The Journal of the Astronautical Sciences, Vol. 58, No. 4, October–December 2011, pp. 583–613

Modified Chebyshev-Picard Iteration Methods for Orbit Propagation

Xiaoli Bai[1] and John L. Junkins[2]

Abstract

Modified Chebyshev-Picard Iteration methods are presented for solving high precision, long-term orbit propagation problems. Fusing Chebyshev polynomials with the classical Picard iteration method, the proposed methods iteratively refine an orthogonal function approximation of the entire state trajectory, in contrast to traditional, step-wise, forward integration methods. Numerical results demonstrate that for orbit propagation problems, the presented methods are comparable to or superior to a state-of-the-art 12^{th} order Runge-Kutta-Nystrom method in a serial processor as measured by both precision and efficiency. We have found revolutionary long solution arcs with more than eleven digit path approximations over one to three lower-case Earth orbit periods, multiple solution arcs can be patched continuously together to achieve very long-term propagation, leading to more than ten digit accuracy with built-in precise interpolation. Of revolutionary practical promise to much more efficiently solving high precision, long-term orbital trajectory propagation problems is the observation that the presented methods are well suited to massive parallelization because computation of force functions along each path iteration can be rigorously distributed over many parallel cores with negligible cross communication needed.

Introduction

The solution of initial value problems (IVPs) provides the state history of a given dynamic system, for prescribed initial conditions. Beginning with the pioneering work of Euler in the 1700s, numerical methods for solving IVPs have challenged applied mathematicians, engineers, and scientists for about three centuries. Although a substantial amount of literature exists with many well-proven methods for solution of IVPs associated with systems described by nonlinear ordinary differential equations (ODEs), how to optimize the methods to utilize

[1]Postdoctoral Research Associate, Department of Aerospace Engineering, Texas A&M University, TAMU-3141, College Station, Texas 77843-3141.
[2]Regents Professor, Distinguished Professor of Aerospace Engineering, Holder of the Royce E. Wisenbaker '39 Chair in Engineering, Department of Aerospace Engineering, Texas A&M University, TAMU-3141, College Station, Texas 77843-3141.

emerging parallel computing architectures provides a driver for pursuit of enhancements of existing methodology.

Compared with the significant achievement of using parallel computation techniques in other scientific computation fields, research on developing parallel algorithms for solving the IVPs of celestial mechanics is advancing at a slower pace, mainly because most of the current integration methods implemented on parallel machines are only modified versions of traditional forward integration approaches, which are typically poorly suited for parallelization [1]. Recently, Bai and Junkins have proposed Modified Chebyshev Picard Iteration (MCPI) methods for solutions of IVPs and boundary value problems (BVPs) [2–4]. MCPI methods approximate both the state trajectory and the integrand along the trajectory by the same set of discrete Chebyshev polynomials. Through using the Picard iteration method [5], MCPI methods integrate the basis functions term-by-term to establish a recursive trajectory approximation technique that inherently contains the new basis function coefficients linearly on each iteration without linearization. Because it is straightforward to distribute the computation of the integrand at all the discrete nodes to different professors, MCPI methods are inherently parallelizable, thus advanced parallel techniques and computer architectures can be effectively used.

The development of MCPI methods builds on some historical work fusing Picard iteration with approximation theory [6–11]. With the available advanced and inexpensive parallel computation architectures such as Graphics Card Units (GPUs), MCPI methods have made the following contributions.

- A unified vector-matrix form of Chebyshev-Picard methods has been developed and proven to be applicable to solving both IVPs and BVPs computationally efficiently.
- The convergence characteristic of Chebyshev-Picard methods has been studied, new insights are reported that establish fundamental conditions that guarantee the implemented algorithm is a contraction mapping in the vicinity of the solution, over maximal time intervals. These results provide important insights about how to choose the solution segment step size, which is vital for efficiently and accurately solving IVPs on a large interval. The insights are conclusive for the case of linear systems and provide important insight for nonlinear systems.
- We have found that MCPI methods are applicable to high precision satellite motion propagation problems, even prior to parallel implementation.
- We have also implemented MCPI on a graphics card to obtain a parallel implementation and the speedup achieved is the largest that has ever been reported [2].

This article presents MCPI methods for solving IVPs, and is especially addressed to orbit propagation problems. The related methods for solving BVPs have been addressed in a preliminary way in Bai's dissertation [4] and are treated further in a companion to this article. We briefly review relevant background literature first. We then outline the MCPI methods for solving IVPs, followed by presenting recent significant progress from the results reported in references [2] and [3]. In the results prior to the present study, we used a Runge-Kutta (RK) 4–5 method to provide a convenient and familiar reference solution for a qualitative basis for assessment of MCPI methods. For a two-body propagation problem, which is similar to the one studied in the current article, we have achieved up to three orders

of magnitude better accuracy while also achieving over one magnitude of speedup over a RK45 method prior to parallel implementation. Bai's dissertation [4], with a less refined version of the MCPI method, compared many solutions to RK45 and showed one to two orders of magnitude speed up in a serial computational implementation, while maintaining comparable accuracy. In the present study, we compare the performance of MCPI methods with a state-of-the-art RKN12(10) method [12, 13]. We also introduce a newly-developed second order MCPI method that is computationally much more efficient for systems whose equations of motion are a second order system of differential equations, rather than converting them to first order form to apply the original first order MCPI formulation. We also report here insights on how to choose optimal polynomial orders and segment length to guarantee both accuracy and efficiency.

Background Literature Review

Parallel Approaches for Solving IVPs

The most common parallelization approach to solve IVPs of orbital mechanics is to cluster the integration of subsets of the N orbital differential equations on separate processors and compute each orbit using serial integration, but on distributed parallel processors. A more fundamental speedup could be achieved if the computation of each precision orbit was itself highly parallelizable, but most of the currently popular numerical integration methods do not have properties that lend themselves efficiently to highly parallel computation [1]. Franklin compared three approaches to parallelize the existing forward integration methods [14]: a parallel block implicit method, segmenting the equations to separate parts which can be solved using multiple processes, and revising the forward integration methods to a predictor-corrector form which was designed by Miranker and Liniger [15]. Gear [1] proposed two types of parallelism: 1) parallel across the method, and 2) parallel across the problem. The way to use parallel techniques for both explicit RK methods and implicit RK methods [16] by concurrent function evaluation belongs to the first type, and the number of processors that can be utilized depends on the number of stages of the RK methods, which usually is less than twenty [17, 18]. Parallelism of a nonlinear vector differential equation obviously depends on the coupling implicit in the problem itself and one simple example is to distribute the computation of the time derivative of each state to different processors - however, the degree of coupling frequently makes rigorous parallelism difficult. Bellen and Zennaro used an iteration method to solve initial value problem from a guessed starting solution and named their approach as the third type: parallel across the time [19]. A variant of this third type of parallelism is being relied upon in the present article. This third type has also been studied by Gear and Xu [20] and the way they used the Picard Iteration approach belongs to the more general waveform relaxation methods. However, Gear and Xu's approach was found promising for a limited family of problems. For the dynamic systems that are described by second order equations and where the force functions are not dependent on the velocity, the corresponding specialized Runge-Kutta-Nystrom (RKN) methods are usually more efficient than the general purpose RK methods. Houwen et al. studied the stability of implicit RKN methods for solving second order equations based on collocation methods [21]. However, the authors found that there are a number of instability regions when the collocation points are Gauss

nodes, Radau nodes, or Lobatto nodes, which indicate that careful stability studies must accompany the node selection in this method. Sommeijer presented a parallel, explicit RKN method, which computes the s stages of function evaluations in parallel and uses m iterations in the RKN method [22]. He studied using both Gauss-Legendre and Radau type RK methods as the correctors. In the developments below, we considered the same benchmark problem and report over a two order of magnitude increase in the interval over which a converged solution can be obtained by our MCPI algorithm.

Picard Iterations, Chebyshev Polynomials, and Chebyshev-Picard methods

Picard iteration is a successive solution approximation technique that is often used to prove the existence and uniqueness of the solutions to IVPs. However, except for some special cases, it is usually difficult to use the classical Picard iteration method for solving IVPs, mainly because the integrals are not analytically tractable. Several researchers over the past half century have pursued the goal of rendering Picard Iteration a more practical approach for solving IVPs and some moderate degree of success has been achieved. For example, Parker and Sochacki have studied the use of Picard Iteration to generate solutions of IVPs in the form of a family of local Taylor series [23], however, convergence of these series is not generally attractive compared with the methods we present below. In Gear and Xu's study [20], a paradox was reported: using their implementation of the Picard method, their approach converged poorly but was found to be highly parallelizable; however using their generalized Picard method, or waveform formulation, their modified method converged faster but with less parallelism.

Chebyshev polynomials are a complete set of orthogonal polynomials that are very important for function approximation. We remark, for the case of discrete Chebyshev polynomials, these orthogonal functions become complete only as the number of nodes and the degree of the polynomials approach infinity. Practical convergence to small tolerances approaching machine precision is easily demonstrated for most smooth functions, and we have found these polynomials readily approximate multi-revolution gravity and drag perturbed orbits with sub-millimeter precision, so long as propulsive forces are not present. Not surprisingly, when impulsive or highly irregular local disturbances are encountered, then the interval of approximation must be shortened adaptively over the duration of such disturbances. It has been proved that if the zeros of Chebyshev polynomials are used as the nodes for polynomial interpolations, the resulting approximating polynomial minimizes the Runge's phenomenon and provides the best approximation under the minimax norm [24]. Many researchers have contributed to the research on using Chebyshev polynomials to solve IVPs [25–30], but typically not adopting Picard iteration as the basis for the solution process. Note that the most straightforward approach of parameterizing the trajectory in terms of basis functions leads to a nonlinear programming problem, if the trajectory is expanded in a linear combination of basis functions and substituted into the integrand. Urabe demonstrated the existence of an isolated periodic solution always implies the existence of convergent Galerkin approximations and that if there exists a Galerkin approximation of sufficiently high order, under some smoothness conditions, an exact solution exists and the approximation error can be estimated [29]. Urabe also developed a numerical computation approach for periodic nonlinear systems using the Galerkin approximations [30]. However, using nonlinear programming to find the required

large number of basis function coefficients with his approach is computationally inefficient for high dimensional state spaces and frequently the curse of dimensionality also limits practical convergence (this assessment agrees with the discussion by Vlassenbroeck and Dooren [31]).

This proposed MCPI approach builds on the historical formulations by a small group of researchers. Clenshaw and Norton first proposed to solve IVPs using both Picard iteration and Chebyshev polynomials (Chebyshev-Picard methods) [6]. Shave studied using Chebyshev-Picard methods for orbit propagation and estimation problems based on the assumption of a single instruction, multiple data (SIMD) parallel architecture [7]. Sinha and Butcher developed a method that uses Picard iteration and shifted Chebyshev polynomials to symbolically solve for the approximate solutions of the state transition matrix for linear time-periodic dynamic systems [8]. In addition to the work by Shave, the parallel nature of the Chebyshev-Picard methods has also been partially addressed by Feagin [9, 11] and Fukushima [10]. Feagin presented a vector-matrix form of the Chebyshev-Picard method that is closely related to MCPI methods we propose here [9]. Fukushima implemented a Chebyshev-Picard algorithm on a vector computer [10]. However, for one example problem, the vector code was shown to be slower than the scalar code and the author suggested that this was because his approach could not be vectorized efficiently and the compilation put more additional overhead. Before our introduction of MCPI methods, Chebyshev-Picard methods have not been considered a viable competitor to traditional existing methods such as high order RK or multi-step methods to solve the problems of celestial mechanics.

State of the Art Methods for Numerical Solution of Second Order Equation: RKN12(10)

Most of the numerical methods presently in routine use for orbit propagation can be categorized as either Runge-Kutta type methods, multi-step extrapolation methods, or Taylor series (analytical continuation) methods. All of these methods owe their heritage to Euler's original (late 1700s) first order analytical continuation method and/or Gauss' (mid 1800s) predictor-corrector method. The MCPI method proposed here departs from these traditional methods in a way that allows us to approximate a large finite path segment as opposed to extrapolating a small step along the path using either the single or multi-step methods. To illustrate and evaluate the potential for the proposed MCPI methods for solving IVPs, we choose to compare the performance of MCPI with a 12^{th} order Runge-Kutta-Nystrom (RKN) method (RKN12(10)), which utilizes also a 10th order RK approximation to facilitate automatic step size modification for error control [12, 13]. Several recent studies indicate RKN12(10) is representative of the state of the art, because its efficiency and accuracy relative to many of the competing methods have been well-documented [32–36]. Through solving an orbit problem Fox compared the performance of the Gauss-Jackson method (for low eccentricity orbits), Gauss-Jackson-Merson method (for high eccentricity orbits), fourth order Runge-Kutta method, sixth order Runge-Kutta method, embedded seventh order Runge-Kutta-Nystrom (RKN) method, Adams methods, Taylor series methods, and the extrapolation methods by Bulirsch [32]. Fox concluded that the Gauss-Jackson method is the best method for near circular orbits and he found that the seventh order RKN method is the most efficient for high eccentricity. Filippi and Grf [33] and Dormand et al. [13] developed high order RKN methods and showed the relative

efficiency of their higher order RKN methods over the lower order methods. In an important sequence of studies, Montenbruck compared several RK methods, recent higher order RKN methods, multi-step methods, and extrapolation methods for a planar two-body problem, to decide which was "best" based on solution of benchmark problems [34]. The results showed the RKN methods are superior to the pre-existing similar order RK methods and furthermore, if the differential equations do not contain a velocity dependent term, the high-order RKN methods, especially RKN12(10), are as fast or faster than multi-step methods while maintaining high precision. A problem with RKN12(10) reported by Montenbruck is that its efficiency drops significantly if the output points are required at times other than those resulting from the natural optimum step size using the built-in step size control algorithm. More recently, Sharp compared nine non-symplectic and two symplectic integrators through solving four different N-body problems [36]. For the two problems where the force functions are not dependent on the velocity, the RKN12(10) method was found to require the least CPU time with high precision maintained over long solution intervals. Hadjifotinou and Gousidou-Koutita compared a 10th-order Gauss-Jackson method, RKN12(10), and a recurrent power series (RPS) method by using seven test problems where the considered dynamical system involves one to four large moons orbiting a point-mass planet [35]. The integration time was 12,000 days (about 30 years). The RPS method introduces a set of new auxiliary variables to solve the problem by the Taylor series approach. Although the RPS method is shown to have about one order of magnitude better accuracy than the other two methods for the particular problem solved, the algorithm needs to be redesigned whenever the system equations change. Hadjifotinou and Gousidou-Koutita showed that the optimal step sizes for the RPS method are much larger than the other two methods: ranging from 1/13 of an orbit to as large as 1/6 of an orbit (in contrast, the optimal time interval for MCPI solution intervals usually varies from one orbit period to over three orbit periods for near-circular problems, so an order of magnitude longer solution arc is feasible).

Fundamentals of the MCPI Approach

Consider a dynamic system described by a first order differential equation

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(t, \mathbf{x}) \tag{1}$$

with the initial condition $\mathbf{x}(t_0) = \mathbf{x}_0$. The first step of MCPI methods is to transform the generic independent variable t to a new variable τ, which is defined on the valid range, the closed interval $[-1, 1]$, of Chebyshev polynomials

$$t = \omega_1 + \omega_2\tau \qquad \omega_1 = \frac{t_f + t_0}{2} \qquad \omega_2 = \frac{t_f - t_0}{2} \tag{2}$$

Introducing this time transformation of equation (2), equation (1) is re-written as

$$\frac{d\mathbf{x}}{d\tau} = \mathbf{g}(\tau, \mathbf{x}) \equiv \omega_2\mathbf{f}(\omega_1 + \omega_2\tau, \mathbf{x}) \tag{3}$$

and Picard iteration provides the solution of equation (3) as

$$\mathbf{x}^i(\tau) = \mathbf{x}_0 + \int_{-1}^{\tau} \mathbf{g}(s, \mathbf{x}^{i-1}(s))\, \mathrm{d}s \quad i = 1, 2, \ldots \tag{4}$$

Now, we introduce Chebyshev polynomial approximations of both the unknown trajectory $\mathbf{x}^i$ and the integrand of equation (4) along the trajectory $\mathbf{x}^i$. The Chebyshev polynomial of degree k is denoted by T_k. The $(N + 1)$ discrete nodes that are used to represent the state trajectory are the Chebyshev-Gauss-Lobatto (CGL) nodes, which are computed through

$$\tau_j = \cos(j\pi/N) \quad j = 0, 1, 2, \ldots, N \tag{5}$$

Assume the force function vector is approximated by an N^{th} order Chebyshev polynomial

$$\mathbf{g}(\tau, \mathbf{x}^{i-1}(\tau)) \approx \sum_{k=0}^{k=N}{}' \mathbf{F}_k^{i-1} T_k(\tau)$$

$$\equiv \tfrac{1}{2}\mathbf{F}_0^{i-1} T_0(\tau) + \mathbf{F}_1^{i-1} T_1(\tau) + \mathbf{F}_2^{i-1} T_2(\tau) + \cdots + \mathbf{F}_N^{i-1} T_N(\tau) \tag{6}$$

Using the discrete orthogonality property of Chebyshev polynomials, the coefficient vectors $\mathbf{F}_k$ can be calculated immediately through [24]

$$\mathbf{F}_k^{i-1} = \frac{2}{N} \sum_{j=0}^{N}{}'' \mathbf{g}(\tau_j, x^{i-1}(\tau_j)) T_k(\tau_j)$$

$$= \frac{1}{N}\mathbf{g}(\tau_0, x^{i-1}(\tau_0)) T_k(\tau_0) + \frac{2}{N}\mathbf{g}(\tau_1, x^{i-1}(\tau_1)) T_k(\tau_1) \ldots + \frac{1}{N}\mathbf{g}(\tau_0, x^{i-1}(\tau_0)) T_k(\tau_0) \tag{7}$$

In equations (6) and (7), $\sum'$ denotes that the first term is halved and $\sum''$ represents that both the first and last terms are halved. Notice each coefficient of $\mathbf{F}_k^{i-1}$ is obtained through the summation of $(N + 1)$ independent terms, each of which is an inner product of the force function $\mathbf{g}(\tau, x(\tau))$ and the Chebyshev polynomials $T_k(\tau)$ evaluated at the CGL points of equation (5). Furthermore, all the coefficient vectors are independent of each other, and can therefore be computed in parallel processors. Also, and most importantly, for problems where calculating the force vector function $\mathbf{g}(\tau, x(\tau))$ is time consuming, significant time performance improvement can be achieved by simultaneously computing $\mathbf{g}(\tau_j, x(\tau_j))$ at each τ_j on $(N + 1)$ parallel processors.

Assuming the solution at the i^{th} step is denoted $\mathbf{x}^i(\tau)$, Picard iteration provides the recursion to calculate $\mathbf{x}^i(\tau)$ as a Chebyshev polynomial approximation over the entire time interval as

$$\mathbf{x}^i(\tau) = \mathbf{x}_0 + \sum_{r=0}^{N}{}' F_r^{i-1} \int_{-1}^{\tau} T_r(s)\, \mathrm{d}s \equiv \sum_{k=0}^{N}{}' \beta_k^i T_k(\tau) \tag{8}$$

The coefficient vectors for the updated trajectory expressed below in equations (9–11), are obtained directly from recollecting the term-by-term analytical inte-

gration of equation (8), and imposing the initial boundary conditions. The derivation of these formulations can be found in Bai's dissertation [4].

$$\beta_k^i = \frac{1}{2k}(F_{k-1}^{i-1} - F_{k+1}^{i-1}) \quad k = 1, 2, \ldots, N-1 \tag{9}$$

$$\beta_N^i = \frac{F_{N-1}^{i-1}}{2N} \tag{10}$$

$$\beta_0^i = 2x_0 + 2\sum_{k=1}^{k=N}(-1)^{k+1}\beta_k^i \tag{11}$$

The updated coefficient vectors define the new trajectory approximation for use in integrand (equation (6)) for the next iteration ($i + 1$) (see Fig. 1 for the algorithm overview). Thus the solutions are iteratively improved until some accuracy requirements are satisfied. To account for the nonlinearity issues, the stopping criterion we choose is to require both the maximum difference (among all the N + 1 CGL nodes) between solutions $\mathbf{x}^i(\tau)$ and $\mathbf{x}^{i-1}(\tau)$ and the maximum difference between solutions $\mathbf{x}^i(\tau)$ and $\mathbf{x}^{i+1}(\tau)$ are less than some tolerance.

Instead of adopting a term by term scalar process to solve for the state at the $(N + 1)$ CGL nodes, the $(N + 1)$ Chebyshev coefficients, and the updated $(N + 1)$ Chebyshev coefficients, we have developed a compact vector-matrix approach to implement MCPI methods, which is shown in Fig. 2 and the derivation of the matrices can be found in Bai's dissertation [4]. The basis functions arising from the process are collected in the matrices

$$C_x \equiv \mathrm{TW} = \begin{bmatrix} T_0(\tau_0) & T_1(\tau_0) & \ldots & T_N(\tau_0) \\ T_0(\tau_1) & T_1(\tau_1) & \ldots & T_N(\tau_1) \\ \vdots & \vdots & \vdots & \vdots \\ T_0(\tau_N) & T_1(\tau_N) & \ldots & T_N(\tau_N) \end{bmatrix} \begin{bmatrix} \tfrac{1}{2} & 0 & \ldots & 0 \\ 0 & 1 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 \end{bmatrix} \tag{12}$$

$$C_\alpha \equiv \mathrm{RSTV} \tag{13}$$

$$S = \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{2}{3} & \frac{1}{4} & -\frac{2}{15} & \ldots & (-1)^{N+1}\frac{1}{N-1} \\ 1 & 0 & -1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & -1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \ldots & 1 & 0 & -1 \\ 0 & 0 & 0 & 0 & \ldots & 1 & 0 \end{bmatrix} \tag{14}$$

where the r^{th} ($r = 2, 3, \ldots, N - 1$) column of the first row has a form as

$$S[1, r] = (-1)^{r+1}\left(\frac{1}{r-1} - \frac{1}{r+1}\right) \tag{15}$$

And the diagonal matrices R and V are defined as

$$R = \mathrm{diag}\left(\left[1, \frac{1}{2}, \frac{1}{4}, \ldots, \frac{1}{2(N-1)}, \frac{1}{2N}\right]\right) \tag{16}$$

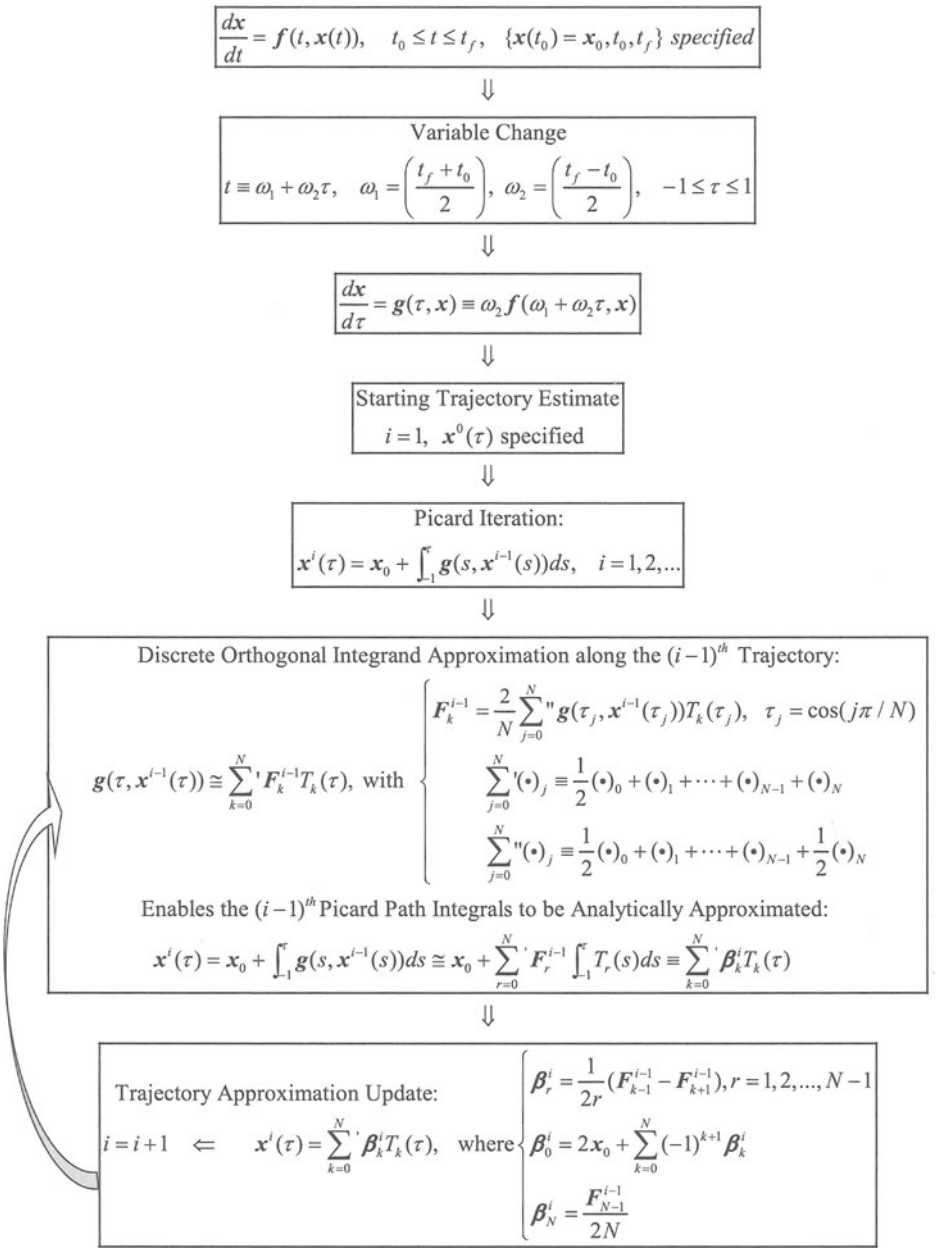

FIG. 1. MCPI Iteration for Solution of Initial Value Problems.

$$V = \text{diag}\left(\left[\frac{1}{N}, \frac{2}{N}, \frac{2}{N}, \ldots, \frac{2}{N}, \frac{1}{N}\right]\right) \tag{17}$$

Notice that the C_x and C_α matrices and the product $C_x C_\alpha$ are constant (once N is selected), so all computations of inner product can be efficiently precomputed. Furthermore, the eigenstructure of the matrix product $C_x C_\alpha$ is of crucial important in analyzing convergence of MCPI methods.

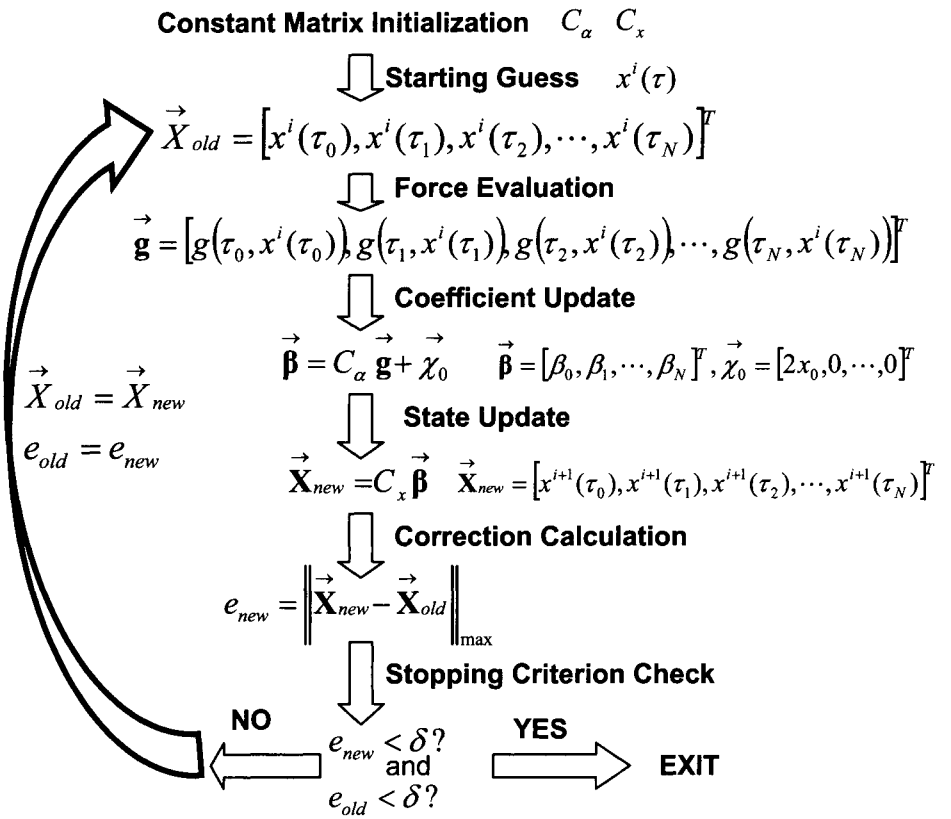

FIG. 2. Vector-Matrix Form of MCPI for Solution of Initial Value Problems.

Second Order MCPI Approach

Consider a second order ODE described by

$$\frac{d^2\mathbf{x}}{dt^2} = f(t, \mathbf{x}, \dot{\mathbf{x}}) \qquad \mathbf{x}(t) \in R^n \quad \mathbf{x}(t_0) = \mathbf{x}_0 \quad \dot{\mathbf{x}}(t_0) = \dot{\mathbf{x}}_0 \tag{18}$$

The MCPI formulation presented in the last section can solve this problem after we introduce a new state variable to transform equation (18) to a system of $2n$ first order equations. Recently, we have developed a cascaded MCPI formulation, i.e., a second order MCPI approach, which solves second order ODEs directly and also can be generalized to systems described by higher order differential equations. We emphasize this approach can solve problems with differential equations dependent on the velocity, whereas many of the most efficient RKN methods such as RKN12(10) cannot. In lieu of equations (3) and (4), we have the transformed version of equation (18)

$$\frac{d\upsilon}{d\tau} = \mathbf{g}(\tau, x, \upsilon) \qquad \frac{d\mathbf{x}}{d\tau} = \boldsymbol{\upsilon} \tag{19}$$

Similar to equation (4), the acceleration along the $(i-1)^{th}$ trajectory approximation is integrated to velocity from equation (20)

$$\boldsymbol{v}^i(\tau) = \boldsymbol{v}_0 + \int_{-1}^{\tau} \mathbf{g}(s, \mathbf{x}^{i-1}(s), \boldsymbol{v}^{i-1}(s))\, \mathrm{d}s \qquad i = 1, 2, \ldots \tag{20}$$

Importantly, the position vector is obtained, not from Picard iteration, but by direct integration of $v^i(\tau)$ to position through using the exact kinematic equation (the second of equation (19))

$$\mathbf{x}^i(\tau) = \mathbf{x}_0 + \int_{-1}^{\tau} \boldsymbol{v}^i(s)\, \mathrm{d}s \tag{21}$$

Notice the approximation errors incur at the velocity level in equation (20) when we expand the velocity trajectory and the integrand of equation (20) in Chebyshev basis functions and carry out the integrals of equation (20) term by term—then no further approximation is required. An exactly consistent position approximation is derivable from term by term integration of the linearly contained velocity in the integrand of equation (21). In the equivalent first order Picard iteration, the position is obtained by modeling the position and velocity independently, resulting in $2n$ approximations in the integrand of equation (4) with $\{\mathbf{x} \in R^n \Rightarrow \mathbf{z} \in R^{2n}\}$, constrained through the state variable definitions (so that velocity approximations implicitly know that they are the derivative of position). However, in the above cascaded formulation the velocity vector approximation directly dictates the corresponding coefficients for the position vector through the kinematic constraint implicitly in taking the integral of equation (21). Using the matrix-vector form of the MCPI approach, computation of position approximation from velocity simply amounts to a matrix multiplication with and addition of invariant (computed once) coefficient matrices C_x, C_α as

$$\mathbf{V}^i = C_x C_\alpha g(\vec{\mathbf{V}}^{i-1}) + C_x \Theta_{v0} \tag{22}$$

$$\mathbf{X}^i = \frac{t_f - t_0}{2} C_x C_\alpha(\vec{\mathbf{V}}^i) + C_x \Theta_{x0} \tag{23}$$

where the i^{th} step state trajectory evaluated at the $(N+1)$ CGL nodes has been represented by vectors $\mathbf{X}^i = [x(\tau_0), x(\tau_1), x(\tau_2), \ldots, x(\tau_N)]^{\mathrm{T}}$, and $\mathbf{V}^i = [v(\tau_0), v(\tau_1), v(\tau_2), \ldots, v(\tau_N)]^{\mathrm{T}}$. The initial condition vectors are contained in the vectors $\Theta_{x0} = [2x_0, 0, 0, \ldots, 0]^{\mathrm{T}} \in R^{N+1}$ and $\Theta_{v0} = [2v_0, 0, 0, \ldots, 0]^{\mathrm{T}} \in R^{N+1}$. The difference between using (i) an Picard iteration simultaneously for both position and velocity and (ii) a cascaded Picard iteration for velocity (equations (20) and (22)) and subsequent integration to get position (equations (21) and (23)) may at first look minor, however this approach is computationally much more attractive and is also more accurate. Note that this efficiency and accuracy advantage is enjoyed on each step, and therefore the entire Picard iteration is accelerated accordingly.

Convergence Analysis

Because of the accumulation of round off and approximation errors during the iterations (when a finite order of Chebyshev polynomial is used to approximate

solutions), the convergence domain of MCPI methods is different from the ideal conditions under which Picard iteration theoretically converges (Lipschitz continuity). Establishing a rigorous convergence domain of MCPI methods applicable for general nonlinear systems is not possible by any known approach. To obtain some essential insight we first use a linear scalar problem as an example to show that the global convergence of MCPI methods is not generally guaranteed, and we then address the practical approaches to enlarge the convergence domain. Consider a scalar linear dynamic system

$$\frac{dx(t)}{dt} = cx(t) \tag{24}$$

The i^{th} step position history evaluated at the $(N+1)$ CGL nodes is represented by a vector $\mathbf{X}^i = [x(\tau_0), x(\tau_1), x(\tau_2), \ldots, x(\tau_N)]^{\mathrm{T}}$ and an initial condition vector is defined as $\Theta_{x0} = [2x_0, 0, 0, \ldots, 0]^{\mathrm{T}} \in R^{N+1}$. The matrix-vector form of MCPI (Fig. 2) leads to the recursive solution

$$\mathbf{X}^i = \left[\frac{t_f - t_0}{2} cC_xC_\alpha\right]\mathbf{X}^{i-1} + C_x\Theta_{x0} \tag{25}$$

It is known from linear system theory that this sequence is convergent to a fixed point only if all the eigenvalues of matrix $\left[\frac{(t_f - t_0)}{2}\right] cC_xC_\alpha$ are within a unit circle (i.e., equation (25) must be a contraction mapping, to converge to a fixed point). Notice that the scalars appear multiplicatively and therefore simply scale the maximum eigenvalues of C_xC_α, leading to an attractive analysis for this simplest case of a linear problem. Thus the convergence of the MCPI method is dependent on the dynamical system characteristics c, the length of the time interval $(t_f - t_0)$, and the matrix multiplication C_xC_α, which only depends on the order of the Chebyshev polynomial used. For convergence, we require

$$\left|\frac{t_f - t_0}{2} c\lambda_{\max}(C_xC_\alpha)\right| < 1 \quad \text{or} \quad |t_f - t_0| < \frac{2}{|c||\lambda_{\max}(C_xC_\alpha)|} \tag{26}$$

Remarkably, we can see that this identical invariant (given N) matrix C_xC_α appears multiplicatively in the nonlinear generalization in vector-matrix notation (see Fig. 2), therefore, during the terminal iterations of a convergent solution process, we can expect this type of eigenanalysis to give approximate behavior useful in a more general setting. In fact, for a vector time varying nonlinear system, the above bound changes only with the scalar c being replaced by the infinity norm of the Jacobian. Notice that C_xC_α depends solely on the choice of Chebyshev basis functions, the nodal pattern selected, and the degree of the approximation. Therefore C_xC_α is invariant, can be computed once, and the eigenanalysis can be studied once for all N and applies in all subsequent MCPI solutions.

The maximum eigenvalues of C_xC_α are shown in Fig. 3. We found for small N ($N < 40$), this value decays approximately from 0.7 to 0.054, almost linearly on a log–log scale. Thereafter, for $N > 40$, this value remains approximately constant at 0.054. This gives rise to the maximum interval length as $|t_f - t_0|_{\max} = |(2/c)(1/\lambda_{\max}(C_xC_\alpha) \approx 37/c|$. Although this condition guarantees convergence of the MCPI method, for a fixed N, it does not guarantee that N is sufficiently high to give an accurate approximation of the solution. It is fortunate, as is evident in Fig. 3, that

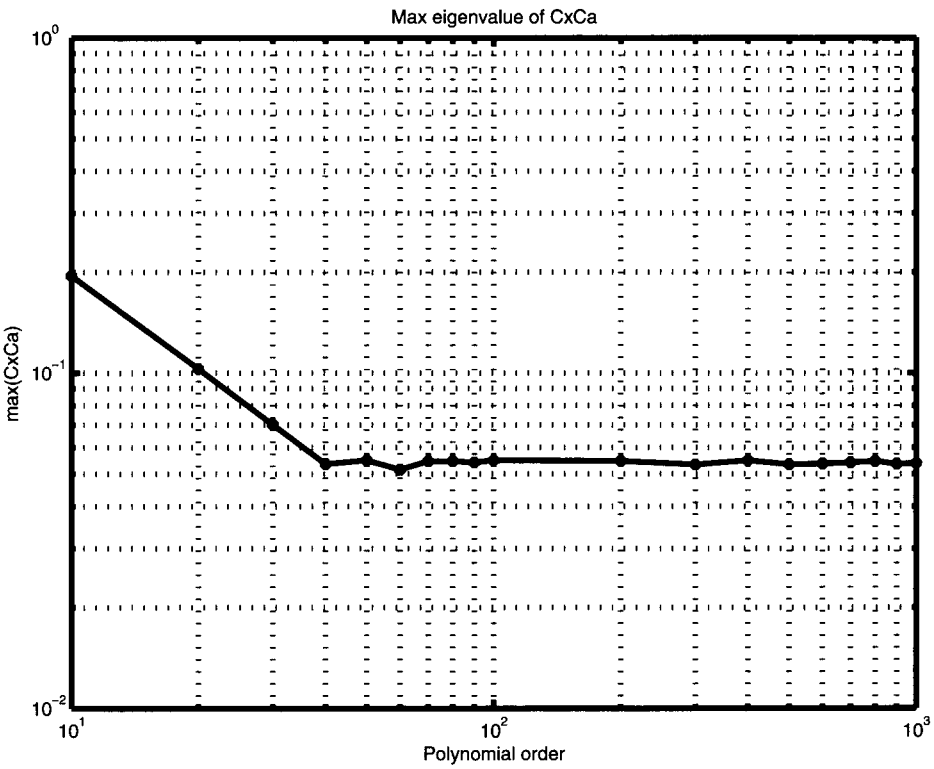

FIG. 3. Maximum Eigenvalue of C_xC_α.

convergence does not degrade for large N, or put another way, $N > 40$ can be adjusted to achieve high solution precision, without affecting the convergence of MCPI method.

For convergence insight into the case of second order differential equations, we consider $\mathrm{d}^2x(t)/\mathrm{dt}^2 = cx(t)$. The updated velocity equation is

$$\mathbf{V}^i = c\left(\frac{t_f - t_0}{2}\right)^2 C_xC_\alpha C_xC_\alpha \mathbf{V}^{i-1} + c\frac{t_f - t_0}{2} C_xC_\alpha C_x\Theta_{x0} + C_x\Theta_{v0} \tag{27}$$

We found that the maximum eigenvalues of $C_xC_\alpha C_xC_\alpha$ decreases from 0.038 to 0.003 as N increases from 10 to 40, analogous to the case for the first order system. For all the $N > 40$, the maximum eigenvalues are asymptotically approach about 0.003. Thus the convergence condition for the second order MCPI method is approximately $c(t_f - t_0)^2 < 4/0.003 \approx 1333$. We mention the significant truth that this represents a two order of magnitude increase over the size of region $c(t_f - t_0)^2 < 12$ in Sommeijer's study [22].

Although these linear analyses tell us that MCPI methods only converge on a finite interval, we can anticipate using a piecewise approach to solve a significant family of IVPs over an arbitrarily large time domain. The initial conditions on the subsequent segments should be the final state values from the previous segment. This may sound similar to the concept of the step size control used in forward integration methods such as Runge-Kutta or analytical continuation methods. However, the step size used by MCPI methods is typically a much larger finite

interval than the steps used by typical numerical methods, as will be shown in the following examples. Furthermore, compared with the forward integration methods in which the integration errors are typically increasing with time in a secular unstable fashion, we anticipate that better stability/accuracy can be achieved from using MCPI methods because, qualitatively, the largest errors from MCPI methods usually appear in the middle of the interval and the smallest errors are at the ends where adjacent (successive) segments are joined. The fundamental reason for this special characteristic of MCPI methods is because of the chosen Chebyshev basis functions and CGL nodes which are denser at the boundaries and sparser in the middle.

Numerical Examples

All computations underlying this article were done in a conventional PC. The settings of the computer and the development environment used are: Intel(R) Pentium(R) D CPU 3.4GHz, 3.4GHz, 2.0GB of RAM; Windows XP Operating System; MATLAB R2009b. The computation time shown below is the average CPU time of ten running cases. Also, except for the first example where we use the first order MCPI method to solve a first order ODE, second order MCPI methods are used for all the other examples.

Example 1: A First Order Nonlinear System

Consider a dynamic equation

$$\frac{\mathrm{d}y}{\mathrm{d}t} = f(y, t) = \cos(t + \varepsilon y) \qquad t_0 = 0 \quad t_f = 256\pi \quad y(t_0) = 1 \quad \varepsilon = 0.001 \tag{28}$$

Fukushima suggested this problem, which has a closed-form solution, as a benchmark with a known truth for conducting accuracy studies [37]. We first compare the results by using MCPI methods implemented in MATLAB and by using ODE45, which is a Runge-Kutta 4–5 method implemented in MATLAB. For more significant nonlinear problems below, we use more sophisticated (and efficient) integrators as the basis for comparison. The results are shown for this first example in Fig. 4.

The MCPI method uses a Chebyshev polynomial of order $N = 1500$ to approximate the solutions along the entire interval. For this tuning, the MCPI solutions have more than one order of magnitude better accuracy than the ODE45 solutions and the CPU time using ODE45 is about 80 times slower than the CPU time using the MCPI method. We emphasize that other integrators solving first order ODEs may yield better performance than ODE45 in solving this problem, but it is promising to see that the MCPI method achieved sufficient accuracy and significant speedup in the same time. We further note that orders (N) up to of several thousand are feasible with MCPI, without numerical difficulty, owing to the orthogonality properties (no matrix inverses or other linear algebra ill-conditioning opportunities to lose significant digits) and, especially, highly efficient recursions based on simple inner products. The optimal order is typically much less, but obviously high order approximation in numerical integration now takes on a new meaning. As we will show later, these solutions have not taken advantage of the fact that the long intervals can be subdivided, which will reduce

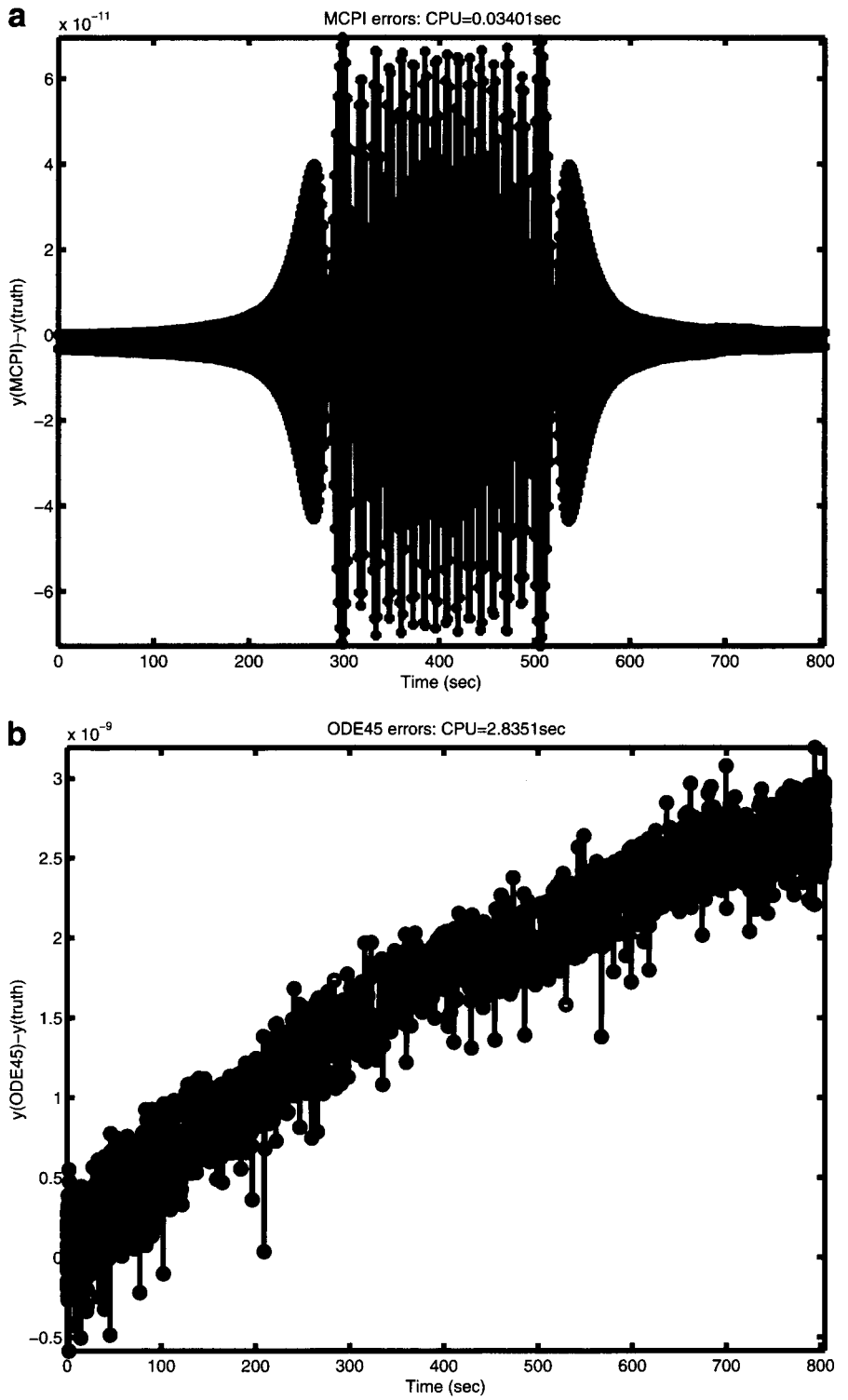

FIG. 4. Integration Errors and CPU Time for Example 1.

the order of the required polynomial approximations thus more speedup can be achieved by the MCPI methods when implemented on a serial machine. Furthermore, if parallel computation environment is available, more speedup will also be obtained because the un-coupled function evaluations matrix operations can be distributed to different processors. We also notice that although the errors from the reference ODE45 solution have a typical secular increase, which is a pattern common to all forward integration methods, the errors using the MCPI method have the maximum values near the middle of the interval and the smallest errors at the boundaries. The fundamental reason is because of the CGL nodes, which are dense at the boundaries and sparse in the middle. This special feature makes MCPI methods more attractive than the forward integration methods in reducing the global errors for long time integrations, for which the solutions in different segments are patched together at the terminal points of each solution interval where the errors are typically smallest. Additionally, we note convergence can be obtained up to some problem dependent maximum final time. For a linear problem, this maximum can be determined. For nonlinear problems, such as this one, approximation or adaptive tuning is required. The interval for a practical convergence is found to be significantly greater than 256π (about 128 oscillation periods). For qualitative purpose, we note that expanding equation (28) in ε leads to $dy/dt = \cos(t) - \varepsilon \sin(t)y + \ldots$, so the linear (in y) coefficient is bounded by $\pm\,\varepsilon$. Even though it is not rigorous, we can estimate from the above analysis of the constant coefficient linear system that convergence might be expected if $H < (2/\varepsilon)(1/\max(\lambda(C_x C_\alpha))$. Thus for the chosen polynomial $N > 100$, the convergence condition is approximately that H should be less than $2/(0.05 \times 0.001) \approx 40{,}000$. Although these estimations may be too optimistic, we verified excellent convergence was actually achieved if $H \leq 8000\pi \approx 5026.5$. Perhaps the most striking feature of this example is that very high precision can be achieved via MCPI over long time periods including many main period oscillations of a nonlinear system, whereas many time steps per period are required by all step-by-step integrators known to achieve comparable precision.

Example 2: A Second Order Nonlinear System

The following second order differential equation has the same analytical solution as the above first order example, and allows us to conduct accuracy studies for those integrators designed for second order systems

$$\frac{d^2y}{dt^2} = f(y, t) = -\sin(t + \varepsilon y) - \tfrac{1}{2}\varepsilon \cos(2t + 2\varepsilon y)$$

$$t_0 = 0 \quad t_f = 256\pi \quad y(t_0) = 1 \quad \dot{y}(t_0) = 1 \quad \varepsilon = 0.001 \tag{29}$$

Among the many convergent possibilities, we have tuned the second order MCPI method to use a Chebyshev polynomial of order 130 to approximate the solution over an interval length of 16π (eight periods of unperturbed oscillations), and found convergent solutions on the 16 segments of 16π duration that are patched together to generate the final solution. At the starting iteration, all the positions and velocities at the $N + 1$ CGL nodes are simply chosen from straight line solutions ensuing from the initial position and the initial velocity, thus a very poor starting guess is provided for the MCPI method so that the timing results are very

conservative. To provide a more meaningful comparison *vis à vis* relative efficiency, we adopt the 12th order Runge-Kutta-Nystrom algorithm RKN12(10) with adaptive step size control, which is widely regarded as one of the more powerful nonlinear differential equation solvers available. The errors and CPU times in seconds of MCPI and RKN12(10) are shown in Fig. 4. We can see with slightly better accuracy achieved by the MCPI solution, MCPI also obtained a speedup of about 30 over RKN12(10). This speedup is (in spite of the fact that the number of function evaluations was not substantially reduced) a consequence of the computational efficient recursive vector-matrix nature of the MCPI algorithm and large step size MCPI used. The RKN12(10) algorithm calls the function evaluation routine 14, 974 times, whereas totally MCPI takes 113 iterations, which leads to $113 \times (130 + 1) = 14,803$ function evaluations (remarkably, almost the same number in this case). Thus on a serial machine, even with a very poor starting solution estimate, MCPI requires essentially the same number of function evaluations as does RKN12(10). However, it is vitally important to recognize that the MCPI acceleration evaluations on all the CGL nodes are independent, because the entire path approximation is available at once on each iteration. In an ideal parallel environment where we can distribute the function evaluations on the $(N + 1)$ CGL nodes onto $(N + 1)$ processors, the theoretical speedup factor is 131, and can be expected to approach that theoretical limit if 131 or more cores are available, because little shared memory is involved. Figure 5 shows the errors are in the 11th significant figure for both solutions, although the MCPI solution has about ¼ the error norm of the RKN12(10) solution. To recap, the speedup achieved on a serial processor was 30, the theoretical speedup on a parallel processor with over 130 cores is two additional orders of magnitude for this problem. Impressive potential exists, if these results for "toy" idealized problems extend to the problems of orbit mechanics. In the results presented below, the test cases to date indicate that these speedups are typical for the more nonlinear problems of central practical interest. Additionally, comparing the computational time and accuracy of this tuned second order MCPI with the previous first order MCPI using one segment, we see the benefit to use the second order formulation and also the potential for better accuracy and more speedup when careful tuning is applied to the MCPI methods, see Figs. 4(a) and 5(a) for this comparison.

Example 3: Integration of Unperturbed Keplerian Motion (Natural Second Order System)

Although a classical problem that has been used very often for performance comparison of ODE solvers is a planar two-body problem [12, 13, 32–34], we choose to use an example that integrates a three dimensional near circular orbit for one week to help us draw more practical insight. The dynamic equations are

$$\ddot{x} = -\frac{\mu}{r^3}x \qquad \ddot{y} = -\frac{\mu}{r^3}y \qquad \ddot{z} = -\frac{\mu}{r^3}z \qquad r = \sqrt{x^2 + y^2 + z^2} \tag{30}$$

The six classical orbital elements are: $a = 6644.75$ km, $e = 0.01$, $i = 68°$, $\Omega = 92°$, $\omega = -160°$, $T_p = 5.3905 \times 10^3$ sec. Both the MCPI methods and RKN12(10) are tuned such that sub-millimeter position accuracy, relative to the exact analytical solution, is achieved for the whole week. A Chebyshev polynomial of order 40 is chosen for the MCPI method and the segment length is selected to be 5,400 sec. The classical F&G method [38] provides an analytical truth that is used to calculate

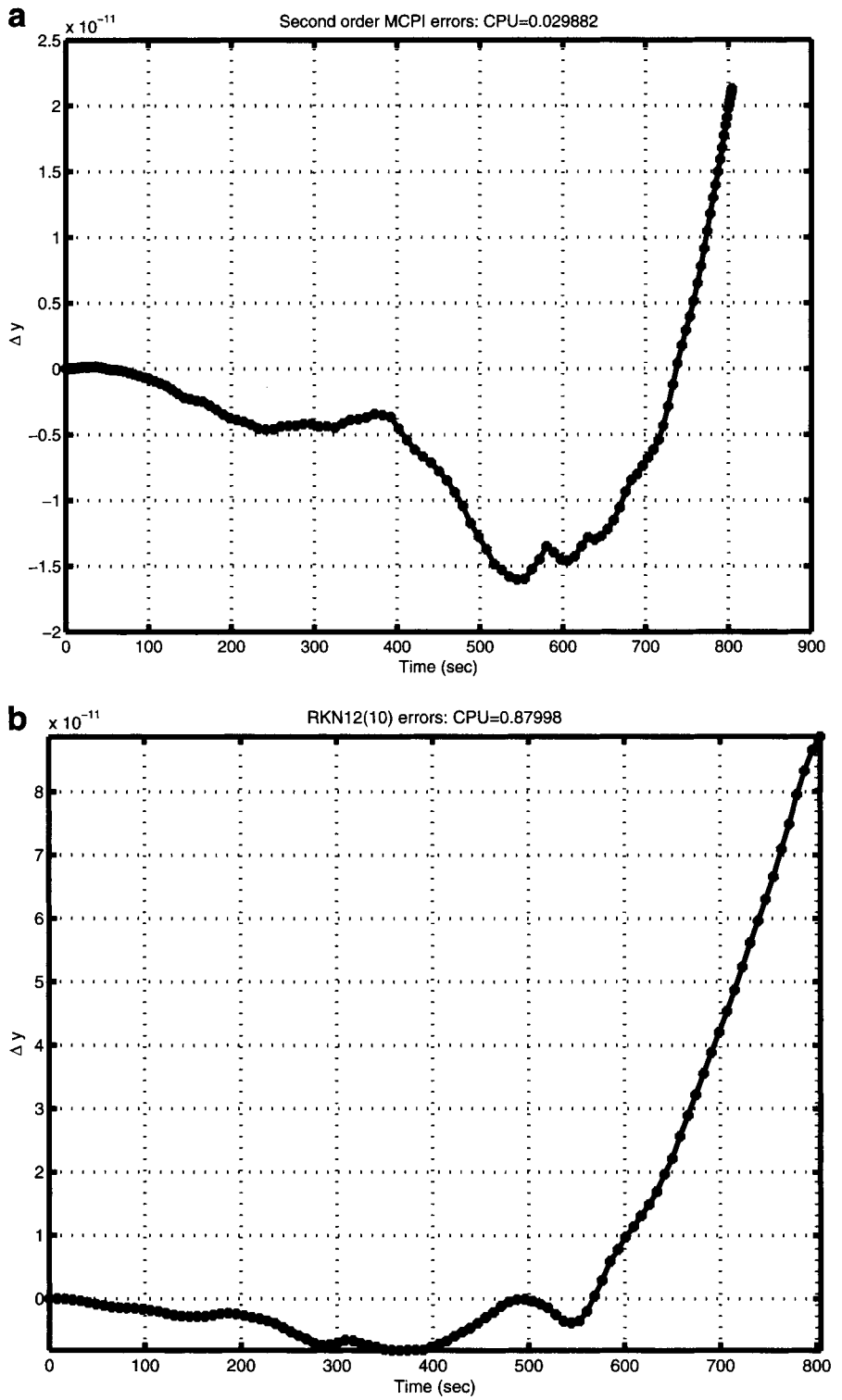

FIG. 5. Integration Errors and CPU Time for Example 2.

the solution errors, of which for the MCPI method and the RKN method are shown in Figs. 6 and 7. Several observations are summarized here. First, the computational time is 0.2639 sec for MCPI and 1.8882 sec for RKN12(10). Thus with slightly better accuracy in both position and velocity, MCPI achieved a speedup factor of seven. Second, RKN12(10) calls the differential equations 29,662 times. Using an initial starting solution that the position and velocity at all the CGL nodes are the same as the prescribed initial position and velocity, MCPI took 2465 iterations in total. Thus on a serial machine, the ratio of function evaluation of RKN over MCPI is $29662/(2465 \times 41) \approx 0.3$. However, in an ideal parallel environment where we can distribute the function evaluation on the $(N + 1)$ CGL nodes to $(N + 1)$ processors, the ratio is $29662/2465 \approx 12$. Third, the reason for the speedup of MCPI over RKN in a serial implementation lies in two aspects. The first reason is that the vector-matrix form of the MCPI approach is computationally very efficient. The second reason attributes to the large step size that MCPI can use. For RKN12(10), the minimum step size is 0.0465 sec, the maximum is 629.4089 sec, and the mean is 363.4615 sec that is about 7% of the one full orbit which is the step size that the MCPI method used. Notice there is a significant qualitative difference in approximating a path versus taking small steps along it! Last, we have gained some preliminary insight about how to tune the polynomial order and the segment length.

Figures 8 and 9 show the computational time and accuracy for the MCPI method when the orders are chosen from 40 to 300, and the segment lengths are chosen from about 0.1 of the orbit period up to 2.2 of the orbit period. We found the minimum computation time is 0.1847 sec, which is obtained when we choose $N = 50$ and segment length as 10,260 sec (about 1.9 orbit). The most time consuming settings are the cases where a low order Chebyshev polynomial is used to integrate a rather large segment. We also characterize the solution errors as the maximum global relative error

$$e = \max\left(\frac{|r(\text{MCPI}, t) - r(\text{FG}, t)|}{r(\text{FG}, t)}\right) + \max\left(\frac{|v(\text{MCPI}, t) - v(\text{FG}, t)|}{v(\text{FG}, t)}\right) \tag{31}$$

where the first argument indicates the method to compute the solution and FG denotes the classical analytical solution. The significant figures shown in Fig. 9 are defined as $\varepsilon = -\log 10(e)$. Figures 8 and 9 show that although we currently do not have a rigorous way to find the optimal settings for the minimum computational time, we have a large region where we could obtain more than eleven significant digits in less than one second of computational time, which is still significantly faster than RKN12(10)). The irregularity of the 11 to 12 significant digit contours is a consequence of the solution accuracy approaching the noisy precision limit associated with finite precision arithmetic. The RKN12(10) algorithm also experiences similar bumpy convergence when it approaches twelve digit accuracy. In this case, we have the choice over a large space of interval lengths and orders to achieve twelve digit accuracy, but of course, nine digit accuracy for orbit problems is typically considered sufficient for "engineering accuracy," because this already corresponds to centimeter precision. For runtime efficiency, the optimal region in this tuning space for serial machines is as near the top left boundary of Figs. 8 as accuracy allows, whereas, for parallel machines, it is near the top boundary but further to the right (we accept the longest practical conver-

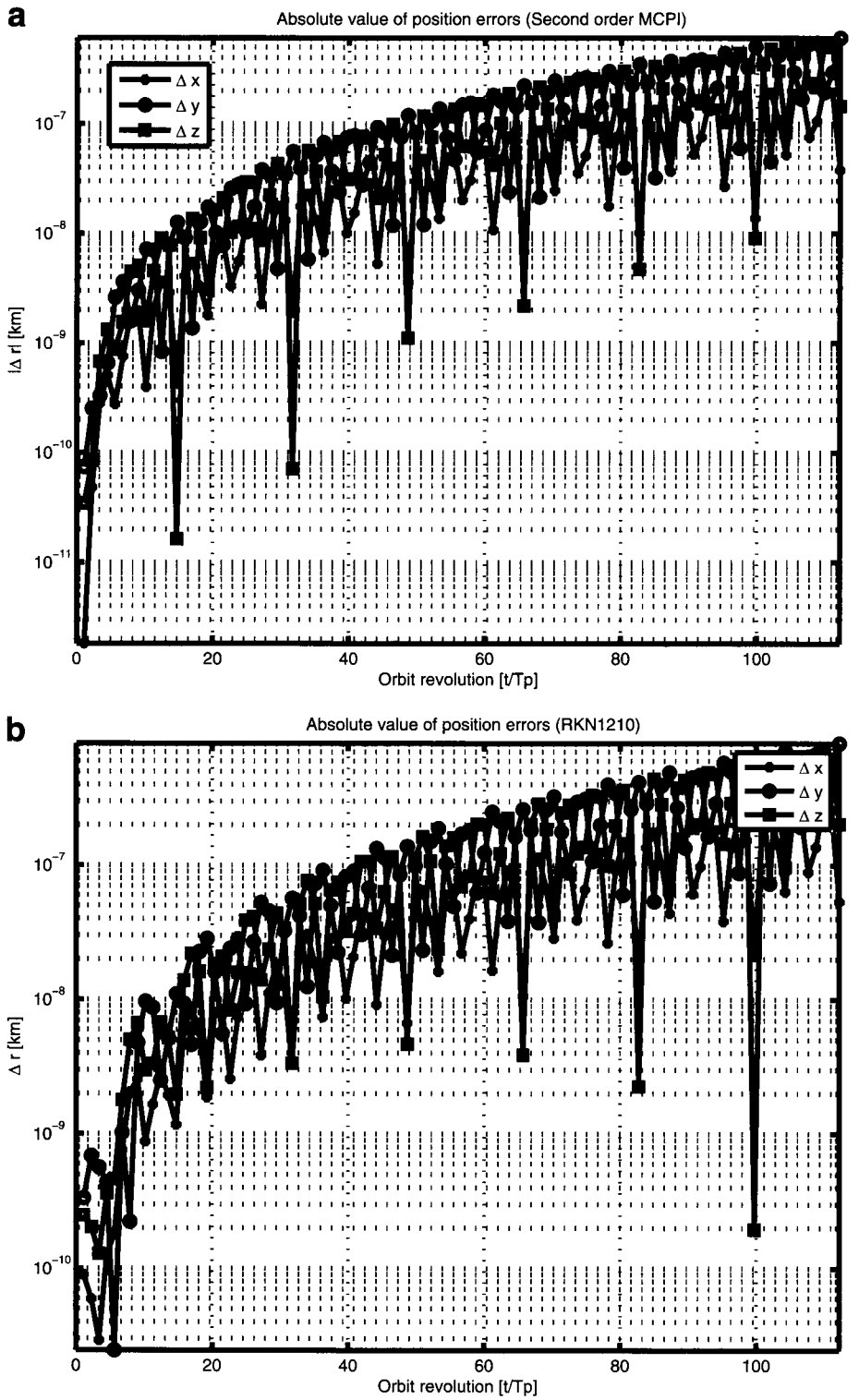

FIG. 6. Position Errors of MCPI and RKN12(10) for Example 3.

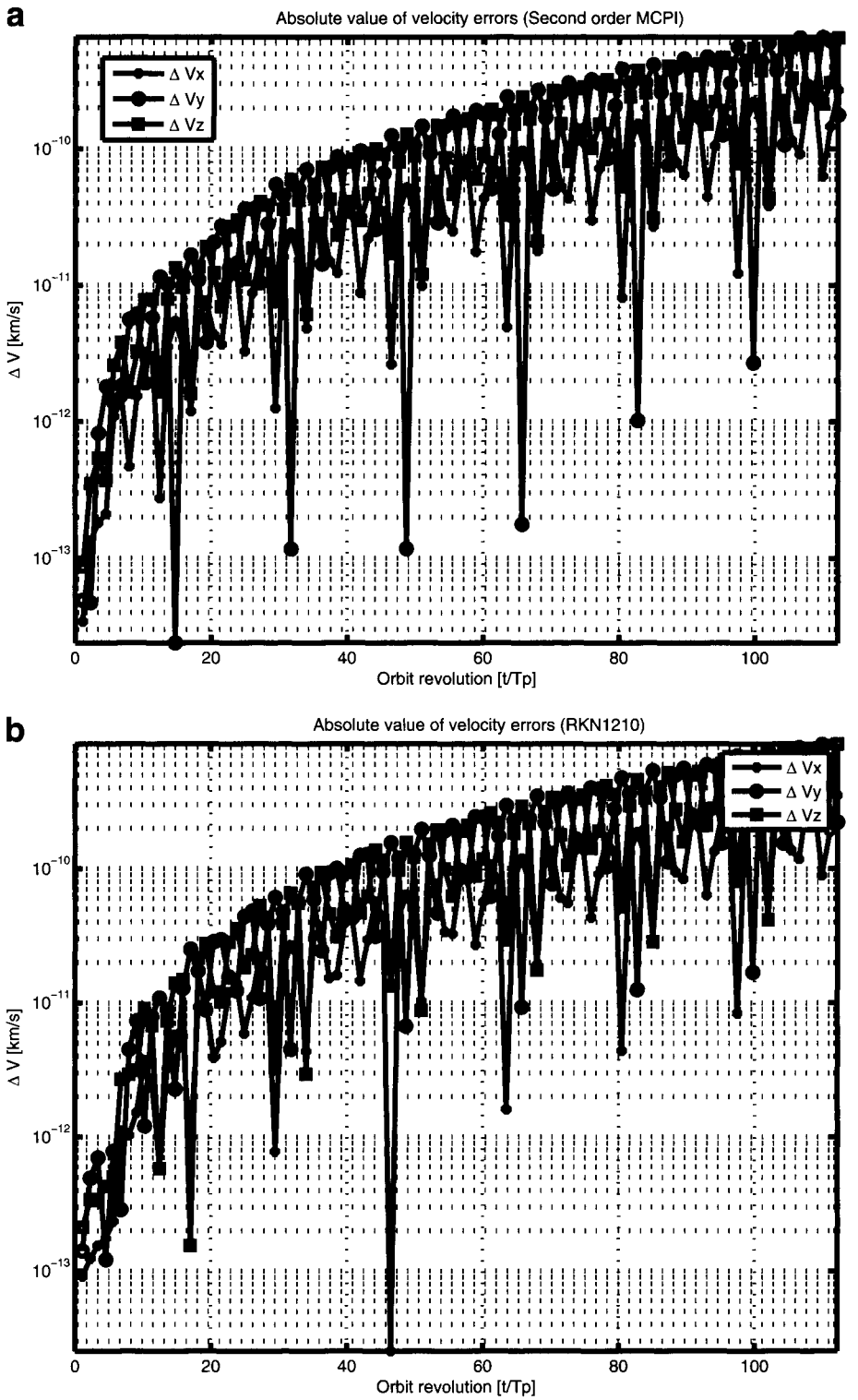

FIG. 7. Velocity Errors of MCPI and RKN12(10) for Example 3.

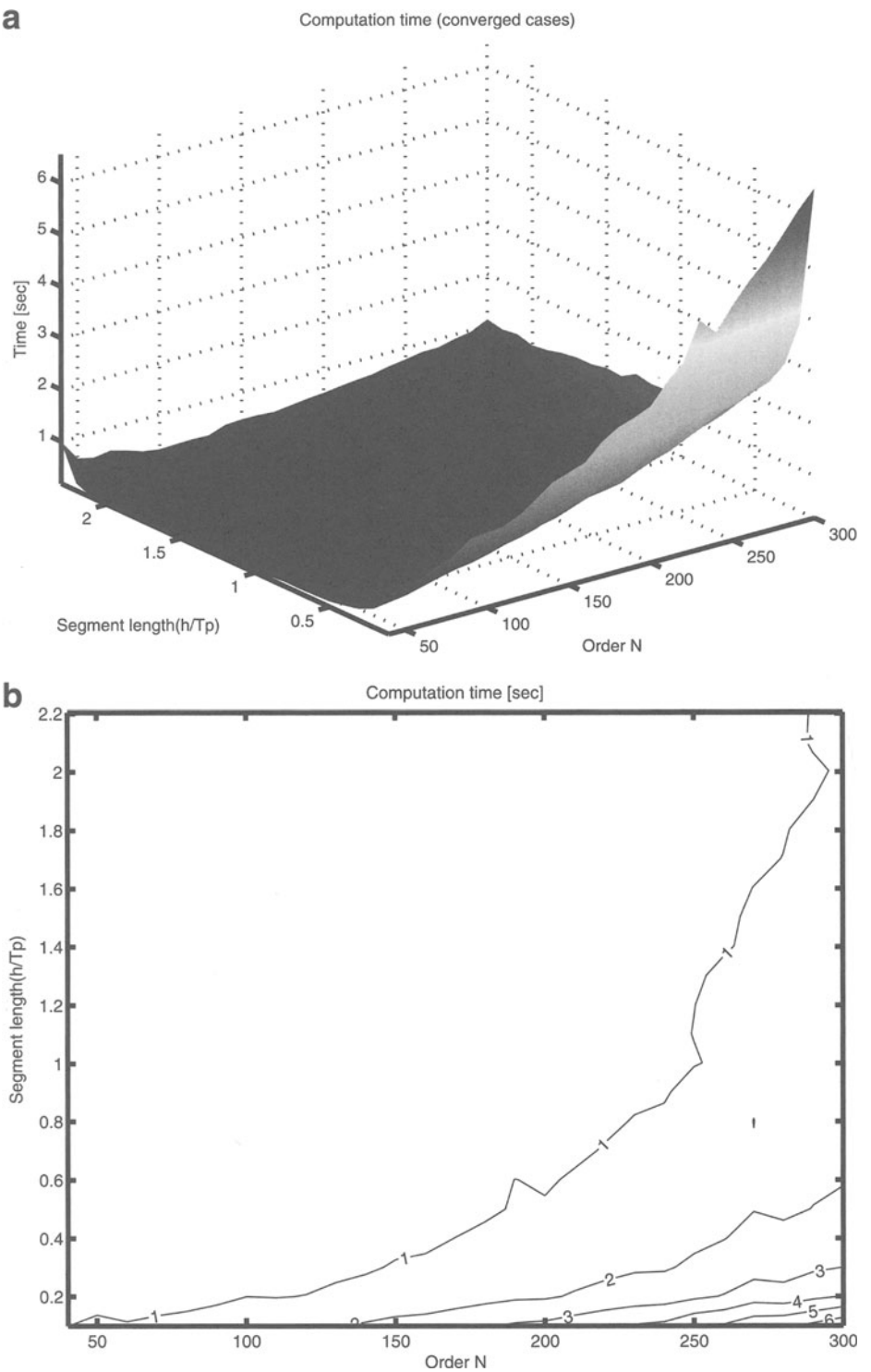

FIG. 8. Computation Time of MCPI for Example 3.

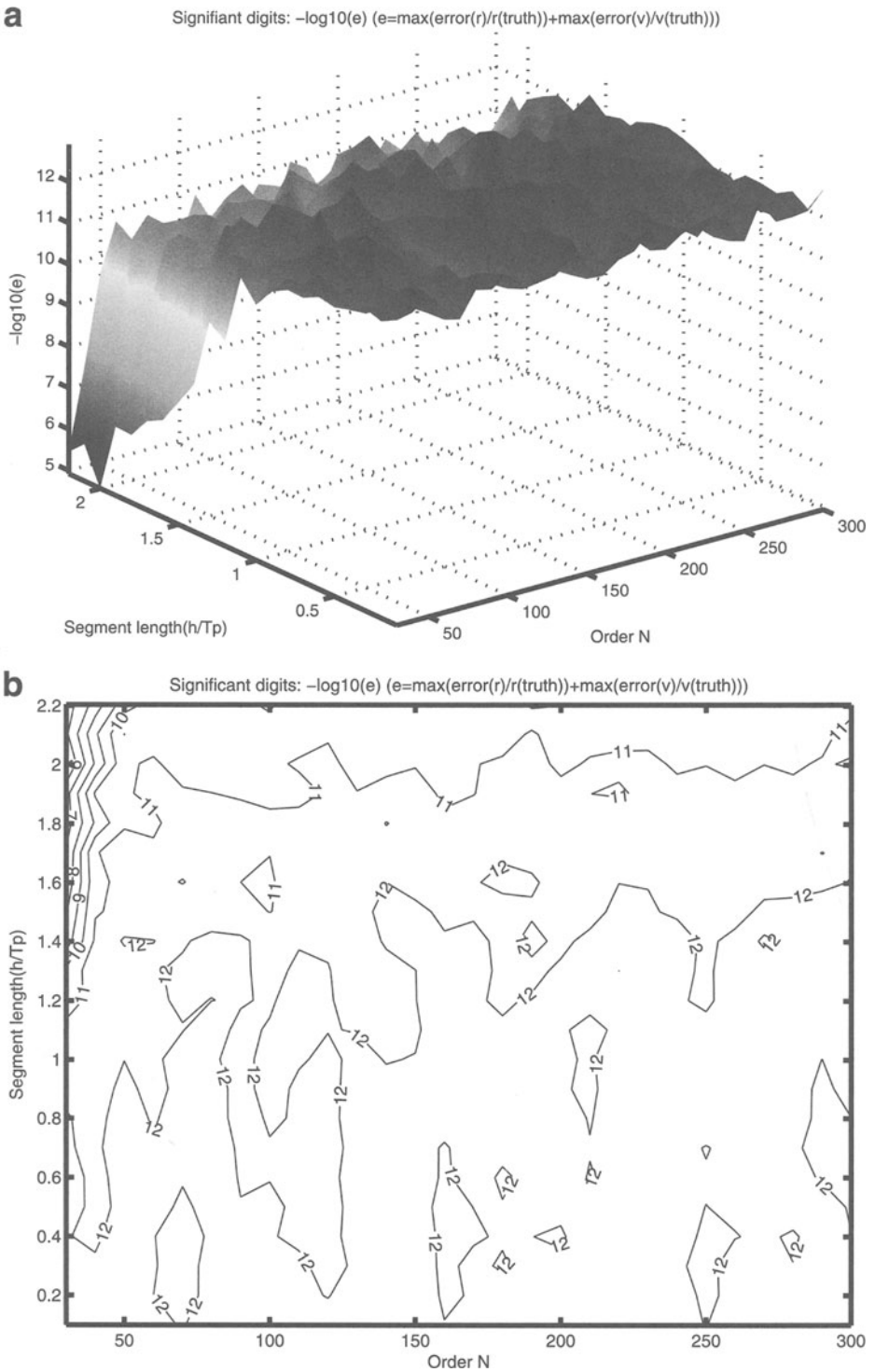

FIG. 9. Significant Digits of MCPI for Example 3.

gence interval, because the order can be adjusted, by moving right for larger N equal to the number of cores available (so the number $(N + 1)$ of function evaluations along each iterative trajectory can be carried out simultaneously to achieve a theoretical speedup of $(N + 1)$). The flatness of the accuracy surface permits a large space for adjustment to take full advantages of various parallel architectures. Additionally, we emphasize although we are targeting for high accuracy solutions in this article, there is a tradeoff between the computation time and solution accuracy, both of which are dependent on the chosen polynomial order, segment step size, and stopping criterion. The general rule is that higher order polynomials and stricter stopping criteria will lead to more accurate solutions but require longer computation time. Some experimental studies can be found in Bai's dissertation [4].

Propagating a Family of Perturbed Orbits

The ability to propagate satellite motion quickly and accurately is one of the major factors that affects the performance of tasks such as collision avoidance for space objects. For these tasks, numerical integration of the satellite motion with even more accurate and complicated perturbation force models has become necessary [39, 40]. Possibly a degree and order 36×36 or higher gravity model and a realistic atmospheric density model will be required to model perturbation accelerations in addition to the dominant force. In the following studies, we include the zonal harmonic perturbation forces up to the fifth order [38] in the dynamic models and investigate how the performance of the algorithms changes as the force model becomes more complicated. Including zonal harmonic perturbations up to the order of k leads to the dynamic equation as

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r} + \sum_{i=2}^{i=k} \mathbf{a}_d^i \tag{32}$$

where a_d^i is the i^{th} order zonal perturbation terms. We compare the computational time and the number of function evaluations when using MCPI and RKN12(10). We look at a low eccentricity problem as well as a high eccentricity problem. The purpose of considering the perturbations in this way is to assess the role that model complexity plays on the relative efficiency and accuracy of MCPI in comparison to existing methods. We also vary the eccentricity of the orbits from near circular to very eccentric, to assess the degree to which rapidly varying nonlinearity impacts the relative merits of the algorithms.

Example 4: Integration of Perturbed Orbits with Low Eccentricity ($e = 0.01$)

The initial conditions for this example are the same as those used in Example 3. For the MCPI method, the Chebyshev polynomial order is 40 and the segment length is 5,400 s, which have been tested in the unperturbed case to provide submillimeter position accuracy. For the four perturbed cases, although no analytical solutions are available, we have verified the relative energy changes for both methods are in the range of 10^{-13}. The computational times for the two methods are shown in Fig. 10(a) and the speedup results are shown in Fig. 10(b). The order "1" case is the unperturbed Example 3 we studied before and we include it here to illustrate the performance trend with respect to the complication level of the

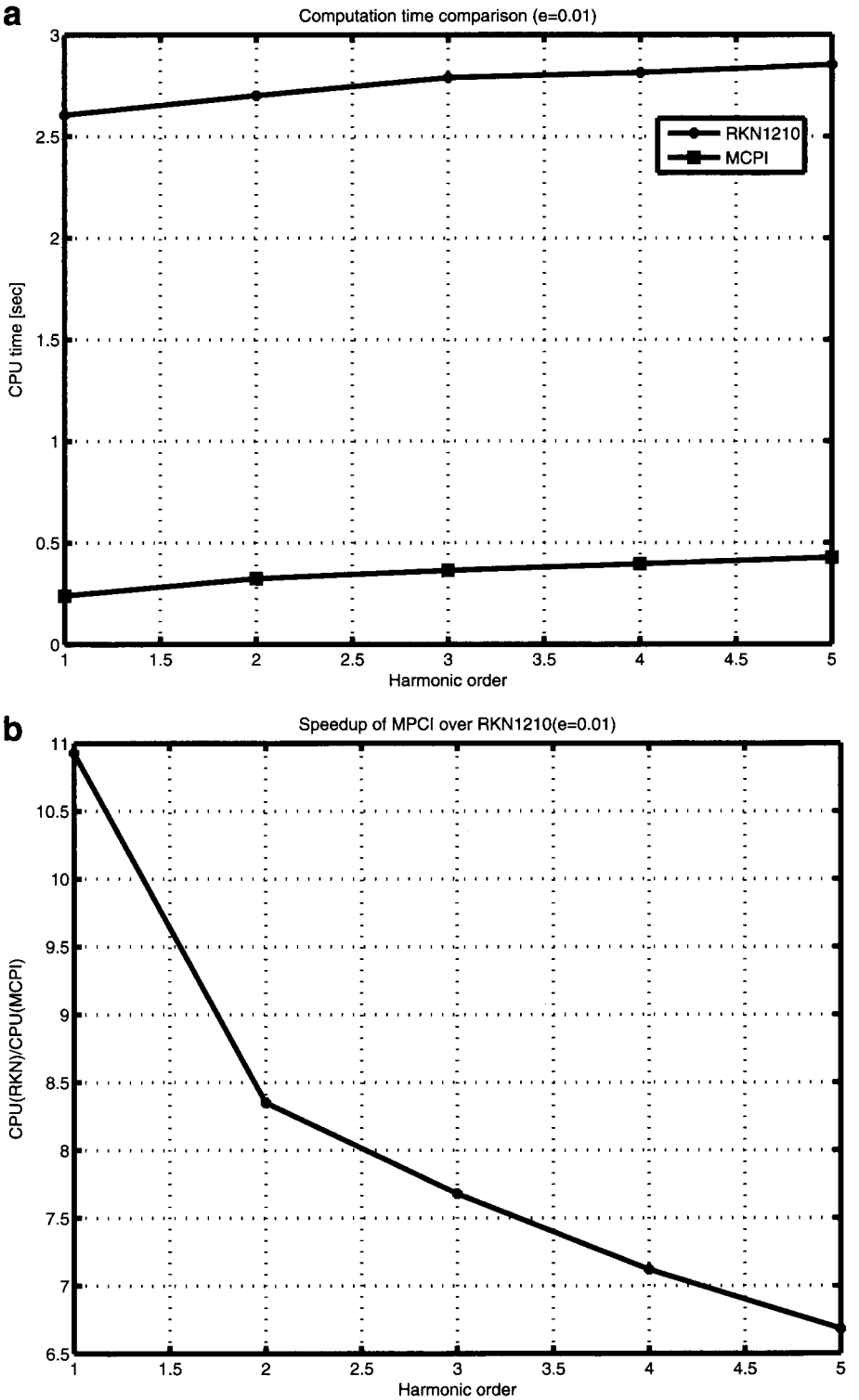

FIG. 10. Time Performance for Example 4.

perturbation models. Figure 10(b) shows that the MCPI method achieved six to eleven speedup over RKN12(10). Figure 11(a) shows that RKN12(10) calls about 30% of the number of function evaluations required by the MCPI method. Figure 11(b) shows that in an ideal parallel computation environment where we can distribute the force evaluation on the $(N + 1)$ CGL nodes onto $(N + 1)$ processors, RKN12(10) calls about twelve times of the number of function evaluations required by the MCPI method. Although the speedup achieved by the MCPI is shown to be decreasing on Fig. 10(b) in a serial implementation, we predict that the trend will change to be beneficial to the MCPI method on an advanced parallel machine because of the following two reasons. First, Fig. 11 shows that as more perturbation terms are included, the function call ratio of RKN12(10) over MCPI is increasing. Second, as we discussed earlier, the parameters for the MCPI method we used here are not the optimal settings. We anticipate more speedup can be achieved after we establish an adaptively tuning algorithm

Example 5: Integration of Perturbed Orbits with High Eccentricity ($e = 0.9$)

For this highly eccentric orbit, the six classical orbital elements are $a = 65{,}000$ km, $e = 0.9$, $i = 68°$, $\Omega = 92°$, $\omega = -160°$, $T_p = 1.6492 \times 10^5$ sec. For the MCPI method, the Chebyshev polynomial order is 45 except the segment passing the perigee, during which we use a polynomial of order 110. We have kept the segment length as 1/20 of the orbit. These settings have been tested in the unperturbed case to provide sub-millimeter position accuracy. For the perturbed cases, we have verified the relative energy changes for both methods are in the range of 10^{-13}. The computational time for the two methods are shown in Fig. 12(a) and the speedup results are shown in Fig. 12(b). Figure 12(b) shows that the MCPI method achieved more than one order magnitude of speedup over RKN12(10) and this speedup is higher than the one achieved for the low eccentric case. Figure 13(a) shows that RKN12(10) calls about 40% of the number of function evaluations required by the MCPI method whereas Fig. 13(b) shows that in an ideal parallel computation environment where we can distribute the force evaluation on the $(N + 1)$ CGL nodes onto $(N + 1)$ processors, RKN12(10) calls about twenty times of the number of function evaluations required by the MCPI method.

Conclusion

Through solving four exemplar problems, the presented MCPI methods have demonstrated orders of magnitude speedup and high precision relative to the state-of-the-art RKN12(10) algorithm. At a high level, we believe this is the first time that high precision orbits have been represented (with up to about twelve digit precision) with a single functional approximation. That such multi-orbit solution arcs can be obtained with computational efficiency comparable or superior (in a serial algorithm) to well-established high order numerical differential equation solvers is remarkable in and of itself. However, the most significant truth is that the MCPI algorithm is ideally suited for parallel computation, and this feature offers the near term prospect of better than two orders of magnitude speedup for high precision orbit integration.

We remark currently the tuning of the MCPI algorithm is a trial and error process. We are developing an adaptive tuning mechanism which will bring more speedups and accuracy advantages to MCPI methods. We are also programming

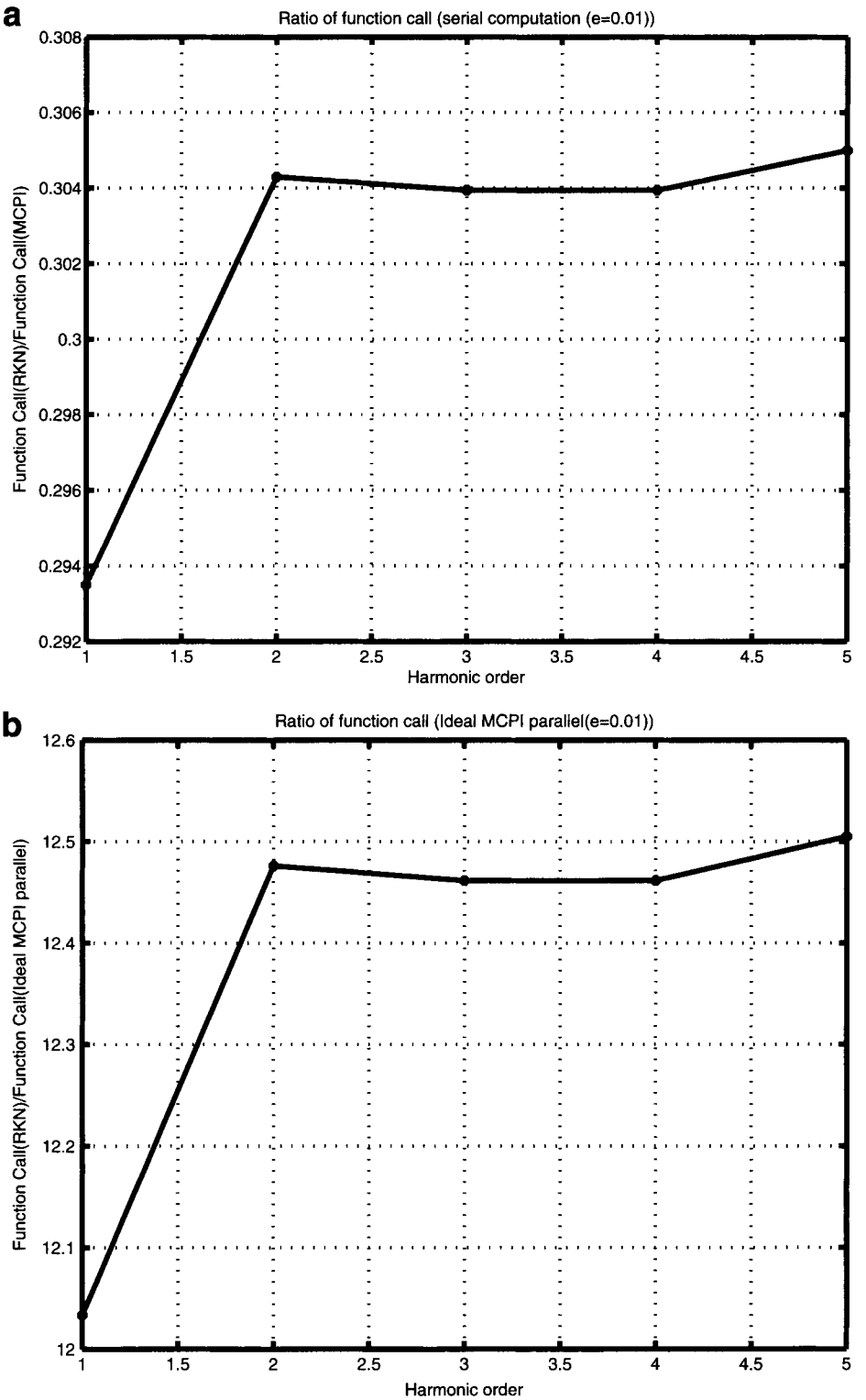

FIG. 11. Number of Function Call for Example 4.

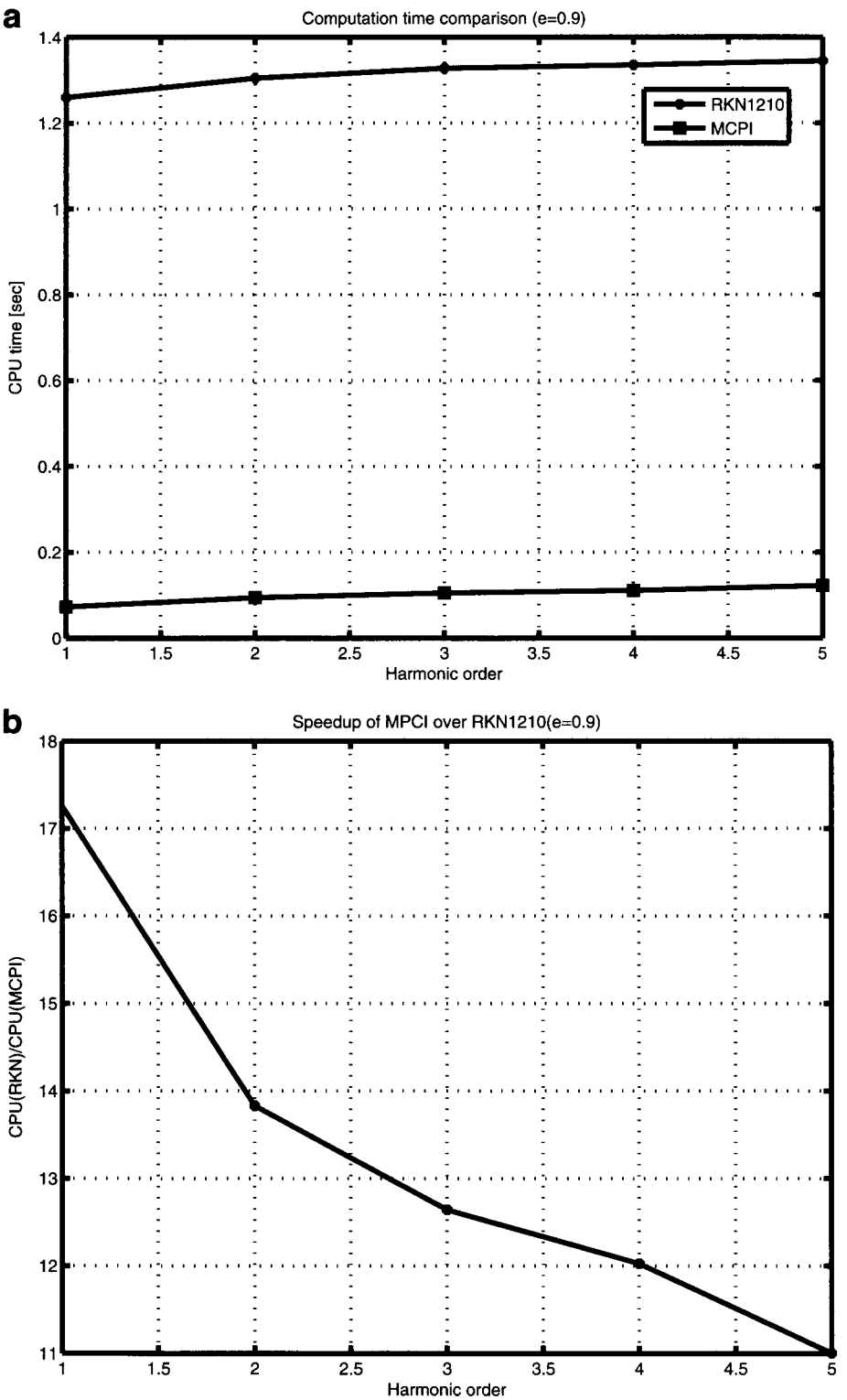

FIG. 12. Time Performance for Example 5.

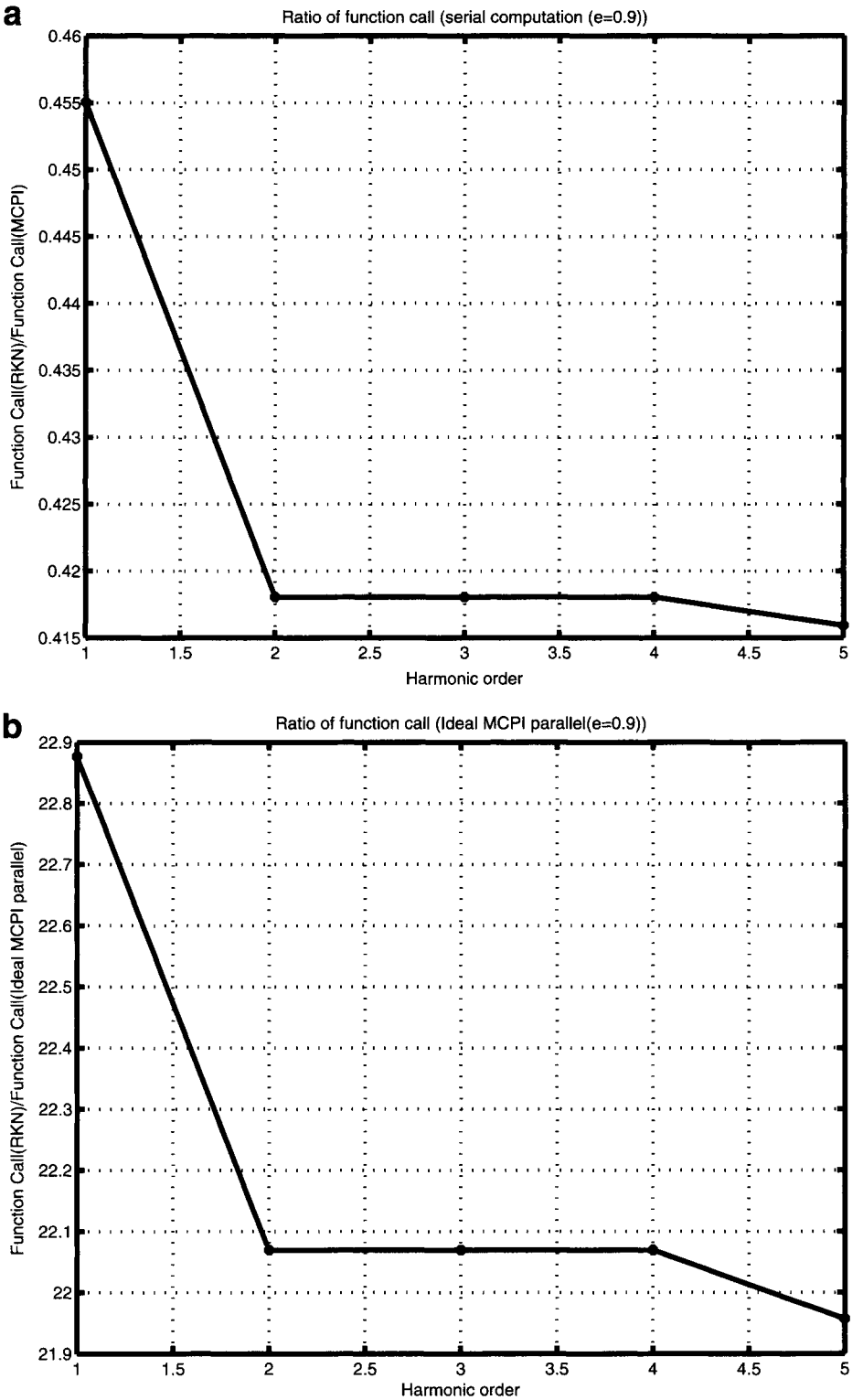

FIG. 13. Number of Function Call for Example 5.

MCPI methods using the newly released NVIDIA Tesla, which has peak performance of 515 GFLOPS for double precision floating point computation, and the results will be published in a future study.

References

[1] GEAR, C.W. "Parallel Methods for Ordinary Differential Equations," *Calcolo*, Vol. 25, Mar. 1988, pp. 1–20.

[2] BAI, X. and JUNKINS, J.L. "Solving Initial Value Problems by the Picard-Chebyshev Method with NVIDIA GPUS," *Proceedings of the 20th AAS/AIAA Spaceflight Mechanics Meeting,* San Diego, CA, Feb., 2010.

[3] BAI, X. and JUNKINS, J.L. "Modified Chebyshev-Picard Iteration Methods for Solution of Initial Value Problems," *Proceedings of the Kyle T. Alfriend Astrodynamics Symposium,* Monterey, CA, May, 2010.

[4] BAI, X. *Modified Chebyshev-Picard Iteration Methods for Solution of Initial Value and Boundary Value Problems.* Ph.D. Dissertation, Texas A&M University, College Station, TX, 2010.

[5] INCE, E.L. *Ordinary Differential Equations*, Dover Publications, Inc, New York, NY, 1956.

[6] CLENSHAW, C.W. and NORTON, H.J. "The Solution of Nonlinear Ordinary Differential Equations in Chebyshev Series," *The Computer Journal*, Vol. 6, No. 1, 1963, pp. 88–92.

[7] SHAVER, J.S. *Formulation and Evaluation of Parallel Algorithms for the Orbit Determination Problem.* Ph.D. Dissertation, Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA, Mar. 1980.

[8] SINHA, S.C. and BUTCHER, E. "Symbolic Computation of Fundamental Solution Matrices for Linear Time-Periodic Dynamic Systems," *Journal of Sound and Vibration*, Vol. 206, No. 1, 1997, pp. 61–85.

[9] FEAGIN, T. and NACOZY, P. "Matrix Formulation of the Picard Method for Parallel Computation," *Celestial Mechanics and Dynamical Astronomy*, Vol. 29, Feb. 1983, pp. 107–115.

[10] FUKUSHIMA, T. "Vector Integration of Dynamical Motions by the Picard-Chebyshev Method," *The Astronomical Journal*, Vol. 113, Jun. 1997, pp. 2325–2328.

[11] FEAGIN, T. *The Numerical Solution of Two Point Boundary Value Problems Using Chebyshev Series.* Ph.D. Dissertation, The Universtiy of Texas at Austin, Austin, TX, 1973.

[12] DORMAND, J.R., EL-MIKKAWY, M.E.A., and PRINCE, P.J. "Families of Runge-Kutta-Nystrom Formulae," *IMA Journal of Numerical Analysis*, Vol. 7, No. 2, 1987, pp. 235–250.

[13] DORMAND, J.R., EL-MIKKAWY, M.E.A., and PRINCE, P.J. "High-Order Embedded Runge-Kutta-Nystrom Formulae," *IMA Journal of Numerical Analysis*, Vol. 7, No. 4, 1987, pp. 423–430.

[14] FRANKLIN, M.A. "Parallel Solution of Ordinary Differential Equations," *IEEE Transactions on Computers*, Vol. 27, May 1978, pp. 413–420.

[15] MIRANKER, W.L. and LINIGER, W. "Parallel Methods for the Numerical Integration of Ordinary Differential Equations," *Mathematics of Computation*, Vol. 21, Jul. 1969, pp. 303–320.

[16] ISERLES, A. *A First Course in the Numerical Analysis of Differential Equations*, Cambridge University Press, NY, Second Ed., 2008.

[17] BURRAGE, K. "Parallel Methods for Initial Value Problems," *Applied Numerical Mathematics*, Vol. 11, Jan. 1993, pp. 5–25.

[18] HOUWEN, P.V.D. and SOMMEIJER, B.P. "Parallel ODE Solvers," *ACM SIGARCH Computer Architecture News- Special Issue: Proceedings of the 4th international conference on Supercomputing*, Vol. 18, Sep. 1990, pp. 71–81.

[19] BELLEN, A. and ZENNARO, M. "Parallel Algorithms for Initial-Value Problems for Difference and Differential Equations," *Journal of Computational and Applied Mathematics*, Vol. 25, No. 3, 1989, pp. 341–350.

[20] GEAR, C. and XU, X. "Parallelism Across Time in ODEs," *Applied Numerical Mathematics*, Vol. 11, No. 1–3, 1993, pp. 45–68.

[21] HOUWEN, P.J., SOMMEIJER, B.P., and CONG, N.H. "Stability of Collocation-Based Runge-Kutta-Nyström Methods," *BIT Numerical Mathematics*, Vol. 31, No. 3, 1991, pp. 469–481.

[22] SOMMEIJER, B. "Explicit, High-Order Runge-Kutta-Nyström Methods for Parallel Computers," *Applied Numerical Mathematics*, Vol. 13, No. 1–3, 1993, pp. 221–240.

[23] PARKER, G.E. and SOCHACKI, J.S. "Implementing the Picard Iteration," *Neural, Parallel and Scientific Computations*, Vol. 4, No. 1, 1996, pp. 97–112.

[24] FOX, L. and PARKER, I.B. *Chebyshev Polynomials in Numerical Analysis*, Oxford University Press, London, UK, 1972.

[25] CLENSHAW, C.W. "The Numerical Solution of Linear Differential Equations in Chebyshev Series," *Mathematical Proceedings of the Cambridge Philosophical Society*, Vol. 53, 1957, pp. 134–149.

[26] SCRATON, R.E. "The Solution of Linear Differential Equations in Chebyshev Series," *The Computer Journal*, Vol. 8, No. 1, 1965, pp. 57–61.

[27] WRIGHT, K. "Chebyshev Collocation Methods for Ordinary Differential Equations," *The Computer Journal*, Vol. 6, No. 4, 1964, pp. 358–365.

[28] NORTON, H.J. "The Iterative Solution of Nonlinear Ordinary Differential Equations in Chebyshev Series," *The Computer Journal*, Vol. 7, No. 2, 1964, pp. 76–85.

[29] URABE, M. "Galerkin's Procedure for Nonlinear Periodic Systems," *Archive for Rational Mechanics and Analysis*, Vol. 20, Jan. 1965, pp. 120–152.

[30] URABE, M. and REITER, A. "Numerical Computation of Nonlinear Forced Oscillations by Galerkin's Procedure," *Journal of Mathematical Analysis and Application*, Vol. 14, No. 1, 1966, pp. 107–140.

[31] VLASSENBROECK, J. and DOOREN, R.V. "A Chebyshev Technique for Solving Nonlinear Optimal Control Problems," *IEEE Transactions on Automatic Control*, Vol. 33, Apr. 1988, pp. 333–340.

[32] FOX, K. "Numerical Integration of the Equations of Motion of Celestial Mechanics," *Celestial Mechanics and Dynamical Astronomy*, Vol. 33, No. 2, 1984, pp. 127–142.

[33] FILIPPI, S. and GRÄF, J. "New Runge-Kutta-Nyström Formula-Pairs of Order 8(7), 9(8), 10(9) and 11(10) for Differential Equations of the Form $y'' = f(x, y)$," *Journal of Computational and Applied Mathematics*, Vol. 14, Mar. 1986, pp. 361–370.

[34] MONTENBRUCK, O. "Numerical Integration Methods for Orbital Motion," *Celestial Mechanics and Dynamical Astronomy*, Vol. 53, No. 1, 1992, pp. 59–69.

[35] HADJIFOTINOU, K.G. and GOUSIDOU-KOUTITA, M. "Comparison of Numerical Methods for the Integration of Natural Satellite Systems," *Celestial Mechanics and Dynamical Astronomy*, Vol. 70, No. 2, 1998, pp. 99–113.

[36] SHARP, P.W. "N-Body Simulations: The Performance of Some Integrators," *ACM Transactions on Mathematical Software*, Vol. 32, No. 3, 2006, pp. 375–395.

[37] FUKUSHIMA, T. "Picard Iteration Method, Chebyshev Polynomial Approximation, and Global Numerical Integration of Dynamical Motions," *The Astronomical Journal*, Vol. 113, May 1997, pp. 1909–1914.

[38] SCHAUB, H. and JUNKINS, J.L. *Analytical Mechanics of Space Systems*, American Institute of Aeronautics and Astronautics, Inc, Reston, VA, First Ed., 2003.

[39] COFFEY, S.L., HEALY, L., and NEAL, H. "Applications of Parallel Processing to Astrodynamics," *Celestial Mechanics and Dynamical Astronomy*, Vol. 66, Mar. 1996, pp. 61–70.

[40] COFFEY, S.L., NEAL, H.L., VISEL, C., and CONOLLY, P. "Demonstration of a Special-Perturbations-Based Catalog in the Naval Space Command System," *Proceedings of the 1998 AAS/AIAA Spaceflight Mechanics Meeting*, Breckenridge, CO, 1998.