M. Massari, P. Di Lizia, M. Rasotto
Nonlinear Uncertainty Propagation in Astrodynamics Using Differential Algebra and Graphics Processing Units
Journal of Aerospace Information Systems, Vol. 14, N. 9, 2017, p. 493-503
doi:10.2514/1.I010535

Numerical Methods for Monte Carlo Nonlinear Uncertainty Propagation in Astrodynamics using Differential Algebra and GPGPUs

Mauro Massari [a] and Pierluigi Di Lizia [b]
Politecnico di Milano, Milan, Italy

Mirco Rasotto [c]
Dinamica Srl, Milan, Italy

In this paper two numerical methods for nonlinear uncertainty propagation in astrodynamics are presented and thoroughly compared. Both methods are based on the Monte Carlo idea of evaluating multiple samples of an initial statistical distribution around the nominal state. However, while the GPGPU implementation aims at increasing the performances of classical Monte Carlo approach exploiting the massively parallel architecture of modern GPGPUs, the method based on Differential Algebra is aimed at the improvement and generalization of standard linear methods for uncertainty propagation. The two proposed numerical methods are applied to test cases considering both simple two-body dynamics and a full n-body dynamics with JPL ephemeris. The results of the propagation are thoroughly compared with particular emphasis on both accuracy and computational performances.

I. Introduction

The problem of nonlinear uncertainty propagation represents a crucial issue in spaceflight dynamics since all practical systems - from vehicle navigation to orbit determination or target track-

[a] Assistant Professor, Department of Aerospace Science and Technology, Via La Masa 34, 20156, Milan , Italy
[b] Assistant Professor, Department of Aerospace Science and Technology, Via La Masa 34, 20156, Milan , Italy
[c] Senior Engineer, Via Morghen 13, 20158, Milan , Italy

ing - involve nonlinearities of one kind or another. One topic of recent interest, for example, is the accurate representation of spacecraft position and velocity uncertainties under nonlinear orbital dynamics. Also space surveillance requires methods to propagate the initial condition uncertainty of resident space objects in order to identify and track them. Another relevant application consists in computing landing dispersion both for reentry missions and for estimating the casualty area of space debris [1]. In addition, within the space mission design process, uncertainty propagation is a fundamental tool to assess the satisfaction of mission constraints, to evaluate mission performances, to perform sensitivity analyses, and to verify the robustness of guidance and control laws. However, most spaceflight mechanics problems involve nonlinear behavior. This observation finds proof even in basic applications. For instance, many problems in celestial mechanics require conversions between different coordinate systems (e.g. the conversion from polar to Cartesian coordinates that form the foundation for the observation models of many sensors). Such transformations are typically nonlinear and therefore entail the problem of applying nonlinear transformations to the estimated statistics.

Uncertainty propagation and estimation in nonlinear systems is extremely difficult. The optimal Bayesian solution to the problem requires the propagation of the description of the full probability density function (PDF). Because the form of the PDF is not restricted, it cannot be described using a finite number of parameters and any practical estimator must use an approximation of some kind [2]. Hence, the main goal of uncertainty propagation is to obtain an accurate finite-dimensional representation of the predicted state PDF. This requires developing tools to improve the approximation of standard linear methods available in the literature or to reduce the computational time required by standard Monte Carlo simulations. In this paper two numerical methods for nonlinear uncertainty propagation are presented and compared.

The first proposed method is aimed at increasing the computational performances of classical Monte Carlo simulations exploiting their intrinsic parallel structure and taking advantage of the massively parallel architecture of modern GPGPUs (General-Purpose computing on Graphical Processing Units) [3]. GPGPUs have been successfully applied in the field of astrodynamics in the solution of Multiple-Rendezvous problem [4] and in the parallel propagation of trajectories [5].

GPGPUs are typically characterized by a very high number of computing cores (usually more than one thousand) with dedicated fast memory without caches, in order to exploit to a maximum the performance of each computing core. Obviously the features of a single arithmetic logic unit (ALU) are much more limited with respect to ALUs found on a general purpose CPU which is much more complex. GPGPU computation is affected also by a series of limitations, mainly due to the fact that GPGPUs are usually separated from the main CPU and memory controller, thus reducing the bandwidth of memory transfer between the CPU main memory and the GPGPU memory to that of the PCI Express Bus. For this reason the computational code developed for GPGPUs should be designed in a completely different way than the code developed for CPUs.

The second proposed method is based on Differential Algebra [6] and is originally aimed at the improvement and generalization of the standard linear methods. Differential Algebra supplies the tools to compute the derivatives of functions within a computerized environment. More specifically, by substituting the classical implementation of real algebra with the implementation of a new algebra of Taylor polynomials, any function f of n variables is expanded into its Taylor polynomial up to an arbitrary order k. The availability of such high order expansions can be exploited when uncertainties are propagated through nonlinear functions or dynamical models.

The two proposed numerical methods are applied to test cases considering both the simple two-body dynamics and full n-body dynamics with JPL ephemeris. The results of the propagation are thoroughly compared with particular emphasis on both the accuracy and the computational performances of the proposed methods. In the next section the nonlinear propagation of uncertainty is introduced, with the details of the two compared methods, the parallel Monte Carlo on GPGPUs and Differential Algebra based Monte Carlo. Then, in the following section the test cases considered are introduced, followed by the results of the uncertainty propagation and by the conclusions.

II. Nonlinear Uncertainty Propagation

Given a dynamical system, the evolution of a particular state can be completely characterized by the governing equations of motion that may be affected by uncertain system models, inputs, or measurements. Hence, studying a deterministic trajectory may not provide sufficient information about the trajectory. Thus, the uncertainty propagation represents a crucial issue, especially in space

application such as spacecraft navigation, orbit determination, target tracking or space surveillance.

Given an initial state and its associated uncertainties (usually described with a mean and a covariance matrix or a probability density function), the goal of uncertainty propagation is to predict the state and its statistical properties at some future time, or possibly along the entire trajectory, considering the statistical properties of the initial state. Except under certain assumptions, however, uncertainty propagation is an extremely difficult process, especially if one wants a complete statistical description. This is because it generally requires one to solve partial differential equations such as the Fokker-Planck Equation (FPE) or to carry out particle-type studies such as Monte Carlo simulations. Moreover, most spaceflight mechanics problems involve the nonlinear behavior, making the uncertainty propagation problem much more tough. The linear assumption can sometimes fail to characterize the true spacecraft dynamics and statistics when a system is subject to a highly unstable environment or when mapped over a long duration of time. Thus, in such cases, an alternate method which accounts for the system nonlinearity must be implemented.

In this paper an efficient parallel implementation on GPGPUs of classical Monte Carlo simulations is compared with Monte Carlo simulations based on repeated evaluation of high order expansion of the flow.

A. Monte Carlo Simulations

Monte Carlo Simulations (MCS) provide true trajectory statistics, but are computationally very expensive. MCS approximates the statistics by averaging over a large set of random samples. Particularly, given the nonlinear dynamic system with uncertain initial conditions:

$$\dot{x} = f\left(t, x\right) \tag{1}$$

MCS consists of the following three steps:

1. the generation of Ω random samples, x^k $k \in \{1, \ldots, \Omega\}$, according to the statistical uncertainty distribution to be propagated;

2. the propagation of each sample through the solution flow;

3. the approximation of the transformed probability density function by evaluating the desired number of moments from the distribution of the propagated sample points. The first two moments at the time t can be computed as:

$$\mu_i(t) = \frac{1}{\Omega}\sum_{k=1}^{\Omega} x_i^k(t) \qquad i \in \{1,\ldots,N\} \tag{2}$$

$$P_{ij}(t) = \frac{1}{\Omega-1}\sum_{k=1}^{\Omega}\left(x_i^k(t)-\mu_i\right)\left(x_j^k(t)-\mu_j\right) \qquad i,j \in \{1,\ldots,N\} \tag{3}$$

The above-described method, even if valid for a wide range of problems, is computationally expensive since it requires a large number of samples trajectories to be propagated to guarantee the convergence of the statistics of the simulations. Although this caveat is becoming less significant for analysis purposes, it can still be prohibitive if the propagation has to be performed in real-time, such as in robust optimizing control. In addition, every time any change occurs in the initial distribution, the whole simulation needs to be repeated. On the other hand, if properly carried out, it can provide the true statistics.

B. Parallel Monte Carlo on the GPU

As stated above, Monte Carlo Simulations can be computationally very intensive if a good representation of the statistics should be derived. However, it is straightforward to note that all the different propagations of the dynamics in MCS are independent and can be performed in parallel. Thus, if a sufficient number of computing units is available, MCS can be accelerated distributing the computation across multiple units.

Following this approach, parallel MCS implementations are very scalable reducing the computational time linearly with the number of computing units. Unfortunately, this linear reduction of computational time is true only if considering computing units that have fast access to a common memory block. This is surely true on multi-core modern CPUs, but the number of computing units is usually limited to be well below 100 (typically 24 to 64) even for multi CPU Workstations.

In recent years, GPGPUs have become very powerful following the increasing demand of both the gaming market and professional graphical applications. Usually not all available computational

power is used when dealing with far from peak performance scene rendering. For this reason GPGPU producers started to investigate the possibility of using the unused computational power for general purpose computations. Today, high-end graphics cards are all capable of providing vast computational power for general purpose computation in all areas of scientific computing. Applications range from fluid dynamics to data mining [7].

However, GPGPU computational capabilities are conceptually very different from the ones of typical CPUs. This is due to the different architecture of GPGPUs, which originally were designed to perform the same operations in parallel on all the pixels of an image (*Single Instruction Multiple Threads*, SIMT) [7–9].

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

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

Thus, GPGPU architecture seems perfect for the implementation of massively parallel Monte Carlo simulations, overcoming the limitations of multi-core CPUs.

GPGPU computation is affected also by a series of other limitations, mainly due to the fact that GPGPUs are separated from the main CPU and memory controller, thus reducing the bandwidth of memory transfers between the CPU main memory and the GPGPU memory to that of the PCI Express Bus. The theoretical peak data transfer rate of 16 GB/s for the x16 PCI Express generation 2 connector of the NVIDIA Tesla K20 card, for example, is an order of magnitude slower than the

208 GB/s of the DDR5 memory on the card [10].

These are just some of the reasons why traditional parallelization techniques commonly used, such as multithreading on multicore CPU systems with shared memory [11] or distributed computing with message passing on computing clusters [12] cannot be applied directly to GPGPU programming. Instead, code and algorithms developed for GPGPUs must be designed in a different way than those for CPUs.

Best performances on GPGPUs can be obtained when running the same piece of code (called a kernel) in parallel on different data sets while avoiding branching. In this way each thread performs the same operations on a different portion of the dataset which is common to all the threads [9]. This is often referred to as data parallelization.

Note that not all available GPGPUs can be used for general purpose computations. Only GPGPUs dedicated to high-end gaming or built explicitly for computing have reasonable performance. There are mainly two alternatives available on the market, NVIDIA with CUDA cores and AMD (ATI) with the GCN architecture. NVIDIA GPGPUs are more commonly used for GPGPU computing than AMD ones, a fact influenced in large part by the existence of specialized CUDA libraries for scientific computing. Unfortunately the CUDA libraries are proprietary and can only be used to develop code for NVIDIA GPGPUs. However, both NVIDIA and AMD GPGPUs support the OpenCL toolkit [13] for parallel computing.

OpenCL is designed to take advantage of all the possible devices that allow parallel computation, similar to the OpenGL specification for 3D graphics rendering. It consists of a C-like language to write GPGPU code which is compiled at runtime for the selected platform. Thus OpenCL code can be executed on all major GPGPU cards such as those by NVIDIA, Intel and ATI. Furthermore, it provides libraries for both the host side (CPU) as well as the client side (GPGPU) to facilitate common tasks in heterogeneous programming in a hardware independent manner. For detailed description of OpenCL features, we refer the reader to the OpenCL 2.0 Specification [14].

In this work it has therefore been decided to use the OpenCL framework for the implementation of the parallel MCS, so that it can be easily run on both NVIDIA and AMD GPGPU as well as multicore CPUs. In this way a thorough comparison of the performances on those different architectures

can be performed.

The actual implementation of the parallel MCS algorithm in OpenCL required the implementation of a dedicated adaptive step propagator tailored for the peculiar architecture of GPGPU, carefully avoiding all possible bottlenecks due to memory transfer between the host and the GPGPU and to thread divergence. In particular, this last aspect is crucial for achieving high performances on GPGPU as reducing the divergence of execution path of the different threads allows to increase the data parallelization. The developed parallel propagator is based on an adaptive step Runge-Kutta scheme of order 7(8).

One of the side effect of the memory bottleneck due to the PCI Express bus is the fact that it is not possible to use already available libraries developed for the CPU in the computation of the right hand side of the dynamics. Therefore, it is not possible to use already available implementation of ephemeris library, like NAIF spice or the JPL Horizon routine. In particular, access to any resource which is not on the GPGPU memory is highly time consuming, therefore all the necessary data should be transferred to the GPGPU memory once at the beginning. As the GPGPU memory is limited in size, the amount of those data can be limited.

All the modules needed for the computation of the dynamics reported in the next section have been implemented in the OpenCL framework and tailored in the same way as the propagator.

The final architecture of the implemented Parallel Monte Carlo Simulation method on the GPU is reported in figure 1.

C. Differential Algebra based Monte Carlo

Recently, a new set of methods has gained interest. Those methods are referred to as high order methods. They represent a trade-off between the efficiency of linearized models and the accuracy of Monte Carlo simulations using high order information. In principle, high order methods rely on the availability of the expansion of the flow of ODEs. Once this is obtained, different algorithms can be straightforwardly implemented for mapping uncertainties:

1. Monte Carlo simulations based on the repeated evaluation of one single flow expansion rather than on multiple pointwise integrations[15–17];

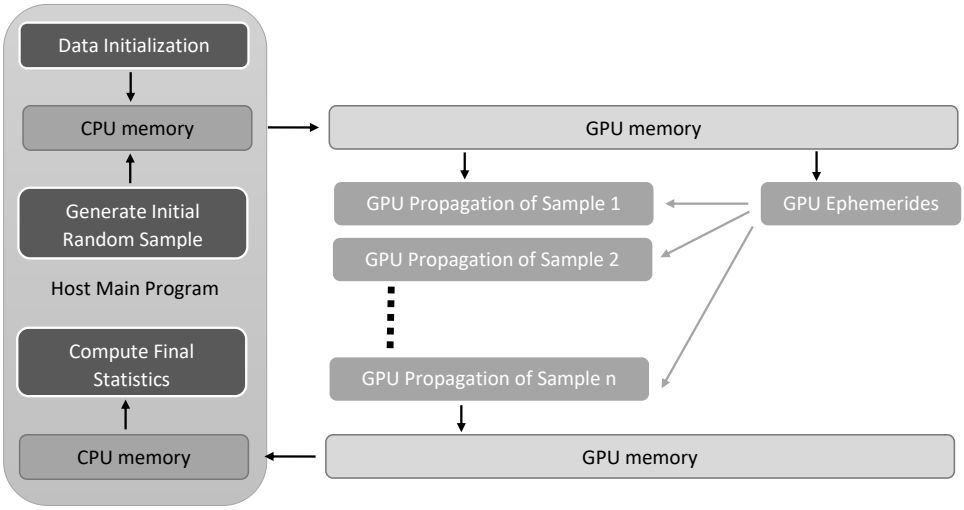


Fig. 1: Architecture of Parallel Monte Carlo on the GPU

2. Analytically mapping PDF[18], or computing high order moments of the mapped PDF [19–21];

3. Range estimation using polynomial bounder[22].

The computation of the expansion of the flow is commonly based on the analytical derivation and subsequent numerical integration of the variational equations related to the State Transition Tensor (STT), which is the high order counterpart of the State Transition Matrix (STM). The main drawbacks of this approach are:

1. It is ODE dependent: high order variational equations must be derived for each dynamical system. This can be tedious and requires extensive computations (regardless of whether symbolic manipulators or finite difference methods are employed).

2. The complexity of the equations of motions increases with the order of the derivatives, thus requiring more time to evaluate.

A better approach for obtaining the high order expansion of the flow is based on the use of Taylor polynomial algebra that allows operating on functions in a computer environment rather than on floating point numbers. The main advantages of this approach are:

1. It is ODE independent: being based on the definition of Taylor polynomial algebra and on operation overloading, coding dynamical systems and expanding the flow of an ODE is easy and compact;

2. A proper implementation of the algebra allows for a drastic reduction of time with respect to variational approaches;

3. The extension to obtain high order expansions with respect to parameters is straightforward.

The high order expansion of the flow of an ODE with respect to initial conditions and uncertain parameters can be easily obtained using the algebra of Taylor polynomials, also known as Taylor differential algebra or just differential algebra (DA). The DA finds its origin in the attempt to solve analytical problems by an algebraic approach. One of the initiators of the field was Liouville in connection with the problem of integration of functions and differential equations in finite terms. It was then significantly enhanced by Ritt (1932), who provided a complete algebraic theory of the solution of differential equations that are polynomials of the functions and their derivatives and that have meromorphic coefficients[23]. Historically, the treatment of functions in numerics has been based on the treatment of numbers, and the classical numerical algorithms are based on the mere evaluation of functions at specific points. DA technique is based on the observation that it is possible to extract more information on a function rather than its mere values. The basic idea is to bring the treatment of functions and the operations on them to the computer environment in a similar way as the treatment of real numbers. In fact, the real numbers cannot be treated, in a strict sense, in a computer environment, instead they are approximated by floating point (FP) numbers with finitely many digits.

With reference to Figure 2, let us consider two real numbers a and b, and their floating point counterpart $\bar{a}$ and $\bar{b}$ respectively; then, given any operation "$*$" in the set of real numbers, an adjoint operation "$\circledast$" is defined in the set of floating point numbers such that the diagram in figure commutes. Consequently, transforming the real numbers a and b in their FP representation and operating on them in the set of FP numbers returns the same result as carrying out the operation in the set of real numbers and then transforming the achieved result in its FP representation. In a similar way, suppose two sufficiently regular functions f and g are given. In the framework of differential algebra, the computer operates on them using their Taylor series expansions, F and G respectively. Therefore, the transformation of real numbers in their FP representation is now substituted by the extraction of the Taylor expansions of f and g. For each operation in the

function space, an adjoint operation in the space of Taylor polynomials is defined such that the corresponding diagram commutes; i.e., extracting the Taylor expansions of f and g and operating on them in the function space returns the same result as operating on f and g in the original space and then extracting the Taylor expansion of the resulting function.

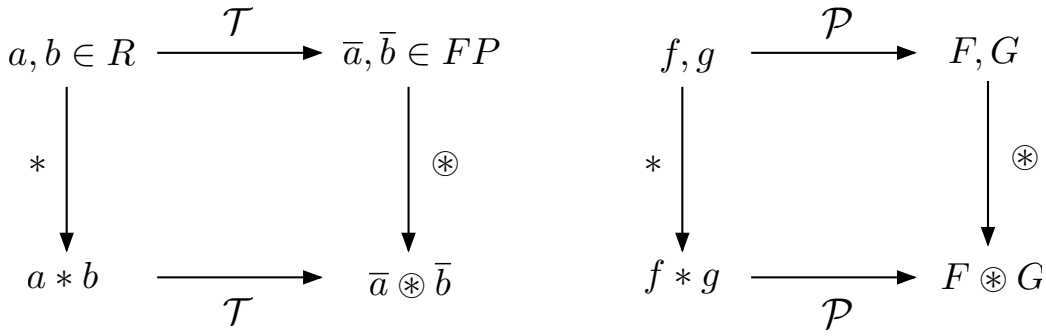


Fig. 2: Analogy between the FP representation of real numbers in computer environment (left) and the algebra of Taylor polynomials in DA framework (right)

The straightforward implementation of differential algebra in a computer allows computing the Taylor coefficients of a function up to a specified order k, along with the function evaluation, with a fixed amount of effort. The Taylor coefficients of order k for sums and product of functions, as well as scalar products with real numbers, can be computed from those of summands and factors; this means that the set of equivalence classes of functions can be endowed with well-defined operations, leading to the so called truncated power series algebra (TPSA). In addition to sum and product, other algebraic operations can be performed, as composition/inversion of functions or nonlinear systems solution. Moreover, the analytic operations of differentiation and integration have been developed on these function spaces, defining a differential algebraic structure, [23].

The original implementation of differential algebra is contained in the software COSY INFINITY [24] developed at Michigan State University. The differential algebra implementation used in this paper is the DACE (Differential Algebra Computational Engine) software library, developed by Dinamica Srl under the ESA contract: *"ITT AO/1-7570/13/NL/MH Nonlinear Propagation of Uncertainties in Space Dynamics based on Taylor Differential Algebra"*.

The high order expansion of the flow relies on the fact that any integration scheme is based on a finite number of algebraic operations, involving the evaluation of the ODE's right hand side at several integration points. Therefore, carrying out all the evaluations in the DA framework,

one can obtain the kth-order Taylor expansion of the flow of the ODE,$\phi\left(t;\delta x_0;t_0\right)=\mathcal{M}_\phi\left(\delta x_0\right)$, at each integration time, assuming a perturbed initial condition $x_0+\delta x_0$. Without loss of generality, consider the scalar initial value problem:

$$\dot{x}\left(t\right)=f\left(t,x\right),\qquad x\left(t_0\right)=x_0 \tag{4}$$

and the associated flow $\phi\left(t;\delta x_0;t_0\right)$. For the sake of simplicity, consider uncertain initial conditions only. Starting from the DA representation of the initial condition x_0, differential algebra allows us to propagate the Taylor expansion of the flow in x_0 forward in time, up to the final time t_f. To this aim, the point initial condition x_0 is replaced by the kth-order DA representative of its identity function, $[x_0]=x_0+\delta x_0$, which is a $(k+1)$-tuple of Taylor coefficients. Then, all the operations of the numerical integration scheme are carried out in the framework of differential algebra. As a result, at each time step, the flow $\phi\left(t;\delta x_0;t_0\right)$ is approximated as a kth-order Taylor expansion in x_0. The result of the final step is the kth-order Taylor expansion of the solution of the ODE at the final time t_f, which is then computed with a limited amount of effort. For the sake of clarity, consider the forward Euler's scheme:

$$x_i=x_{i-1}+f\left(x_{i-1}\right)\Delta t \tag{5}$$

and substitute the initial value with the DA identity $[x_0]=x_0+\delta x_0$. At the first time step one has

$$[x_1]=[x_0]+f\left([x_0]\right)\Delta t \tag{6}$$

If the function f is evaluated in the DA framework, the output of the first step, $[x_1]$, is the kth-order Taylor expansion of the flow $\phi\left(t;\delta x_0;t_0\right)$ in x_0 for $t=t_1$. Note that, as a result of the DA evaluation of $f\left([x_0]\right)$, the $(k+1)$-tuple $[x_1]$ may include several non zeros coefficients corresponding to high order terms in δx_0. The previous procedure can be inferred through the subsequent steps. The result of the final step is the kth-order Taylor expansion of $\phi\left(t;\delta x_0;t_0\right)$ in x_0 at the final time t_f. Thus, the flow of a dynamical system can be approximated, at each time step t_i, as a kth-order

Taylor expansion in a fixed amount of effort. The conversion of standard integration schemes to their DA counterparts is straightforward both for explicit and implicit solvers. This is essentially based on the substitution of the operations between real numbers with those on DA numbers. In this work, the DA-based propagator uses the same adaptive step Runge-Kutta scheme of order 7(8) developed for the parallel propagator.

More recently, the same approach has been adopted in the Jet Transport Method, [25, 26]. The method uses alternative tools to manipulate Taylor polynomials, e.g. the PARI/GP package has been used in the work by Perez et al., [27].

Once a high order expansion of the flow is obtained, it is possible to implement a DA-based Monte Carlo Simulation (MCS). The use of STT definition in MCS allows avoiding multiple numerical integration of the ODE system corresponding to Ω random samples x^k with $k = \{1, \ldots, \Omega\}$, as it is sufficient to evaluate the high order expansion of the flow at final time for each x^k. More in details, the mean and covariance matrix at time t can be computed by:

$$\delta\mu_i(t) = \frac{1}{\Omega}\sum_{k=1}^{\Omega}\phi^i\left(t;\delta x_0^k;t_0\right) \tag{7}$$

$$P_{ij} = \frac{1}{\Omega-1}\sum_{k=1}^{\Omega}\left[\delta x_i\left(t;\delta x_0^k;t_0\right)-\delta\mu_i(t)\right]\left[\delta x_j\left(t;\delta x_0^k;t_0\right)-\delta\mu_j(t)\right] \tag{8}$$

where δx_0^k is the variation in initial condition associated to the k^{th} sample.

Whenever DA framework is used, the MCS is referred as DAMC-k, where DAMC stands for "DA-based Monte Carlo" and k indicates the Taylor series expansion order.

III. Test Cases

To compare the two proposed methods for nonlinear uncertainty propagation, two main test cases in the field of astrodynamics and space engineering have been selected: the simple two-body problem and a more advanced n-body problem with full solar system JPL ephemeris. The two tests cases are illustrated in the following.

A. Two Boby problem

In the two-body problem, the dynamics of an uncontrolled artificial satellite orbiting the Earth is described by the second-order differential equations

$$\frac{d\ddot{\boldsymbol{r}}}{dt} = -\frac{\mu}{\boldsymbol{r}^3}\boldsymbol{r}, \tag{9}$$

where $\boldsymbol{r}$ is the position vector of the spacecraft, and μ is the Earth's gravitational parameter. The initial state assumed for the analysis is reported in table 1 both in terms of Cartesian coordinates and orbital elements.

Table 1: Initial state in the 2 body problem.

Element	Value	Units	Coord.	Value	Units
a	1.5e4	km	r_x	7.5e3	km
e	0.5	-	r_y	0.0	km
i	0.0	rad	r_z	0.0	km
Ω	0.0	rad	v_x	0.0	km/s
ω	0.0	rad	v_y	8.9286	km/s
θ	0.0	rad	v_z	0.0	km/s

For the test, only the r_x and r_y components of the position vector are affected by uncertainty δx and δy, with a variance of $5e-02\ km^2$. The propagation is performed for 30.8 orbital periods, thus computing the statistics in the proximity of the pericenter of the orbit (the initial condition) but not exactly there.

B. N-body dynamics

For this second test case, the motion of a Near Earth Object (NEO) in the Solar System is considered. In this scenario, the object is affected by the gravitational attraction of n bodies, but has no gravitational effect on them. To properly describe its dynamical evolution a n-body model is used, for which the equations of motion, neglecting the relativistic effects, are given by

$$\ddot{\boldsymbol{r}} = G\sum_i \frac{m_i(\boldsymbol{r}_i - \boldsymbol{r})}{r_i^3}, \tag{10}$$

where $\boldsymbol{r}$ is the object's position, G is the gravitational constant; m_i and $\boldsymbol{r}_i$ are the mass and the Solar System barycentric position of the $i-th$ body or planetary system, respectively; $r_i = |\boldsymbol{r}_i - \boldsymbol{r}|$. In Eq. 10, the positions and velocities of the n bodies are considered as given values, computed from the JPL DE405 ephemeris model. In the performed propagation n includes the Sun, planets, and the Moon. For planets with moons, with the exception of the Earth, the center of mass of the system is considered. The dynamical model is written in the J2000 Ecliptic reference frame.

The NEO object considered for this example is the asteroid Apophis, whose initial nominal state on 17 June 2009 at $23:58:53.815$, is reported in Table 2, along with the associated standard deviations σ. Differently from what seen in the two-body example, all the state components are affected by uncertainty.

Table 2: Initial Apophis state and corresponding σ values.

State	Value	σ	Units
r_x	-2.7771869001033994e-01	9.205222855173289e-06	AU
r_y	9.9156793819790212e-01	1.149240014780420e-06	AU
r_z	-5.8967717948226785e-02	4.964697377316424e-07	AU
v_x	-1.5880316335966243e-02	4.365134893019895e-08	AU/day
v_y	-1.8594872501471037e-03	1.553100126996160e-07	AU/day
v_z	-2.8397506615006335e-04	1.947846701786175e-08	AU/day

The propagation lasts for almost 20 years, until 14 April 2029 at $21:45:05.736$, just before the close approach with our planet. An absolute and relative tolerance of 10^{-12} is used in the step adaptation.

IV. Results

The presented test cases have been run on different platforms and architectures. In particular, Parallel Monte Carlo simulations have been executed both on CPUs (single and multiple cores) and GPUs, while the Taylor differential Algebra Monte Carlo has been executed only on CPUs using a single core. The details of the platforms used for the comparison are reported in table 3.

In order to thoroughly compare the two proposed methods, the following simulations have been performed on both the test cases:

1. Sequential Monte Carlo Simulation on CPUs

Table 3: Spec. of simulation platforms.

Name	Type	Cores (Virtual)	Freq. [GHz]	RAM [GB]
4xAMD Opteron 6376	CPU	64	2.3	256
AMD Phenom II X6 1075T	CPU	6	3.0	8
Intel Core i7-4820k	CPU	4(8)	3.7	32
AMD Radeon HD 7950	GPU	1792	0.925	3
NVIDIA Tesla K20c	GPU	2496	0.706	5

2. Parallel Monte Carlo Simulation on CPUs and GPUs

3. DA Monte Carlo of order 1 (STM)

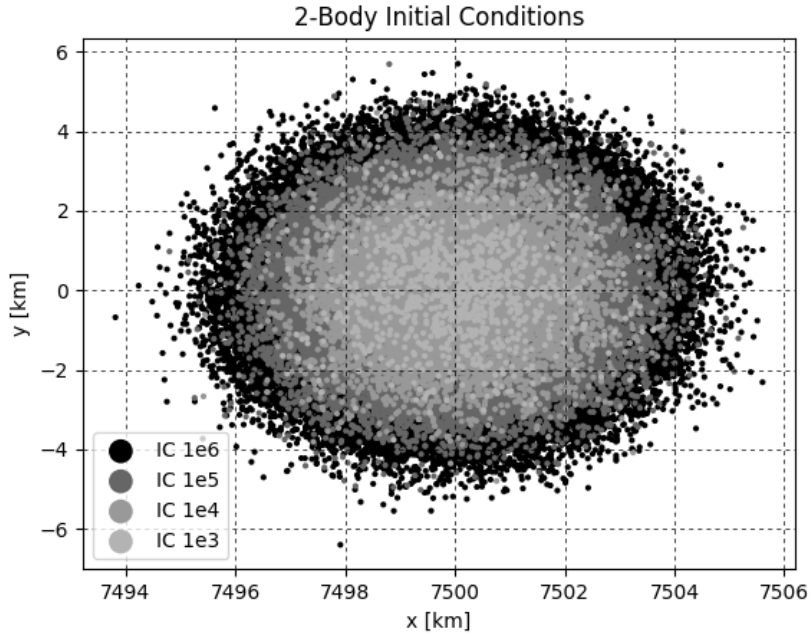
4. DA Monte Carlo of order 3 and 6 (STT)

All the above listed simulations have been performed on the different platforms and considering different amount of initial random samples (from $1 \cdot 10^2$ to $1 \cdot 10^6$). As all the platforms and architectures considered for the simulations conform to the IEEE 754 standard for Binary Floating-Point Arithmetic, the differences in results obtained across different platforms are null. Therefore, in the following the results shown can be considered representative of all the simulations performed.

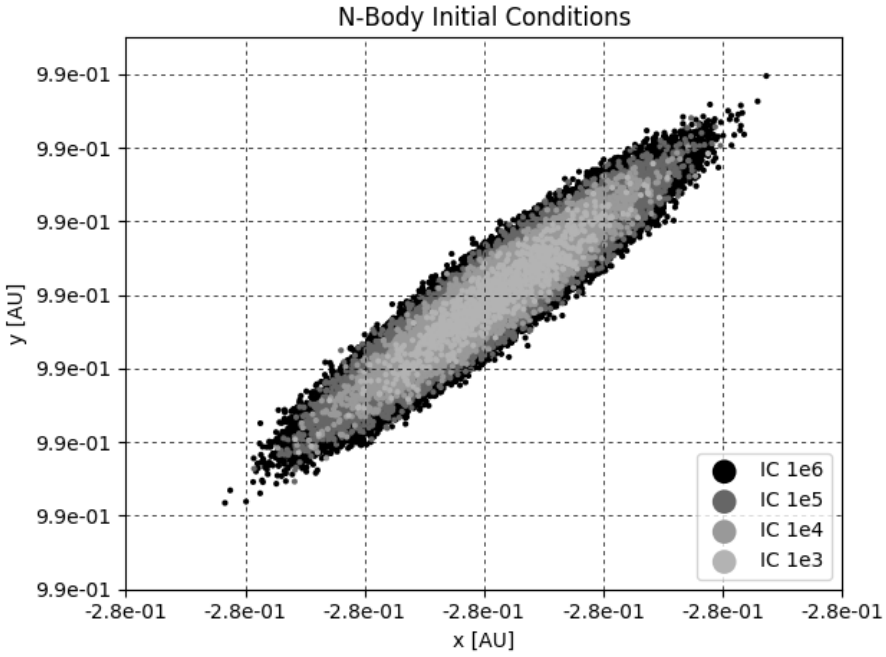
In figure 3 the initial condition for both the test cases are shown, while the resulting final states for the different simulations in the case of $1 \cdot 10^6$ samples are shown in figure 4.

From figure 3 it is clear that in the case of two-body dynamics the variance is constant on both axis, while in the case of Apophis the variance is more representative of a real case. Moreover, it is also worth noting that increasing the sample number it is possible to increase also the distance from the nominal point for an higher number of samples.

Looking at figure 4 it is possible to see that the DAMC-k method is approximating very well the MCS results in the case of $k = 6$ in both the test cases. In particular, Table 4 shows the relative error in the statistical moments of the PDF of the final state obtained by DAMC-k with respect to the moments of the PDF obtained with Pararallel MCS. The statistical moments considered are the mean μ, the variance σ^2, the skewness γ_1 and the kurtosis κ as defined for the samples $(s_1, s_2, \ldots, s_N)$ by:

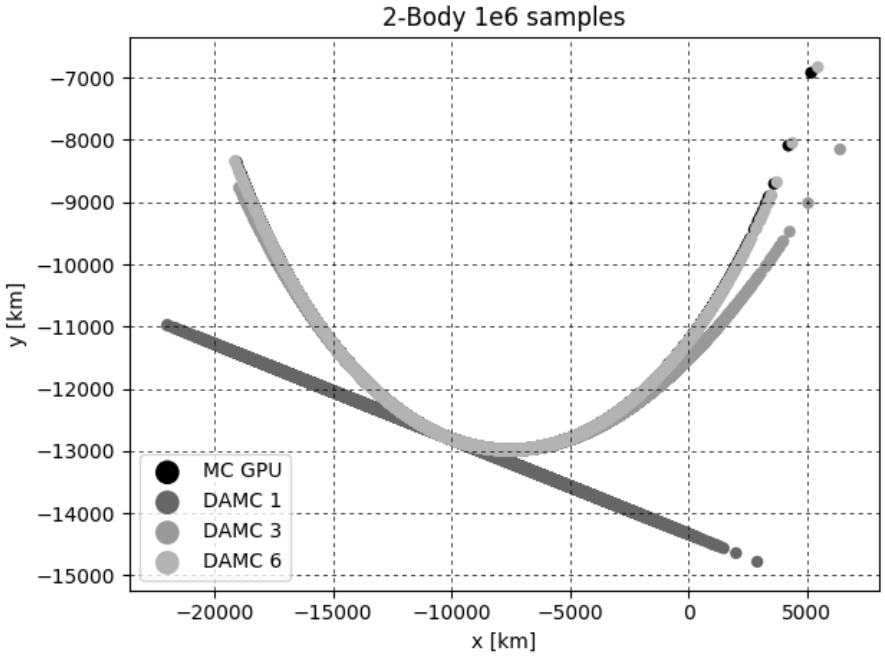


(a)

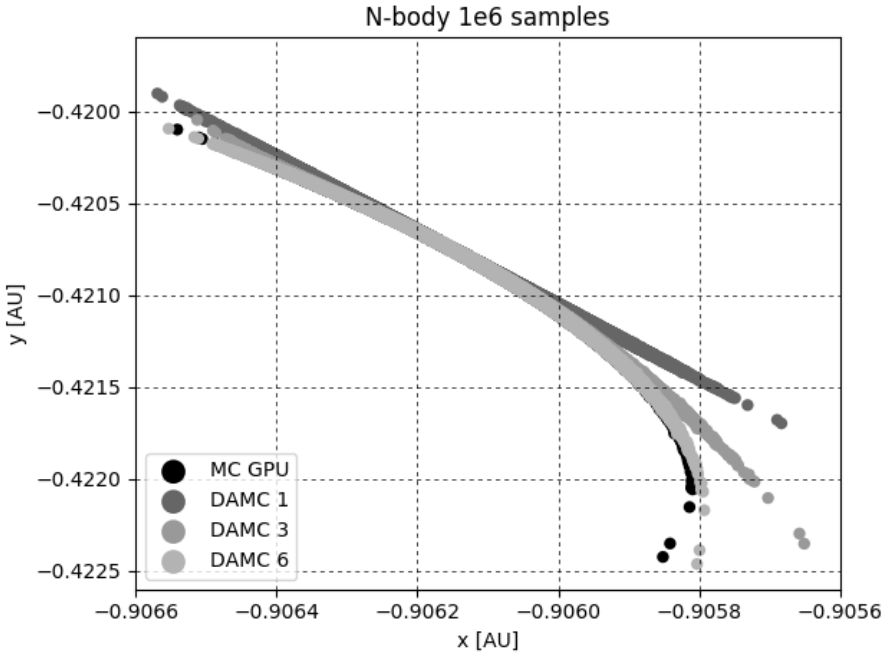


(b)

Fig. 3: Initial condition for 2-Body (a) and N-Body (b) simulations



(a)



(b)

Fig. 4: Final states for 2-Body (a) and N-Body (b) in case of $1 \cdot 10^6$ simulations

$$\mu = \frac{1}{N}\sum_{j=1}^{N} s_j \tag{11}$$

$$\sigma^2 = \frac{1}{N-1}\sum_{j=1}^{N} (s_j - \mu)^2 \tag{12}$$

$$\gamma_1 = \frac{1}{N}\sum_{j=1}^{N} \left(\frac{s_j - \mu}{\sigma}\right)^3 \tag{13}$$

$$\kappa = \left[\frac{1}{N}\sum_{j=1}^{N} \left(\frac{s_j - \mu}{\sigma}\right)^4\right] - 3 \tag{14}$$

where N is the total number of samples.

Table 4: Statistical moments relative error [in % of MCS results]

	Two Body			N-Body		
	DAMC-1	DAMC-3	DAMC-6	DAMC-1	DAMC-3	DAMC-6
μ_x	1.39	0.02	0.00	0.00	0.00	0.00
μ_y	1.45	0.03	0.00	0.00	0.00	0.00
μ_z	0	0	0	1.43	0.08	0.02
μ_{vx}	0.16	0.07	0.00	0.01	0.00	0.00
μ_{vy}	14.17	0.47	0.00	0.10	0.00	0.00
μ_{vz}	0	0	0	1.66	0.09	0.02
σ_x^2	0.22	0.17	0.02	2.06	0.55	0.08
σ_y^2	18.45	2.69	0.02	5.82	0.04	0.17
σ_z^2	0	0	0	18.65	1.87	0.02
σ_{vx}^2	5.86	0.91	0.12	3.99	0.81	0.10
σ_{vy}^2	12.05	0.78	0.04	6.46	0.05	0.20
σ_{vz}^2	0	0	0	17.39	1.59	0.00
γ_{1x}	99.60	3.23	0.11	98.85	28.15	0.39
γ_{1y}	100.07	3.45	0.04	99.73	3.79	0.16
γ_{1z}	0	0	0	99.88	10.93	0.15
γ_{1vx}	101.11	59.99	0.46	98.73	48.67	0.36
γ_{1vy}	99.86	6.85	0.30	99.76	3.36	0.15
γ_{1vz}	0	0	0	99.87	10.04	0.08
κ_x	94.00	21.14	2.10	108.60	73.51	7.04
κ_y	99.84	11.22	0.20	98.92	8.44	0.86
κ_z	0.00	0.00	0.00	99.78	26.80	0.96
κ_{vx}	103.43	16.71	4.68	104.23	51.05	4.60
κ_{vy}	99.53	15.41	1.48	99.10	8.03	0.93
κ_{vz}	0.00	0.00	0.00	99.75	24.75	0.64

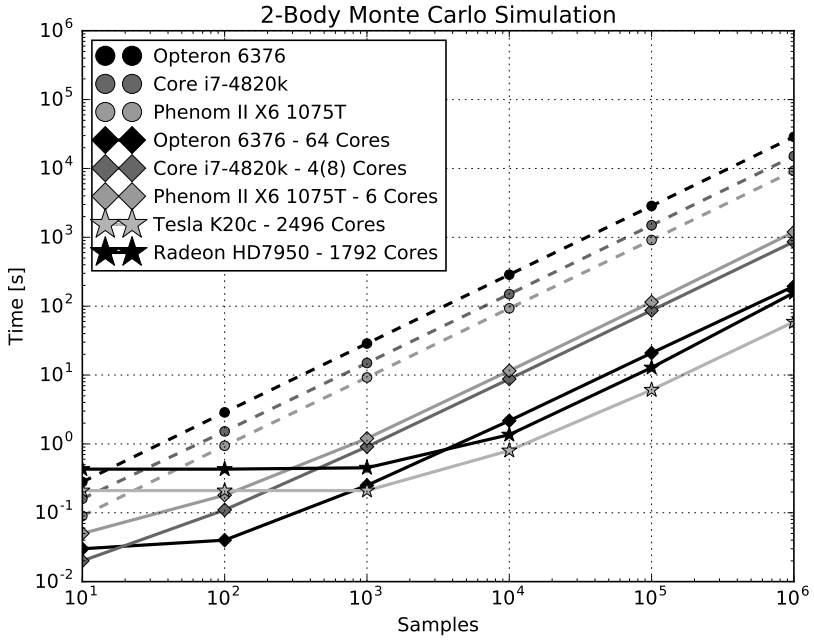
Moreover, it is also possible to see how the DAMC-1 method, which is essentially the linear method based on the STM is not accurate enough (especially in the two body problem), failing to capture the high order moments of the PDF completely.

After comparing the accuracy of the methods the computational performances of both should be confronted. The execution time of the two methods have been computed on the platforms of table 3 and is reported in table 5.

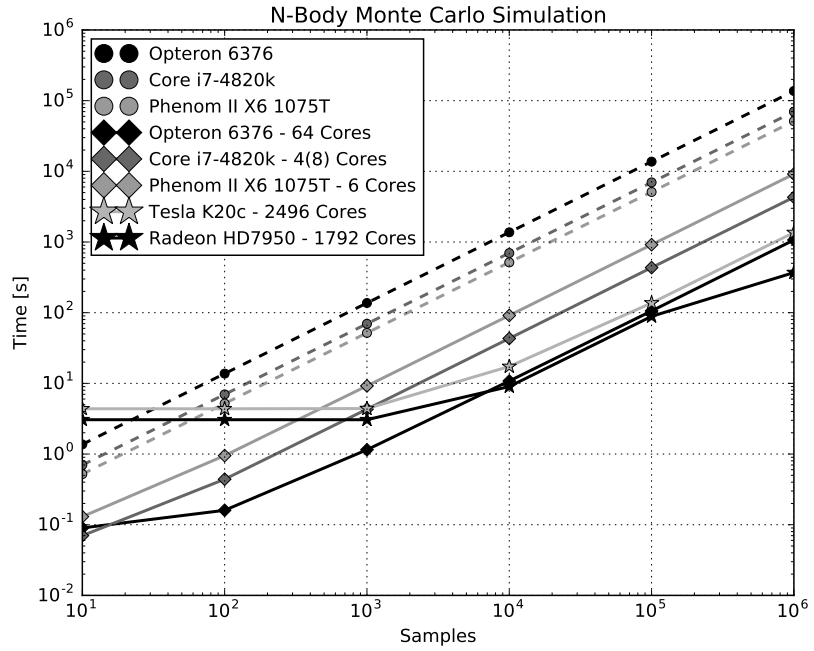
Table 5: Execution time [s] (different samples number)

Execution time [s] for MCS on CPUs and GPUs							
2-Body	Cores	$1 \cdot 10^1$	$1 \cdot 10^2$	$1 \cdot 10^3$	$1 \cdot 10^4$	$1 \cdot 10^5$	$1 \cdot 10^6$
Opteron 6376	1	0.28	2.87	28.69	286.85	2855.14	28783.93
Phenom II X6 1075T	1	0.09	0.94	9.22	93.07	914.03	9250.17
Core i7-4820k	1	0.16	1.52	14.95	149.49	1497.25	15111.18
Opteron 6376	64	0.03	0.04	0.25	2.17	20.89	194.69
Phenom II X6 1075T	6	0.05	0.18	1.2	11.5	114.67	1194.72
Core i7-4820k	4(8)	0.02	0.11	0.91	8.75	86.82	861.97
Tesla K20c	2496	0.21	0.21	0.21	0.8	6.09	59.22
Radeon HD7950	1792	0.43	0.43	0.45	1.36	12.81	156.36
N-Body	Cores	$1 \cdot 10^1$	$1 \cdot 10^2$	$1 \cdot 10^3$	$1 \cdot 10^4$	$1 \cdot 10^5$	$1 \cdot 10^6$
Opteron 6376	1	1.37	13.74	137.17	1371.5	13718.24	137129.22
Phenom II X6 1075T	1	0.52	5.18	51.75	515.11	5109.49	51292.14
Core i7-4820k	1	0.7	6.98	69.69	697.13	6973.27	69848.45
Opteron 6376	64	0.09	0.16	1.15	10.8	105.87	1068.82
Phenom II X6 1075T	6	0.13	0.95	9.21	91.35	916.35	9150.73
Core i7-4820k	4(8)	0.07	0.44	4.31	43.18	434.19	4298.45
Tesla K20c	2496	4.37	4.37	4.4	17.34	137.22	1360.5
Radeon HD7950	1792	3.06	3.06	3.06	9.02	88.73	369.43
Execution time [s] for DAMC-k of various order k							
2-Body	order	$1 \cdot 10^1$	$1 \cdot 10^2$	$1 \cdot 10^3$	$1 \cdot 10^4$	$1 \cdot 10^5$	$1 \cdot 10^6$
Opteron 6376	1	4.84	4.88	5	4.92	5.08	5.21
Phenom II X6 1075T	1	5.32	5.35	5.35	5.35	5.37	5.68
Core i7-4820k	1	2.46	2.42	2.42	2.41	2.41	2.61
Opteron 6376	3	8.87	8.89	8.82	8.87	8.97	9.33
Phenom II X6 1075T	3	9.39	9.39	9.37	9.4	9.4	10.07
Core i7-4820k	3	4.18	4.19	4.22	4.21	4.2	4.43
Opteron 6376	6	23.75	24.01	23.91	23.76	24.07	25.78
Phenom II X6 1075T	6	24.29	24.42	24.31	24.4	24.55	26.15
Core i7-4820k	6	10.72	10.75	10.78	10.94	10.71	11.47
N-Body	order	$1 \cdot 10^1$	$1 \cdot 10^2$	$1 \cdot 10^3$	$1 \cdot 10^4$	$1 \cdot 10^5$	$1 \cdot 10^6$
Opteron 6376	1	3.45	3.47	3.48	3.55	3.52	3.98
Phenom II X6 1075T	1	3.41	3.53	3.55	3.54	3.53	4.1
Core i7-4820k	1	1.75	1.74	1.75	1.76	1.79	2.02
Opteron 6376	3	27.44	27.22	27.29	27.38	27.51	31.32
Phenom II X6 1075T	3	28.79	28.83	28.62	28.55	28.77	33.79
Core i7-4820k	3	12.39	12.44	12.52	12.62	12.58	15.23
Opteron 6376	6	474.09	474.57	479.2	474.06	487.33	496.58
Phenom II X6 1075T	6	500.79	508.61	504.75	504.94	501.34	574.9
Core i7-4820k	6	207.75	212.4	207.5	210.42	224.76	250.74

The same results have been reported also in figure 5 and figure 6 for both the two body and n-body test cases. In figure 5 the computational time of the various cases of the Monte Carlo



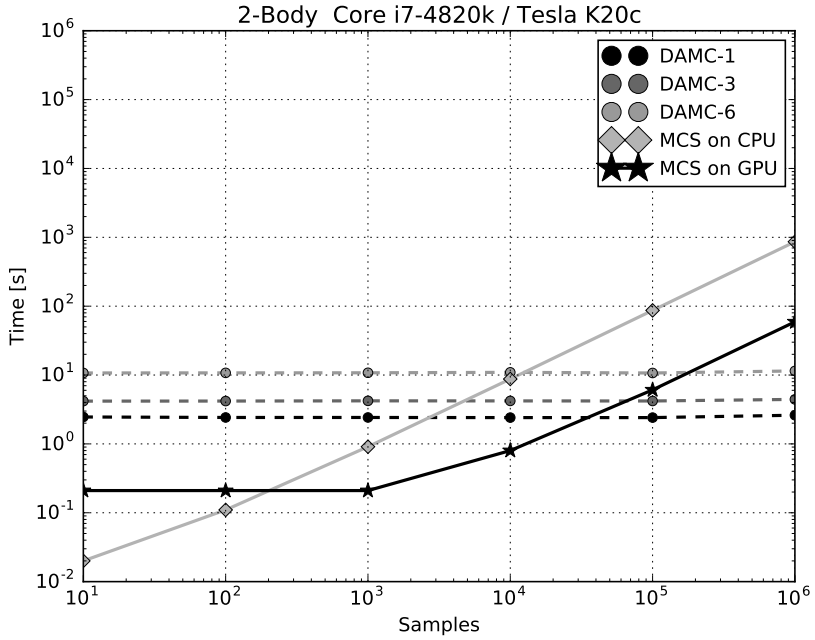
(a)



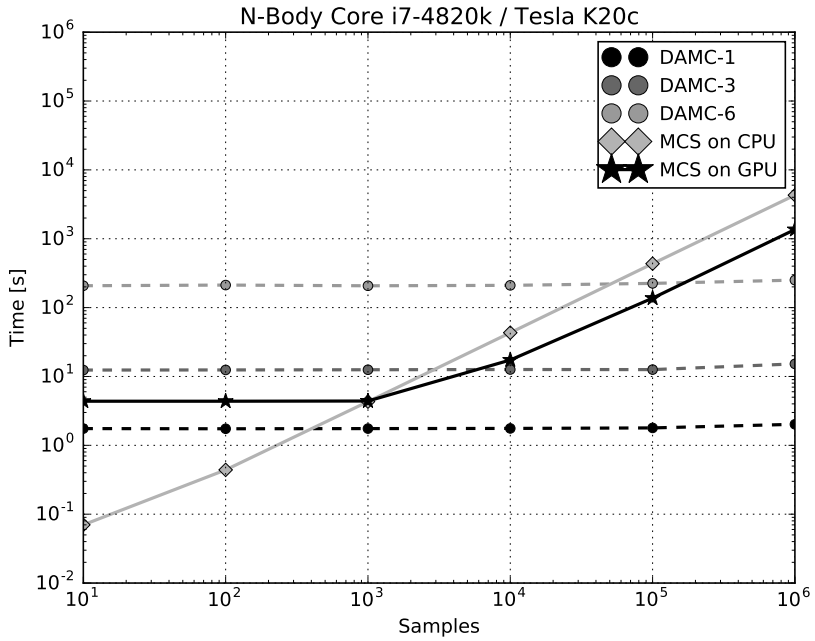
(b)

Fig. 5: Execution Time of MCS method for 2-Body (a) and N-Body (b) test cases

Simulations method are reported, while in figure 6 the execution time of the DAMC-k method is compared with the Parallel MCS on CPU and GPU. In both figures the axis uses a logarithmic scale to simplify the representation of the results.



(a)



(b)

Fig. 6: Execution Time of DAMC-k method for 2-Body (a) and N-Body (b) test cases

From figure 5 it is possible to see how the Sequential MCS computational time increases linearly with the samples number, clearly demonstrating the good scalability of MCS algorithm. Moreover, it is also possible to see that for small number of samples which are less than the number of

computing cores, the Parallel MCS is presenting a constant execution time. This is expected, as the good scalability of the methods allows for a very reduced overhead due to parallelization. However, considering the curve of GPUs, it is possible to see that the constant time region is longer, as there are more computing core, but also higher, indicating a bigger overhead with respect to the CPU cases. This is mainly due to the constant overhead needed to copy the data from the CPU memory to the GPU memory and back. This can be easily seen even looking at the difference between the two body case and the n-body one, where this overhead is bigger and is due to the loading onto the GPU memory of the data needed to compute the ephemeris.

In figure 6 it is possible to see that the DAMC-k has a very constant time of execution, if compared to the MCS. This is mainly due to the computation of the high order expansion of the flow which is independent from the number of samples. The contribution of the samples evaluation is negligible, especially in the two body case where the expansion of the flow is represented by polynomials with far less coefficient than the case of the N-body dynamics. It is also possible to see that the computational time is affected mainly by the order of the computation. However, it is not possible to derive a relation between the order and the computational time, as the DAMC-k is not scalable with the order. This is mainly due to the fact that polynomials of the same order can be characterized by different number of non-zero coefficients. The DACE library is able to tackle efficiently the presence of zero coefficients, thus unbinding the computational time from the order.

From figure 6 it is also possible to see how the DAMC-6 which is comparable to the MCS in terms of accuracy is faster than the parallel MCS on the GPU only for big large number of samples. This is exactly the region of applicability of the DAMC-k methods, when in order to obtain accurate uncertainty propagation the MCS becomes too time consuming, even in the parallel GPU version.

V. Conclusions

In this work, a thorough comparison of Parallel Monte Carlo Simulations method and Taylor Differential Algebra Monte Carlo have been performed, both in terms of accuracy and computational time. From the point of view of the accuracy it is possible to say that the DAMC-k can match the results of the Parallel MCS only with sufficient high order expansion of the flow. From the computational time point of view, the parallel MCS on the GPU is the clear winner only if con-

sidering a number of samples sufficiently small (in the order of $1 \cdot 10^5$). This number of samples is in any case sufficient to correctly describe the statistics with MCS. However, the DAMC-k provide an analytical representation of the expansion of the flow that can be used in ways that are not possible with MCS.

Therefore, it is not possible to declare a clear winner. DAMC-k seems a better method for uncertainty propagation in astrodynamics, especially because it allows computing statistics corresponding to different initial samples without propagating again the flow, thus reducing considerably the time in case of multiple analysis. However, the Parallel MCS is not a clear looser, as the good scalability of the method allows to obtain very similar performances, especially in the GPU version. This will be surely preferable in case of already existing dynamical model that can be easily programmed on the GPU with small modification to the code, that can be very big in the case of DAMC-k implementation. Moreover, the Parallel MCS algorithm can be easily adapted to other applications that require multiple propagations in parallel, like the computation of Poincaré section in Non-Keplerian motions or collision probability computation with big clouds of debris.

VI. Acknowledments

The authors are grateful for the generous support of this work by NVIDIA Corporation through the donation of two NVIDIA Tesla K20c GPGPU accelerator cards within the NVIDIA academic partnership program.

[1] H. Klinkrad, *Space Debris*. John Wiley & Sons, Ltd, 2010, 10.1002/9780470686652.eae325.

[2] S. Julier and J. Uhlmann, "Unscented filtering and nonlinear estimation," *Proceedings of the IEEE*, Vol. 92, Mar 2004, pp. 401–422, 10.1109/JPROC.2003.823141.

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

[4] M. Massari and A. Wittig, "Optimization of Multiple-Rendezvous Low-Thrust Missions on General-Purpose Graphics Processing Units," *Journal of Aerospace Information Systems*, Vol. 13, No. 2, 2016, pp. 1–13.

[5] N. Arora, V. Vittaldev, and R. P. Russell, "Parallel Computation of Trajectories Using Graphics Processing Units and Interpolated Gravity Models," *Journal of Guidance, Control, and Dynamics*, Vol. 38,

No. 8, 2015, pp. 1345–1355.

[6] M. Berz, "Differential algebraic techniques," *Handbook of Accelerator Physics and Engineering. World Scientific*, 1999.

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

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

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

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

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

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

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

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

[15] R. Armellin, P. Di Lizia, F. Bernelli-Zazzera, and M. Berz, "Asteroid close encounters characterization using differential algebra: the case of Apophis," *Celestial Mechanics and Dynamical Astronomy*, Vol. 107, No. 4, 2010, pp. 451–470.

[16] R. S. Park and D. J. Scheeres, "Nonlinear mapping of Gaussian statistics: theory and applications to spacecraft trajectory design," *Journal of Guidance, Control, and Dynamics*, Vol. 29, No. 6, 2006, pp. 1367–1375.

[17] R. S. Park and D. J. Scheeres, "Nonlinear semi-analytic methods for trajectory estimation," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 6, 2007, pp. 1668–1676.

[18] A. B. Younes, J. Turner, M. Majji, and J. Junkins, "High-Order Uncertainty Propagation Enabled by Computational Differentiation," *Recent Advances in Algorithmic Differentiation*, pp. 251–260, Springer, 2012.

[19] D. Giza, P. Singla, and M. Jah, "An approach for nonlinear uncertainty propagation: Application to orbital mechanics," *AIAA Guidance, Navigation, and Control Conference, Chicago IL*, 2009, pp. 1–19.

[20] M. Valli, R. Armellin, P. Di Lizia, and M. Lavagna, "Nonlinear mapping of uncertainties in celestial mechanics," *Journal of Guidance, Control, and Dynamics*, Vol. 36, No. 1, 2012, pp. 48–63.

[21] M. Majji, J. L. Junkins, and J. D. Turner, "A high order method for estimation of dynamic systems," *The Journal of the Astronautical Sciences*, Vol. 56, No. 3, 2008, pp. 401–440.

[22] R. Armellin, P. Di Lizia, M. Berz, and K. Makino, "Computing the critical points of the distance function between two Keplerian orbits via rigorous global optimization," *Celestial Mechanics and Dynamical Astronomy*, Vol. 107, No. 3, 2010, pp. 377–395.

[23] M. Berz, K. Makino, K. Shamseddine, and W. Wan, *Modern map methods in particle beam physics*, Vol. 108. Academic Press, 1999.

[24] K. Makino and M. Berz, "Cosy infinity version 9," *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, Vol. 558, No. 1, 2006, pp. 346–350.

[25] E. Alessi, A. Farrés, À. Jorba, C. Simó, A. Vieiro, À. Jorba, and L. Summerer, "Efficient usage of self validated integrators for space applications," *ESA and Universitat de Barcelona TR-07/5202, Barcelona, Spain*, 2007.

[26] E. M. Alessi, A. Farres, A. Vieiro, A. Jorba, and C. Simó, "Jet Transport and Applications to NEOs," *Proceedings of the 1st IAA Planetary Defense Conference, Granada, Spain*, 2009.

[27] D. Pérez, J. Masdemont Soler, G. Gómez Muntané, *et al.*, "Jet Transport propagation of uncertainties for orbits around the Earth," 2013.