPU@SAT overtakes the GR712RC between 256 and 512 trajectories, demonstrating the benefit of parallel execution. The GR740 remains the fastest platform over the whole tested range. Moreover, the comparison is conservative for GPU@SAT, since its Q8.8 division and square root are software-implemented and more than an order of magnitude slower than their hardware counterparts in single precision.

Numeric precision	NVIDIA Jetson Nano	GPU@SAT
Single	10.290 ms	–
Q8.8	18.826 ms	34.252 ms
Posit 32,3	103.80 ms	188.85* ms

Table 2 *Single-step execution times for absolute orbit propagation with 512 trajectories. *The GPU@SAT posit runtime is estimated by scaling its Q8.8 time by the posit-to-Q8.8 ratio measured on the Jetson Nano.*

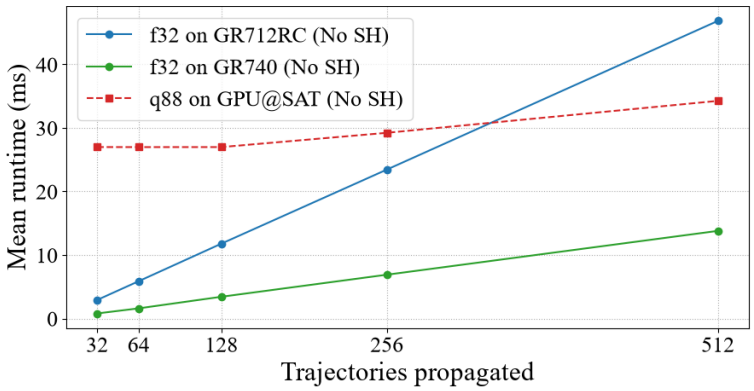

Figure 9 *Step mean runtime for the Q8.8 format on GPU@SAT and single-precision floating point on the GR processors, without spherical harmonics.*

Overall, already at this early development stage GPU@SAT outperforms one of the two space-grade processors in the high-trajectory regime, and its parallel advantage is expected to widen once a hardware posit unit becomes available and the spherical-harmonics memory bottleneck is mitigated.

6 Application

A possible application of the proposed propagator is the propagation of uncertainty through Monte Carlo (MC) analysis. The MC approach relies on propagating thousands of initial conditions, each representing a plausible realization of the spacecraft state sampled from an initial probability distribution, from an initial to a final time. The resulting ensemble of trajectories provides an accurate approximation of the spacecraft state Probability Density Function (PDF) at the final epoch. Although MC analysis is widely regarded as the most accurate technique for estimating the evolution of a PDF, its high computational cost has traditionally prevented its adoption for onboard applications. By exploiting onboard GPU parallelization, the propagation of the large number of trajectories required for MC analysis can be significantly accelerated, reducing the overall computational burden.

The scenario considered assumes an initial Gaussian probability distribution centered at the apogee of the initial orbit, characterized by a semi-major axis $a = 8472.671$ km, eccentricity $e = 0.2$ and inclination $i = \pi/2$ rad, with position and velocity initial uncertainty levels of $\sigma_r = 1000$ m and $\sigma_v = 1$ m/s, respectively.

The aim of this analysis is to evaluate the statistical moments of the propagated distribution at perigee after two orbital revolutions. The orbital configuration and propagation time have been selected to induce a sufficient deviation from Gaussianity, making a MC approach necessary to accurately characterize the uncertainty evolution. To assess the accuracy of the proposed propagation method and quantify the resulting errors, the same set of initial conditions is independently propagated using Orekit and used as the ground truth reference.
The final distribution of the samples is shown in Fig. 10:
The error commited by the implemented propagator in the evaluation of the statistical moments is shown in Tab. 3.

The tailored propagator provides an accurate estimation of the four statistical moments, with errors well within the acceptable range for the intended scenario. These results demonstrate its suitability and effectiveness for this class of applications.

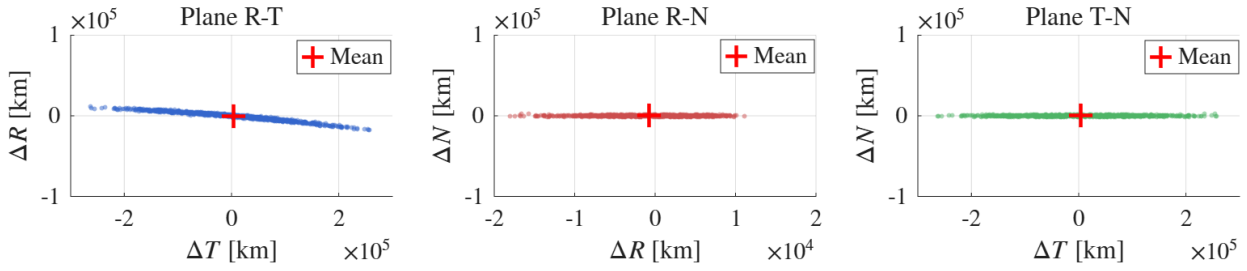

Figure 10 *Final state distribution in the RTN frame.*

	Absolute Error				Relative Error (%)			
State	Mean	Std	Skew.	Kurt.	Mean	Std	Skew.	Kurt.
x [m]	1.349×10^{1}	3.454×10^{-1}	4.050×10^{-6}	3.220×10^{-4}	1.990×10^{-4}	1.679×10^{-2}	3.900×10^{-4}	7.404×10^{-3}
y [m]	1.767×10^{0}	2.137×10^{-4}	2.490×10^{-5}	7.180×10^{-5}	1.755×10^{0}	3.230×10^{-5}	6.909×10^{-2}	2.421×10^{-3}
z [m]	3.251×10^{1}	9.500×10^{-1}	1.210×10^{-4}	8.190×10^{-5}	6.672×10^{-3}	1.003×10^{-3}	3.907×10^{-1}	2.980×10^{-3}
v_x [m/s]	3.087×10^{-2}	8.805×10^{-4}	1.210×10^{-4}	8.370×10^{-5}	5.947×10^{-3}	9.052×10^{-4}	4.982×10^{-1}	3.044×10^{-3}
v_y [m/s]	5.531×10^{-3}	1.980×10^{-6}	1.470×10^{-6}	4.170×10^{-6}	1.950×10^{0}	1.291×10^{-4}	1.626×10^{-3}	1.370×10^{-4}
v_z [m/s]	9.989×10^{-3}	5.050×10^{-4}	3.360×10^{-5}	1.470×10^{-5}	1.190×10^{-4}	4.745×10^{-3}	9.222×10^{-3}	5.100×10^{-4}

Table 3 *Summary of the absolute and relative errors of statistical moments.*

7 Conclusions

This work presented a parallel orbital propagator for LEO, tailored to onboard execution on GPU@SAT, a Class-1 space-qualified soft-GPU offering 512 threads but only 32-bit integer arithmetic. Three parallel-friendly integrators (RK4, DP8, and Picard–Chebyshev) were implemented, with the force models reformulated to use only the four elementary operations and the square root. To counter low-precision round-off, two complementary strategies were adopted: error compensation and the Posit 32,3 format, in synergy with the dimensionless dynamics. RK4 was chosen for deployment on the board.

Validated against Orekit over 1000 dispersed LEO orbits, a naive fixed-point transposition proved unusable, whereas compensation and posits restored a relative accuracy of order 10^{-5}, despite the limited precision of the number formats employed. On runtime, parallelism pays off for large ensembles, with GPU@SAT overtaking the GR712RC between 256 and 512 trajectories if spherical harmonics are excluded. Conversely, spherical harmonics memory traffic and the software emulation of posits currently keep the GPU@SAT behind space-graded CPUs with native floating-point formats.

These results show that accurate ensemble propagation is feasible on low-precision space hardware, with methods transferable to other radiation-tolerant accelerators. Future work includes a hardware posit unit, a memory-aware gravity model, and an on-board validation of accuracy.

Acknowledgements

This work has been conducted as part of ESA Contract No. 4000144652/24/I-DT "EO MakerSpace: AI on Edge Applied to EO Smart Sensors".

References

[1] Masat, A., Colombo, C., and Boutonnet, A., "GPU-based High-Precision Orbital Propagation of Large Sets of Initial Conditions Through Picard–Chebyshev Augmentation," *Acta Astronautica*, Vol. 204, 2023, pp. 239–252. https://doi.org/10.1016/j.actaastro.2022.12.040.

[2] Benelli, G., Todaro, G., Monopoli, M., Giuffrida, G., Donati, M., and Fanucci, L., "GPU@SAT DevKit: Empowering Edge Computing Development Onboard Satellites in the Space-IoT Era," *Electronics*, Vol. 13,

No. 19, 2024. https://doi.org/10.3390/electronics13193928, URL https://www.mdpi.com/2079-9292/13/19/3928.

[3] Atallah, A. M., Woollands, R. M., Elgohary, T. A., and Junkins, J. L., "Accuracy and Efficiency Comparison of Six Numerical Integrators for Propagating Perturbed Orbits," *The Journal of the Astronautical Sciences*, Vol. 67, No. 2, 2020, pp. 511–538. https://doi.org/10.1007/s40295-019-00167-2.

[4] Woollands, R., and Junkins, J. L., "Nonlinear Differential Equation Solvers via Adaptive Picard–Chebyshev Iteration: Applications in Astrodynamics," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 5, 2019, pp. 1007–1022. https://doi.org/10.2514/1.G003318.

[5] Bai, X., and Junkins, J. L., "Solving Initial Value Problems by the Picard-Chebyshev Method with NVIDIA GPUs," *Advances in the Astronautical Sciences*, Vol. 136, 2010, pp. 1459–1476.

[6] Curtis, H. D., *Orbital Mechanics for Engineering Students*, 4th ed., Elsevier, 2020. https://doi.org/10.1016/C2018-0-02232-8.

[7] Montenbruck, O., and Gill, E., *Satellite Orbits: Models, Methods, and Applications*, Springer, Berlin, Heidelberg, 2000.

[8] Russell, R. P., and Arora, N., "Global Point Mascon Models for Simple, Accurate, and Parallel Geopotential Computation," *Journal of Guidance, Control, and Dynamics*, 2012. https://doi.org/10.2514/1.54533.

[9] Srivastava, V. K., et al., "Eclipse prediction methods for LEO satellites with cylindrical and conical geometries: A comparative study of ECSM and ESCM to IRS satellites," *Astronomy and Computing*, Vol. 2, 2013, pp. 11–17. https://doi.org/10.1016/j.ascom.2013.06.001.

[10] Klöwer, M., Hatfield, S., Croci, M., Düben, P. D., and Palmer, T. N., "Fluid Simulations Accelerated With 16 Bits: Approaching 4x Speedup on A64FX by Squeezing ShallowWaters.jl Into Float16," *Journal of Advances in Modeling Earth Systems*, Vol. 14, No. 2, 2022, p. e2021MS002684. https://doi.org/10.1029/2021MS002684, URL https://doi.org/10.1029/2021MS002684.

[11] Muller, J.-M., et al., *Handbook of Floating-Point Arithmetic*, Birkhäuser, Basel, Switzerland, 2018. https://doi.org/10.1007/978-3-319-76526-6, URL https://doi.org/10.1007/978-3-319-76526-6.

[12] Møller, O., "Quasi double-precision in floating point addition," *BIT Numerical Mathematics*, Vol. 5, No. 1, 1965, pp. 37–50. https://doi.org/10.1007/BF01975722, URL https://doi.org/10.1007/BF01975722.

[13] Gustafson, J. L., and Yonemoto, I. T., "Beating Floating Point at its Own Game: Posit Arithmetic," *Supercomputing Frontiers and Innovations*, Vol. 4, No. 2, 2017, pp. 71–86. https://doi.org/10.14529/jsfi170206, URL https://doi.org/10.14529/jsfi170206.

[14] Gustafson, J. L., "Posit Arithmetic," , 2017. URL https://posithub.org/docs/Posits4.pdf.

[15] Battaglia, G., Matarrese, C., Nicali, A., and Panicucci, P., "OpenCL Interface," https://github.com/giabattag/opencl-interface, 2026.

[16] Frontgrade Gaisler, "GR-CPCI-GR740 Development Board," https://www.gaisler.com/products/gr-cpci-gr740, 2026.

[17] Frontgrade Gaisler, "GR712RC Development Board," https://www.gaisler.com/products/gr712rc-board, 2026.

[18] NVIDIA Corporation, "Get Started With Jetson Nano Developer Kit," https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit, 2026. Accessed: 2026-07-15.

[19] Jääskeläinen, P., de La Lama, C. S., Schnetter, E., Raiskila, K., Takala, J., and Berg, H., "pocl: A Performance-Portable OpenCL Implementation," *International Journal of Parallel Programming*, Vol. 43, No. 5, 2015, pp. 752–785. https://doi.org/10.1007/s10766-014-0320-y, URL http://dx.doi.org/10.1007/s10766-014-0320-y.