Computer Physics Communications 182 (2011) 1187–1198

Contents lists available at ScienceDirect

Computer Physics Communications

www.elsevier.com/locate/cpc

Parallel implementation of an adaptive and parameter-free N-body integrator [☆]

C. David Pruett [a,*], William H. Ingham [b], Ralph D. Herman [b]

[a] Department of Mathematics & Statistics, James Madison University, Harrisonburg, VA 22807, United States
[b] Department of Physics & Astronomy, James Madison University, Harrisonburg, VA 22807, United States

ARTICLE INFO

Article history:
Received 6 July 2010
Received in revised form 3 January 2011
Accepted 4 January 2011
Available online 21 January 2011

Keywords:
N-body problem
Initial-value problem
Power series
Maclaurin series
Ordinary differential equations
Parallel computation

ABSTRACT

Previously, Pruett et al. (2003) [3] described an N-body integrator of arbitrarily high order M with an asymptotic operation count of $O(M^2N^2)$. The algorithm's structure lends itself readily to data parallelization, which we document and demonstrate here in the integration of point-mass systems subject to Newtonian gravitation. High order is shown to benefit parallel efficiency. The resulting N-body integrator is robust, parameter-free, highly accurate, and adaptive in both time-step and order. Moreover, it exhibits linear speedup on distributed parallel processors, provided that each processor is assigned at least a handful of bodies.

Program summary

Program title: PNB.f90
Catalogue identifier: AEIK_v1_0
Program summary URL: http://cpc.cs.qub.ac.uk/summaries/AEIK_v1_0.html
Program obtainable from: CPC Program Library, Queen's University, Belfast, N. Ireland
Licensing provisions: Standard CPC license, http://cpc.cs.qub.ac.uk/licence/licence.html
No. of lines in distributed program, including test data, etc.: 3052
No. of bytes in distributed program, including test data, etc.: 68 600
Distribution format: tar.gz
Programming language: Fortran 90 and OpenMPI
Computer: All shared or distributed memory parallel processors
Operating system: Unix/Linux
Has the code been vectorized or parallelized?: The code has been parallelized but has not been explicitly vectorized.
RAM: Dependent upon N
Classification: 4.3, 4.12, 6.5
Nature of problem: High accuracy numerical evaluation of trajectories of N point masses each subject to Newtonian gravitation.
Solution method: Parallel and adaptive extrapolation in time via power series of arbitrary degree.
Running time: 5.1 s for the demo program supplied with the package.

1. Introduction

Of historical and enduring practical interest, the problem of N-body gravity occupies a unique place at the heart of classical physics. N-body integrators seek to accurately capture the trajectories of individual bodies and hence solve Newton's equations of motion without approximation (except those inherent in the numerical scheme itself). Because the computational effort scales as N^2, studies of large-N systems demand efficient parallel implementation.

Parker and Sochacki [1] outlined a novel approach to solving systems of ODEs that was based ostensibly upon (modified) Picard iteration. One can view the Parker–Sochacki approach as a power-series solution technique that expands the state variables of the system as Maclaurin polynomials of arbitrarily high degree, provided that the generator of the system is itself of polynomial form. The key breakthrough in their approach was the recognition that virtually all non-stochastic ODEs can be re-cast with polynomial generators. The merits of this methodology are simplicity, extraordinary accuracy, and wide-ranging applicability. The Parker–

☆ This paper and its associated computer program are available via the Computer Physics Communications homepage on ScienceDirect (http://www.sciencedirect.com/science/journal/00104655).

* Corresponding author. Tel.: +1 540 568 6227.
E-mail addresses: pruettcd@jmu.edu (C.D. Pruett), inghamwh@jmu.edu (W.H. Ingham), hermanrd@jmu.edu (R.D. Herman).
URL: http://www.math.jmu.edu/~dpruett (C.D. Pruett).

0010-4655/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.cpc.2011.01.014

Sochacki method has recently been applied by Stewart and Blair [2] to the integration of spiking neural networks.

In 2003, Pruett et al. [3] adapted the Parker–Sochacki technique to the N-body problem. At the time, they were unaware that Fehlberg [4] had used an analogous approach for the three-body problem as early as 1964 and that Broucke [5] had extended the method to the N-body problem in 1971. Nevertheless, Pruett et al. extended earlier work by rendering the algorithm doubly adaptive (that is, adaptive in both time increment and polynomial order). Moreover, they optimized the scheme for minimum computational effort to render the algorithm parameter-free.

Numerical experiments with the doubly-adaptive variant of the N-body algorithm addressed what was then an open question: What is the optimal order of a numerical integrator for the N-body problem [6,7]? These experiments revealed that optimal order is higher than previously suggested and is inherently dynamic, dependent upon the number of bodies N, the specified global error tolerance ϵ, and the instantaneous configuration of the bodies, higher order being advantageous during close encounters.

The N-body algorithm previously reported [3] is robust and accurate. However, with a per-step operation count of $O(N^2M^2)$, where M is the order, it is not particularly efficient despite the relatively large time step characteristic of high-order methods. This shortcoming is offset by a simple structure that lends itself readily to data parallelism. Here we document and demonstrate a parallel implementation of the algorithm. To improve notation and to correct some typographic errors in the previous paper, we summarize both the original serial algorithm and its parallelization.

The resulting N-body integrator is robust, doubly adaptive (i.e., in time and order), and parameter-free. Moreover, it exhibits linear speedup on distributed parallel processors, provided that each processor has at least a handful of bodies.

The method is demonstrated in the integration of N-body systems of point masses subject to Newtonian gravitation. The potential range of applications of the methodology, however, is broad. For example, particle methods for fluid-flow simulation are structurally similar to N-body algorithms.

2. Governing equations

In [3], the N-body problem is formulated as an autonomous system of ordinary differential equations for time $t > t_0$, namely

$$\frac{dX_{ij}}{dt} = V_{ij} \tag{1}$$

$$\frac{dV_{ij}}{dt} = \sum_{k\neq j} M_k (X_{ik} - X_{ij}) S_{jk}^3 \tag{2}$$

$$\frac{dS_{jk}}{dt} = -S_{jk}^3 \sum_{i=1}^{3} (X_{ik} - X_{ij})(V_{ik} - V_{ij}) \quad (j \neq k) \tag{3}$$

where X_{ij} and V_{ij} are the ith components $(i = 1, 2, 3)$ of position and velocity, respectively, of body j $(j = 1, 2, \ldots, N)$, $M_k \equiv G\mu_k$, μ_k is the mass of body k $(k = 1, 2, \ldots, N)$, and G is the universal gravitational constant. The system is cast with its generator in *polynomial form*. This unusual convention, which is central to the Parker–Sochacki method, is facilitated by defining the inverse separation distance between the centers of bodies j and k as follows: $S_{jk} \equiv 1/r_{jk}$ $(j \neq k)$, where $r_{jk} \equiv \sqrt{\sum_{i=1}^{3}(X_{ij} - X_{ik})^2}$. Note that the generator of Eq. (3) is polynomial of degree five. The governing equations are completed by specifying initial conditions $X_{ij}(t_0)$ and $V_{ij}(t_0)$.

3. Baseline algorithm

The polynomial form of system Eqs. (1)–(3) admits power-series solution techniques. Furthermore, because the system is autonomous, the time origin can be shifted to zero $(t_0 = 0)$ following each integration step without loss of generality. As a result, solution components can be approximated by Maclaurin polynomials in time t. For example, let $x_{ij}(t)$ be the numerical approximant of the exact solution $X_{ij}(t)$, and let x_{ijm} denote the mth-order coefficient of the Mth-order Maclaurin polynomial approximation for X_{ij}. That is, $X_{ij}(t) \approx x_{ij}(t) \equiv \sum_{m=0}^{M} x_{ijm} t^m$. Let similar conventions hold for the approximants v_{ij}, s_{jk}, and powers of s_{jk}.

Now suppose the Maclaurin-polynomial coefficients are known for terms through order $m - 1$. By integrating the Maclaurin polynomial approximation of Eq. (1) with respect to t, for example, the mth-order coefficient of position is obtained as

$$x_{ijm} = \frac{1}{m} v_{i,j,m-1} \tag{4}$$

Similarly, and by using Cauchy products to evaluate quadratic nonlinearities and repeated Cauchy products for higher-order nonlinearities, the following recursion relationships hold among successive Maclaurin coefficients:

$$v_{ijm} = \frac{1}{m} \sum_{k\neq j} M_k \sum_{q=0}^{m-1} (x_{ikq} - x_{ijq}) \left(s^3\right)_{j,k,m-1-q} \tag{5}$$

$$s_{jkm} = -\frac{1}{m} \sum_{q=0}^{m-1} \left(s^3\right)_{jkq} a_{j,k,m-1-q} \quad (k \neq j) \tag{6}$$

$$\left(s^2\right)_{jkm} = \sum_{q=0}^{m} s_{jkq} s_{j,k,m-q} \quad (k \neq j) \tag{7}$$

$$\left(s^3\right)_{jkm} = \sum_{q=0}^{m} \left(s^2\right)_{jkq} s_{j,k,m-q} \quad (k \neq j) \tag{8}$$

$$a_{jkm} \equiv \sum_{q=0}^{m} \sum_{i=1}^{3} (x_{ijq} - x_{ikq})(v_{i,j,m-q} - v_{i,k,m-q}) \quad (k \neq j) \tag{9}$$

The quantity a_{jkm} is defined as a computational convenience. For Eqs. (4)–(9), the coordinate index ranges $i = 1, 2, 3$, and the body indices j and k both range from 1 to N; e.g., $j = 1, 2, \ldots, N$.

The essence of the baseline algorithm consists of three procedures inside a time-advancement loop:

(1) The zero-order quantities x_{ij0} and v_{ij0} are simply the initial positions and velocities. The zero-order inverse separations s_{jk0} follow from the initial positions x_{ij0}. That is, $s_{jk0} \equiv 1/r_{jk}(0)$. Then the quantities $(s^2)_{jk0}$, $(s^3)_{jk0}$, and a_{jk0} follow from Eqs. (7)–(9) for $m = 0$.

(2) The quantities x_{ijm}, v_{ijm}, s_{jkm} $(s^2)_{jkm}$, $(s^3)_{jkm}$, and a_{jkm} are defined recursively by looping through Eqs. (4)–(9) for $m = 1, 2, \ldots, M$. This process yields the Maclaurin coefficients for position and velocity, namely $\mathbf{x}_{jm} = [x_{1jm}, x_{2jm}, x_{3jm}]$ and $\mathbf{v}_{jm} = [v_{1jm}, v_{2jm}, v_{3jm}]$, $m = 0, 1, \ldots, M$.

(3) The system state is advanced in time by Δt merely by evaluating the Maclaurin polynomials of position and velocity. For example, the position and velocity updates are:

$$\mathbf{x}_j(\Delta t) = \mathbf{x}_{j0} + \mathbf{x}_{j1}\Delta t + \mathbf{x}_{j2}\Delta t^2 + \cdots + \mathbf{x}_{jM}\Delta t^M \tag{10}$$

$$\mathbf{v}_j(\Delta t) = \mathbf{v}_{j0} + \mathbf{v}_{j1}\Delta t + \mathbf{v}_{j2}\Delta t^2 + \cdots + \mathbf{v}_{jM}\Delta t^M \tag{11}$$

where $j = 1, 2, \ldots, N$. Polynomial evaluations are accomplished efficiently by Horner's method in $O(MN)$ operations.

Because the method is a one-step method and the generator is autonomous, initial conditions can be reset after each step, in which case Eqs. (10) and (11) are valid for all time steps.

The baseline algorithm is summarized below:

```
input $N$
input initial positions $x_{ij}(0)$ $(i=1,2,3)$, $(j=1,2,\dots,N)$
input initial velocities $v_{ij}(0)$ $(i=1,2,3)$, $(j=1,2,\dots,N)$
set parameters: integration time $T$, order $M$, time increment $\Delta t$
set initial conditions: $x_{ij0}=x_{ij}(0)$, $v_{ij0}=v_{ij}(0)$
$t \leftarrow 0$
Do until $(t > T)$
|   Establish inverse separations $s_{jk0}$ from $x_{ij0}$
|   Establish $(s^2)_{jk0}$, $(s^3)_{jk0}$, and $a_{jk0}$ from Eqs. (7)–(9)
|   Recursively, for $m=1,2,\dots,M$
|   |   Compute coefficients $x_{ijm}$, $v_{ijm}$, $s_{jkm}$, etc., from Eqs. (4)–(9)
|   Update $\mathbf{x}_j(\Delta t)$ and $\mathbf{v}_j(\Delta t)$ by Eqs. (10)–(11)
|   $t \leftarrow t+\Delta t$
|   Output: running time $t$, positions $\mathbf{x}_j(\Delta t)$, velocities $\mathbf{v}_j(\Delta t)$
|   Reset initial conditions: $x_{ij0} \leftarrow x_{ij}(\Delta t)$, $v_{ij0} \leftarrow v_{ij}(\Delta t)$
```

Note that order M of the baseline algorithm is arbitrary.

4. Adaptive stepping

Let Δt_l be the variable interval of step l for a time integration of total duration T, and let ϵ be the global relative error tolerance. Then the *allowable* per-step relative error ϵ_l is given by

$$\epsilon_l = \frac{\epsilon \Delta t_l}{T} \tag{12}$$

For an integration scheme of order m, under reasonable assumptions it can be shown that the global relative error tolerance is maintained provided the *estimated* per-step relative error e_l is bounded by ϵ_l, in which case

$$e_l \approx \frac{v_{\max}(l)}{v_s}\Delta t_l^{m+1} \leqslant \epsilon_l \tag{13}$$

The estimation e_l is based upon the first neglected term of the Maclaurin polynomial approximation of velocity. Here $v_s \equiv \max_j \|\mathbf{v}_{j0}(0)\|_2$ is a velocity scale based upon the Euclidean velocity of the initial condition, and $v_{\max}(l) \equiv \max_j \|\mathbf{v}_{j,m+1}(t_l)\|_2$ is a norm for the $(m+1)$-order Maclaurin coefficient at the lth step. Combining Eqs. (12) and (13) leads to the following bound on the time step:

$$\Delta t_l \leqslant \left[\frac{\epsilon v_s}{T v_{\max}(l)}\right]^{1/m} \tag{14}$$

5. Optimization

Suppose W_+, W_-, $W_\times$, and $W_\div$ denote the number of additions, subtractions, multiplications, and divisions, respectively, per time step in the algorithm. These floating-point operations can be counted precisely and the dominant terms identified. For example,

$$\begin{aligned} W_+(M,N) &= \frac{5}{4}N(N-1)(M+1)(M+2) + \frac{3}{2}N^2 M(M+1) \\ &\quad + \frac{1}{2}N(N-1)M + 6N(M-1) + 4N^2 \\ &\approx \frac{11}{4}M^2N^2 \end{aligned} \tag{15}$$

Similarly,

$$W_-(M,N) \approx 6M^2N^2 \tag{16}$$

$$W_\times(M,N) \approx \frac{17}{4}M^2N^2 \tag{17}$$

$$W_\div(M,N) \approx MN \tag{18}$$

The weighted per-step operation count is

$$\begin{aligned} W(M,N) &= w_+W_+(M,N) + w_-W_-(M,N) + w_\times W_\times(M,N) \\ &\quad + w_\div W_\div(M,N) \end{aligned} \tag{19}$$

where weights w_+, w_-, etc., reflect the respective relative costs on the given platform of the four basic arithmetic operations. For example, if the basic operations share the same weight (unity), then the asymptotic workload is $W(M,N) \approx 13M^2N^2$. Given the per-step workload Eq. (19), the computational cost per unit time for an integration step of size Δt can be defined as follows:

$$P(M,N,\Delta t) = \frac{W(M,N)}{\Delta t} \tag{20}$$

where $W(M,N)$ is the exact weighted operation count from Eq. (19). Numerical experimentation with the time-adaptive, fixed-order, N-body integrator (Fig. 4 of [3]) has shown $P(M,N,\Delta t)$ to be a rapidly decreasing function of M for small M. $P(M,N,\Delta t)$ attains a minimum at a relatively large value of M (roughly 12 and 20 for single- and double-precision calculations, respectively) and then increases gradually. This behavior permits a simple-minded but effective search for the global minimum of P. With the modifications to the baseline algorithm outlined below, the enhanced algorithm adapts both Δt_l and M at each time step l (including the first) to minimize computational effort:

1. Immediately prior to the loop for $m=1,2,\dots,M$ in the baseline algorithm insert:

```
Set reference cost $P_0$ to an outrageously large value
```

2. Immediately after coefficient computation within loop for $m = 1,2,\dots,M$ insert:

```
Evaluate $\Delta t_l$ for order $(m-1)$ via Eq. (14)
Evaluate cost $P_1 \leftarrow P(m,N,\Delta t_l)$ via Eq. (20)
If $(P_1 > P_0)$ then
    $M \leftarrow m$
    Exit
End if
Reset reference cost: $P_0 \leftarrow P_1$
```

The additional computational cost incurred by the optimization is negligible relative to other time-advancement costs.

Finally, by defining $\epsilon \approx u$, where u is the machine unit round-off error of the computational platform, the algorithm is rendered parameter-free.

A demanding test of the doubly adaptive algorithm is the time evolution of a “swarm” of gravitating particles. To this end, a 96-particle system was considered. Initial positions are shown in Fig. 1. Each particle was assigned randomized Cartesian position and velocity components, each initialized to a random number uniformly distributed on $[-2,2]$. For clarity in the figure, vertical (z) position coordinates are shifted by two units so that all particles are plotted with positive stems.

Fig. 2 demonstrates the doubly adaptive behavior of the optimized algorithm during the integration of the particle swarm defined immediately above, for an integration time $0 \leqslant t \leqslant 0.5$. Note that optimal order varies from 15 to 23 while optimal time step varies by more than three orders of magnitude. The total number of time steps is 2211. Small steps correspond to near-collisions. For this calculation, $\epsilon = 10^{-13}$.

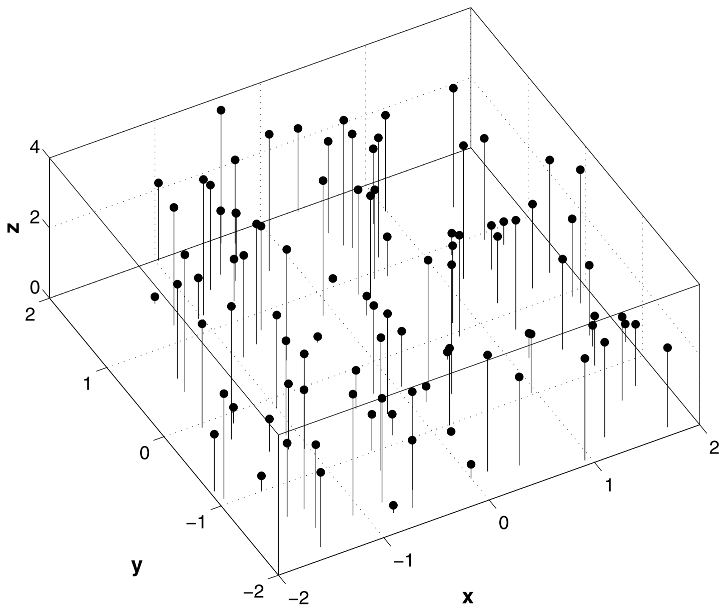

Fig. 1. Initial positions of 96-particle "swarm", with vertical coordinate shifted by two for clarity.

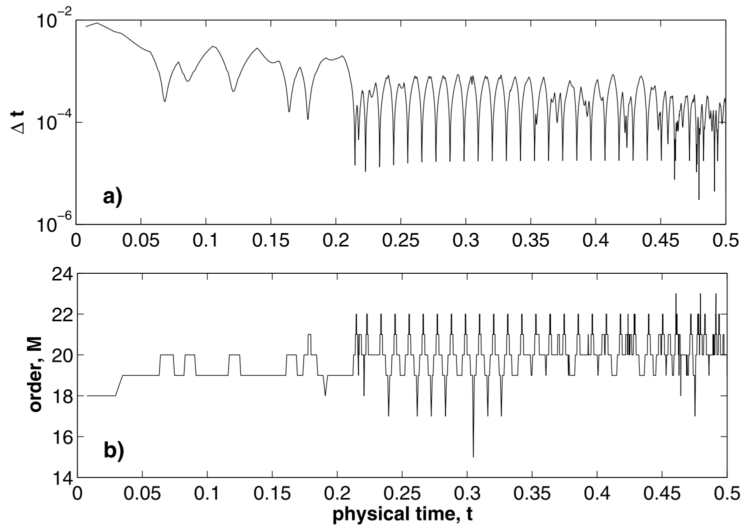

Fig. 2. Behavior of doubly adaptive algorithm for 96-particle system: (a) adaptation of time increment Δt, and (b) simultaneous adaptation of Maclaurin order M.

6. Parallel implementation

A parallel version of the algorithm has been implemented in Fortran 90 with explicit message passing via the Message Passing Interface (MPI) protocol. The parallel algorithm is summarized in pseudocode below, where q is the processor id, and message-passing commands are indicated in **boldface**. First, we summarize the algorithm's basic functions.

Let p be the number of processors; that is, $q = 1, 2, \ldots, p$. The present N-body algorithm lends itself to data-parallel structure, as shown in Fig. 3, illustrated for $p = 4$. In principle, each processor is responsible for updating the positions and velocities of a subset of the total number of bodies, and the data are distributed accordingly. Specifically, for fixed m, quantities x_{ijm} and v_{ijm} are $N \times 3$ matrices. In contrast s_{jkm}, $(s^2)_{jkm}$, $(s^3)_{jkm}$, and a_{jkm} are $N \times N$ matrices. All matrices are "strip-mined" by distributing strips to each of the p processors as shown in the figure.

There are two modes of operation, each designed to ensure computational efficiency via optimal load balancing. In the first mode, any number of processors $p \leqslant N$ is allowed provided

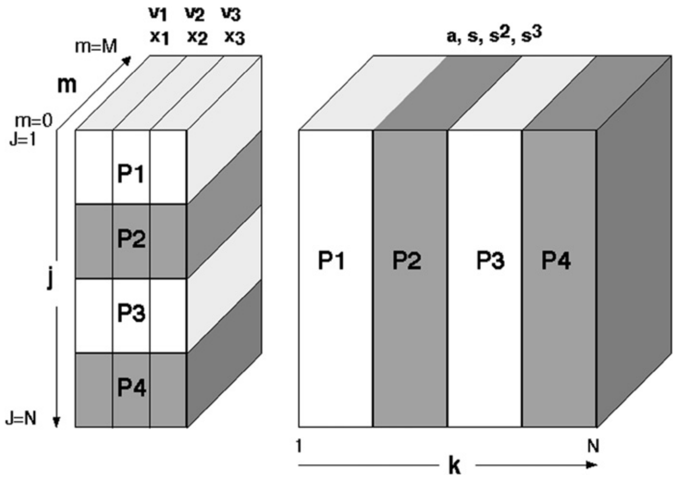

Fig. 3. Data-parallel structure of parallelized N-body algorithm, shown for $p = 4$.

$N \bmod p = 0$, in which case each processor is responsible for exactly N/p bodies. For example, if $N = 32$ and $p = 4$, processors 1, 2, 3, and 4, are responsible for bodies 1 to 8, 9 to 16, 17 to 24, and 25 to 32, respectively. For the first mode of operation, the computational workload is essentially perfectly balanced.

The algorithm defaults to its second mode of operation if p is not an integer divisor of N. In this case, processors 1 to $p-1$ are each assigned $b = \lceil \frac{N}{p} \rceil$ bodies, and processor p is assigned the remainder of bodies $N - b(p-1)$. (Here, $\lceil z \rceil$ is the ceiling function that returns the closest integer greater than or equal to z.) For example, if $N = 32$ and $p = 5$, then $b = 7$, in which case processors 1–4 are each responsible for 7 bodies, and processor 5 is responsible for 4 bodies.

For the second mode of operation, if the number of processors $p \leqslant N$ is not further restricted, then, in some instances, the remainder $N - b(p-1)$ can be negative. Thus, the constraint $b(p-1) < N$ is enforced. Because $b = \lceil \frac{N}{p} \rceil = \lfloor \frac{N+(p-1)}{p} \rfloor$, where $\lfloor z \rfloor$ is the floor function, the constraint equation can be expressed $\lfloor \frac{N+(p-1)}{p} \rfloor (p-1) < N$. Finally, we note that $\lfloor \frac{N+(p-1)}{p} \rfloor (p-1) \leqslant \frac{N+(p-1)}{p}(p-1)$, which yields the working constraint inequality $\frac{N+(p-1)}{p}(p-1) < N$, whose solution is $p < \sqrt{N} + 1$. By numerical experimentation, we have found it permissible to relax the constraint ever so slightly by taking $p \leqslant \lceil \sqrt{N} \rceil + 1$. Therefore, if $N = 32$, for example, the maximum number of processors allowed is $p = 7$.

For either mode of operation, let

$$b = \left\lceil \frac{N}{p} \right\rceil \tag{21}$$

and, for each processor $q = 1, 2, \ldots, p$, define

$$\begin{aligned} j_0(q) &\equiv (q-1)b + 1 = k_0(q) \\ j_1(q) &\equiv \min(qb, N) = k_1(q) \end{aligned} \tag{22}$$

Then processor q is accountable for computing the position coefficients $x_{i,j=j_0(q):j_1(q),m}$ and the velocity coefficients $v_{i,j=j_0(q):j_1(q),m}$, $i = 1, 2, 3$ and $m = 1, \ldots, M$. Needed for these calculations on processor q are $s_{j,k=k_0(q):k_1(q),m}$ for all bodies j and all previous m, and similarly for s^2, s^3, and a.

During an integration step, each processor independently advances from, say, order $m-1$ to m all Maclaurin polynomials belonging to its subset of bodies. Before the computation can proceed to the next higher order, all processors must share data because of the dependency of the Cauchy products for order $m+1$ on the Maclaurin coefficients of position and velocity for all bodies and all previous orders. At this stage, the processors distribute their individual contributions via MPI's ALLGATHER operation. Specifically, each processor gathers $6N$ data values comprised of the Maclaurin-series components of degree m for the position coordinates (x_{ijm}, $i = 1, 2, 3$) and analogous values for the three velocity components (v_{ijm}, $i = 1, 2, 3$). Thus, to build the Maclaurin series to its full order M, $6MN$ data values must be gathered by all processors per time step. The additional ALLGATHER that lies outside the inner (m) loop in the algorithm below does not contribute to the asymptotic communications count.

If p divides evenly into N, the computational workload is almost perfectly balanced, with the exception of one node, whose job it is to perform the additional evaluations such as $P(m, N, \Delta t_l)$ necessary for optimization. The additional effort, however, which primarily involves integer operations and few at that, is minuscule relative to the major computational burden, so there is little or nothing to be gained by attempting to parallelize the optimization process. For mode-two operation, the workload of the pth processor is less than that of the other processors. Thus, in principle there would be a slight computational advantage to assigning the pth processor the task of optimization. However, in our current implementation, the head node (processor 1) assumes all extra tasks, including I/O and optimization.

Here is the complete parameter-free, parallel algorithm:

```
(q=all) query system for p and process id
(q=1) input and (q=all) **broadcast** N
(q=1) input and (q=all) **broadcast** maximum order M
(q=all) compute b from Eq. (21)
(q=all) evaluate $j_0(q) = k_0(q)$ and $j_1(q) = k_1(q)$ via Eq. (22)
(q=all) allocate memory based on N, M, p, and b
(q=1) input integration time T
(q=1) initialize running time $t \leftarrow 0$
(q=1) initialize Terminate1 $\leftarrow$ FALSE and (q=all) **broadcast**
(q=1) input and (q=all) **broadcast** masses $M_j$
(q=1) input and (q=all) **broadcast** initial data $x_{ij0} = x_{ij}(0)$,
    $v_{ij0} = v_{ij}(0)$

(q=all) time advancement: do until (Terminate1)
  | (q=1 only) initialize $P_0 \leftarrow 10^{30}$ (set outrageously large)
  | compute inverse separations $s_{j,k=k_0:k_1,0}$ from $x_{ij0}$, $j = 1, \ldots, N$
  | compute $(s^2)_{j,k=k_0:k_1,0}$, $(s^3)_{j,k=k_0:k_1,0}$, $a_{j,k=k_0:k_1,0}$
  |   via Eqs. (7)–(9)
  | recursively, for $m = 1, 2, \ldots, M$
  | | compute coefficients $x_{i,j=j_0:j_1,m}$ from Eq. (4)
  | | **allgather** $x_{ijm}$, current m
  | | compute coefficients $v_{i,j=j_0:j_1,m}$ from Eq. (5)
  | | **allgather** $v_{ijm}$, current m
  | | compute coefficients $s_{j,k=k_0:k_1,m}$ from Eq. (6)
  | | compute coefficients $(s)^2_{j,k=k_0:k_1,m}$ from Eq. (7)
  | | compute coefficients $(s)^3_{j,k=k_0:k_1,m}$ from Eq. (8)
  | | compute coefficients $a_{j,k=k_0:k_1,m}$ from Eq. (9)
  | | set Terminate2 $\leftarrow$ FALSE
  | | (q=1 only) compute $\Delta t$ via Eq. (14)
  | | (q=1 only) compute relative cost $P_1 \leftarrow P(N, m, \Delta t)$
  | |   via Eq. (20)
  | | (q=1 only) if ($P_1 > P_0$) set Terminate2 $\leftarrow$ TRUE
  | | **broadcast** Terminate2 from q=1
  | | if (Terminate2) exit m loop
  | | $P_0 \leftarrow P_1$
  | (q=1 only) $t \leftarrow t + \Delta t$
  | (q=1 only) if ($t > T$) Terminate1 $\leftarrow$ TRUE
  | **broadcast** $\Delta t$ and Terminate1 from q=1
  | update $\mathbf{x}_{j=j_0:j_1}(\Delta t)$ and $\mathbf{v}_{j=j_0:j_1}(\Delta t)$ by Eqs. (10)–(11)
  | **allgather** $\mathbf{x}_j(\Delta t) \rightarrow x_{ij0}$ and $\mathbf{v}_j(\Delta t) \rightarrow v_{ij0}$ (reset ICs)
  | (q=1 only) output: running time t, positions $x_{ij0}$, velocities $v_{ij0}$
  | if (Terminate1) exit time advancement loop

(q=all) stop
```

Recall that the asymptotic operational workload per step $W(M, N) \approx 13M^2N^2$ if adds and subtracts, multiplies and divides are assumed to share the same weight. It follows that the asymptotic per-processor operation and communication costs per integration step are:

$$\begin{aligned} &\text{per-processor operation count, } W_p \approx \frac{13M^2N^2}{p} \\ &\text{per-processor communication cost, } C_p \approx 6MN \\ &\text{computation/communication ratio, } R = \frac{W_p}{C_p} \approx \frac{13MN}{6p} \end{aligned} \tag{23}$$

Because of latency and bandwidth issues, data transfers between CPUs of distributed-memory systems are typically much slower than floating-point operations, frequently by one or two orders of magnitude. Consequently, parallel performance will be suboptimal unless the ratio R of computational operations to communication transfers is large (i.e., $R \gg 1$). For the sake of illustration, suppose there are at least five bodies per processor ($p \leqslant N/5$). Then, for the current algorithm $R \geqslant \frac{65M}{6} \approx 11M$, based on Eq. (23) above. Thus, a high-order ($M \gg 1$) algorithm may benefit from a naturally high computation to communication ratio and should perform well in a distributed-memory environment. For the current implementation, as shown in the previous section, optimal order M for a 64-bit calculation is typically about 20. Hence, for the current algorithm, with a least a handful of bodies per processor, $R > 200$, which should yield good parallel performance even in environments with relatively slow communication channels. The results of the next section confirm this expectation.

It should be acknowledged that the baseline serial algorithm enjoys a factor of two in computational efficiency over the parallel N-body algorithm. This stems from the fact that the former exploits symmetry with respect to the first two indices for the quantities s_{jkm}, $(s^2)_{jkm}$, $(s^3)_{jkm}$, and a_{jkm}, which the latter cannot do without violating nodal memory independence on distributed-memory platforms.

Finally, we note that, whenever $N \geqslant 36$, mode-two operation guarantees at least a handful of bodies per processor (for all but the last processor), and thus ensures efficient parallel implementation.

7. Results

In Pruett et al. [3], a serial version of the present doubly adaptive N-body algorithm was presented and tested in comparisons with Bulirsch–Stoer methodology. Although the serial algorithm was not as efficient as the Bulirsch–Stoer method, it had the advantage of being parameter-free. Moreover, because of the simplicity of the underlying algorithm, the methodology is readily amenable to a data-parallel implementation, which was detailed in previous sections. Here we validate and demonstrate a Fortran90/MPI implementation of the parallelized N-body algorithm.

Although the baseline serial algorithm is elegant in its simplicity, parallelization adds considerable complexity. Hence, it was essential that, prior to any performance testing, the parallel algorithm be thoroughly validated for correctness. This was accomplished by comparing its results with verified results obtained for identical initial data from the serial algorithm. Several verification test cases were considered. In each case, serial and parallel results were essentially identical. Because parallel algorithms are non-deterministic in general, exact agreement between serial and parallel computations is not guaranteed, nor is perfect replicability of parallel results. Occasional small differences in the last one or two significant digits are a consequence of non-deterministic round-off error propagation stemming from the non-associativity of numerical addition.

To demonstrate the attributes of the current algorithm, we consider three test cases, the first two of which are benign and the third of which challenges any integration scheme, including Bulirsch–Stoer.

7.1. A binary-star simulation

We first consider a simple two-body system for which an exact solution is known. In particular, we consider a "binary-star" scenario with a periodic solution. Initial conditions are prescribed so that two stars revolve in circular orbits about the system's stationary center of mass (CM). The stars remain collinear with and on opposite sides of the CM. Initial data are provided in Table 1. With $G = 1$, the orbital radii are $r_1 = 2$ and $r_2 = 1$, respectively, and both orbits have period 6π. The exact position

Table 1
Initial data for binary-star integration.

j	μ_j	$x_{1j}(0)$	$x_{2j}(0)$	$x_{3j}(0)$	$v_{1j}(0)$	$v_{2j}(0)$	$v_{3j}(0)$
1	1	−2	0	0	0	−2/3	0
2	2	1	0	0	0	1/3	0

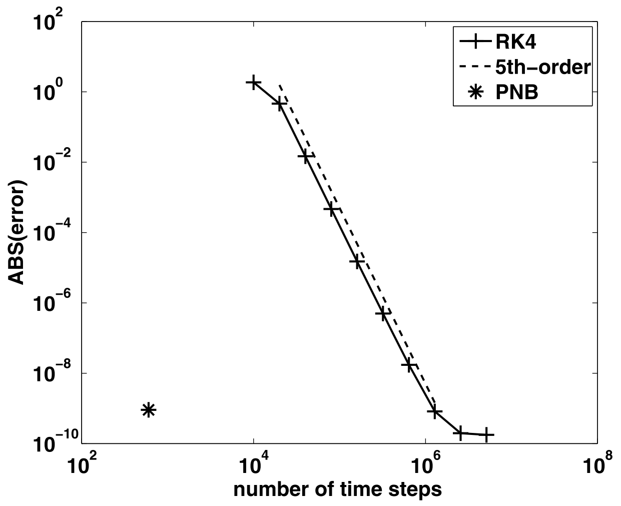

Fig. 4. Comparison of global accuracy for reference (RK4) and current (PNB) integration schemes for binary-star system with known analytic solution. For commensurate absolute error of 10^{-9}, RK4 and PNB schemes require one million and 600 time steps, respectively.

vectors for bodies 1 and 2 are $\mathbf{x}_1(t) = [-2\cos(\omega t), -2\sin(\omega t), 0]$ and $\mathbf{x}_2(t) = [\cos(\omega t), \sin(\omega t), 0]$, respectively, with $\omega = 1/3$. The duration of the integration is $0 \leqslant t \leqslant 5000$, which corresponds to 265.25 orbital periods.

The computing platform was a small shared-memory symmetric multi-processor belonging to the Department of Mathematics and Statistics at James Madison University. Specifically, the platform consists of a single quad-core Intel E5320 Xeon processor running at 1.86 GHz with 16 GB of RAM. The operating system was 64-bit Ubuntu Linux, and the compiler was **gfortran** with default options.

Fig. 4 presents global error at termination of the calculation for two integration schemes: the current algorithm and a classic workhorse, the fourth-order Runge–Kutta (RK4) method. Whereas the present algorithm exploits a polynomial form of the system generator, the RK4 implementation involves a standard form of the generator (that is, one involving roots and reciprocals). For this benign problem, RK4 performs well with a fixed time step. However, more than one million time steps are needed to reduce global absolute error to 10^{-9}. Significant reduction of error beyond this level is prohibited by the accumulation of round-off errors.

Somewhat surprisingly, RK4 exhibits fifth-order convergence to the exact solution for this problem, as shown in Fig. 4, presumably picking up one order of accuracy because of the periodicity of the solution. (Similarly, low-order integration schemes such as composite rectangle rule yield spectral accuracy for periodic problems.) In contrast, the current parameter-free scheme achieves global accuracy roughly commensurate with the best RK4 result in 600 time steps.

For the present algorithm, the maximum order M of the computation is hard-coded at 28. For this periodic problem, the optimization process pegs the scheme at its maximum order, where it remains throughout. Neither scheme benefits from adaptive stepping for the binary-star case, which is dynamically stationary. This is in distinct contrast to the situation observed for the other two

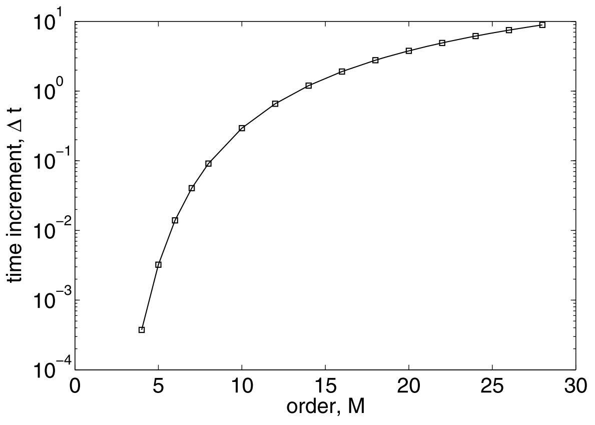

Fig. 5. Time increment Δt vs. Maclaurin polynomial order M for binary-star problem with current algorithm. Global error tolerance is $10u$ for all data points, with double precision machine unit roundoff error $u = 2.2 \times 10^{-15}$.

test cases, for which the optimizer dynamically and dramatically adjusts both M and Δt as shown in Fig. 2.

Whereas the computational time for the RK4 method is about 1.7 seconds, that of the current algorithm is 0.22 seconds. However, the RK4 scheme has been granted several advantages. First, the fixed time intervals are of the form $\Delta t = 2^{-n}$, where $1 \leqslant n \leqslant 10$ is an integer. Such representations can be expressed exactly in binary. Thus, no round-off error accumulates in the time t itself for the RK4 method. As implied above, for the present adaptive scheme, the time step is essentially constant for the binary-star problem; however, it varies ever so slightly. Thus, over long-term evolution modest error accrues in the time, which slightly contaminates the value of the "exact" solution. Without this advantage, the minimum global error attainable by the RK4 scheme was found to be approximately 10^{-8}.

Second, the binary-star problem is spatially two-dimensional, which the RK4 method exploits. The present N-body algorithm, on the other hand, is spatially three-dimensional, and no attempt is made to take advantage of the reduced dimension of the test case. Thus the current method incurs a factor of 3/2 penalty in workload relative to the reference methodology.

Third, the present scheme is run in parallel mode, with a single body assigned to each processor. This represents an extraordinarily inefficient use of the parallel algorithm, which invokes multiple MPI commands and incurs processor-to-processor communication costs unnecessary for systems of small N.

For small N, a fairer comparison pits RK4 methodology against the baseline serial algorithm from which the current parallel algorithm was derived. To that end, the serial algorithm without adaptation or optimization is implemented with a constant time step and $M = 28$. With 800 steps during the integration interval, the baseline algorithm achieves global error of 10^{-10} in 0.126 seconds of execution time. After accounting for the workload discrepancy associated with whether two or three spatial dimensions are involved, the baseline algorithm is more than twenty times faster than RK4, and the parallel algorithm is more than ten times faster, for commensurate accuracies.

The vast difference in the time increment of the two methods underscores the principal advantage of high-order methods, namely that, for commensurate accuracy, they permit larger time steps than do low-order ones. The advantages of very high order are illustrated in Fig. 5, which shows the variation in Δt with Maclaurin order M for the present algorithm implemented on the binary-star problem. As M is varied from 4 to 28, the time step allowed for a pre-specified global accuracy increases by some 4.5 orders of magnitude. Slightly more than half of that gain (in powers of ten) occurs between fourth and eighth order. As stated previously, for this problem, RK4 methodology behaves as a fifth-order method. As such, it might be expected to require a time increment approximately 3.5 orders (~3000 times) smaller than that of the current method with $M = 28$, for commensurate accuracies. Indeed, Fig. 4 confirms this expectation.

Table 2
Extrema of relative errors in total energy ($|\Delta E/E|$) and angular momentum ($|\Delta L/L|$), and computation time T, as functions of integration time step Δt, for hybrid symplectic scheme of Chambers.

Δt [days]	$\lvert\Delta E/E\rvert$	$\lvert\Delta L/L\rvert$	T [s]
20	3.13×10^{-7}	2.18×10^{-13}	11.0
10	3.82×10^{-8}	1.35×10^{-13}	16.4
5	4.70×10^{-9}	2.24×10^{-13}	27.5
2	9.61×10^{-10}	1.97×10^{-11}	58.4
1	1.05×10^{-10}	3.74×10^{-11}	111.8
0.5	1.07×10^{-10}	2.19×10^{-10}	222.7

Table 3
Extrema of relative errors in total energy ($|\Delta E/E|$) and angular momentum ($|\Delta L/L|$), and computation time T, as functions of relative velocity error tolerance ϵ for present serial algorithm.

ϵ	$\lvert\Delta E/E\rvert$	$\lvert\Delta L/L\rvert$	T [s]
10^{-3}	2.63×10^{-7}	3.30×10^{-8}	766.0
10^{-6}	1.56×10^{-10}	1.56×10^{-11}	1077.8
10^{-9}	1.78×10^{-12}	3.06×10^{-12}	1443.9

7.2. *Evolution of the solar system*

Of particular concern for N-body integrators is energy conservation over long-term integration. In the absence of non-elastic collisions, total energy should be exactly conserved. Modern integrators are designed to be *symplectic*. For Hamiltonian systems, symplectic integrators preserve structure in phase space, and as a consequence, they conserve energy to high order. [8,9]. Thus, symplectic integrators represent the current state of the art in simulating orbital dynamics.

Here we consider both short-term (50,000 years) and long-term (three million years) evolutions of the solar system. The first serves simply to benchmark the efficiency of the current algorithm relative to a symplectic integrator. The second firmly establishes the fidelity of the present algorithm.

7.2.1. *Short-term evolution of the solar system*

Representative of the class of symplectic integrators is the hybrid symplectic code MERCURY6 of Chambers [10], which couples a second-order, fixed-step symplectic integrator with adaptive Bulirsch–Stoer methodology. Typically, low-order integrators are unreliable for close encounters. Hence, MERCURY6 switches to Bulirsch–Stoer to negotiate near collisions. The hybrid scheme is both extremely fast and quite robust. Here we compare results obtained from MERCURY6 and the present method for a solar-system evolution of 50,000 years.

For consistency with the precursor to the present paper [3], initial data are taken from *Ephemerides of Minor Planets* for Epoch 1997, December 18.0 (Julian Date). Cartesian position coordinates (x_1, x_2, x_3) are given in astronomical units [AU] referenced from the sun, which is located initially at the origin. The computational platform case is the same as that of the two-body "binary-star" simulation described in Section 7.1.

Both codes were compiled with the same compiler and compiler options, including aggressive optimization. Tables 2 and 3 quantify the accuracy and efficiency of MERCURY6 and the baseline serial scheme of the present method, respectively. For this benign test problem, the adaptive Bulirsch–Stoer option of the hybrid symplectic integrator is not invoked. Thus MERCURY6 functions as a fixed-step integrator of second order. In this mode, MERCURY6 is extraordinarily efficient. Accuracy is controlled simply by specifying the time step Δt in days. Even with a relatively large time step of 20 days, MERCURY6 requires just 11 seconds of compute time for the full 50,000-year calculation, impressively preserving total energy (E) to seven significant digits and angular momentum (L) to thirteen, as indicated by their relative errors $|\Delta E/E|$ and $|\Delta L/L|$, respectively.

Preservation of energy to ten significant digits, however, requires a time step that is 20 times smaller and a computation time that is ten times larger (111.8 seconds, line 5 of Table 2). Round-off error apparently prevents the method from achieving higher accuracies by further reduction of the time step.

In contrast, for the present algorithm, global accuracy is controlled by adjusting the error tolerance ϵ, which constrains global error in velocity. Thus, one would expect energy to be conserved to roughly six places whenever $\epsilon = 10^{-3}$. This is confirmed by the first line in Table 3. At low to moderate accuracies, the present serial algorithm cannot compete in efficiency with the symplectic integrator. At the lowest accuracy ($|\Delta E/E| \approx 10^{-7}$), the symplectic integrator completes the simulation 70 times faster than the current serial algorithm. However, the advantage of the symplectic method diminishes somewhat when higher accuracy is demanded. For example, for $|\Delta E/E| \approx 10^{-10}$, the relative efficiency advantage of the symplectic method is reduced below ten. Moreover, for a modest 33% increase in computational cost, the present method, as shown in line 3 of Table 3 is able to further drive down $|\Delta E/E|$ by two orders of magnitude, which the second-order symplectic method is unable to do, presumably due to the accumulation of round-off error. Furthermore, were the symplectic integrator able to attain this level of precision, its compute time would be very nearly equivalent to that of the present method.

To summarize, the present method was designed for extraordinary accuracy. At low accuracies, it is not competitive with current state-of-the-art methods. At very high accuracy, it is.

The efficiency disadvantage of the current method is further atoned for by two significant advantages. First, the adaptive attributes of the present method render it well-suited for non-benign N-body systems that involve close encounters. Thus, there is no need for hybrid methodology. Second, as a corollary to the first advantage, the algorithm is compact and thus readily parallelized.

7.2.2. *Long-term evolution of the solar system*

Long-term evolution of the solar system remains of continuing interest, particularly in regard to its stability. To this end, we demonstrate the current algorithm by evolving the solar system forward in time for a nominal three million years.

The solar-system simulation presented here was inspired by the three-million year integration by Quinn et al. [11]. Their integration involved high-order (up to 12th) symmetric, multistep integrators. Although the authors have drawn inspiration from this source, for a number of reasons, we have not tried to replicate their work. Among these, Quinn et al. model several high-order corrections that are beyond the scope of this paper (but not the capability of the algorithm). These include relativistic effects, the effect of the finite quadrupole moment of the Earth–Moon system, and tidal-friction effects on the Moon's orbit.

Specifically, the sun and nine planets (including Pluto) are considered ($N = 10$) as point masses. As before, initial conditions are taken from *Ephemerides of the Minor Planets* for 1997, December 18.0. For the interpretation of graphical results to follow, the first two cartesian coordinates are in the nominal plane of the ecliptic, and the third coordinate is normal to the ecliptic. The natural time unit [NTU] is the mean Earth year (365.25689833 days) divided by 2π. The physical duration of the integration is 20 million NTUs, or roughly 3.18 million Earth years.

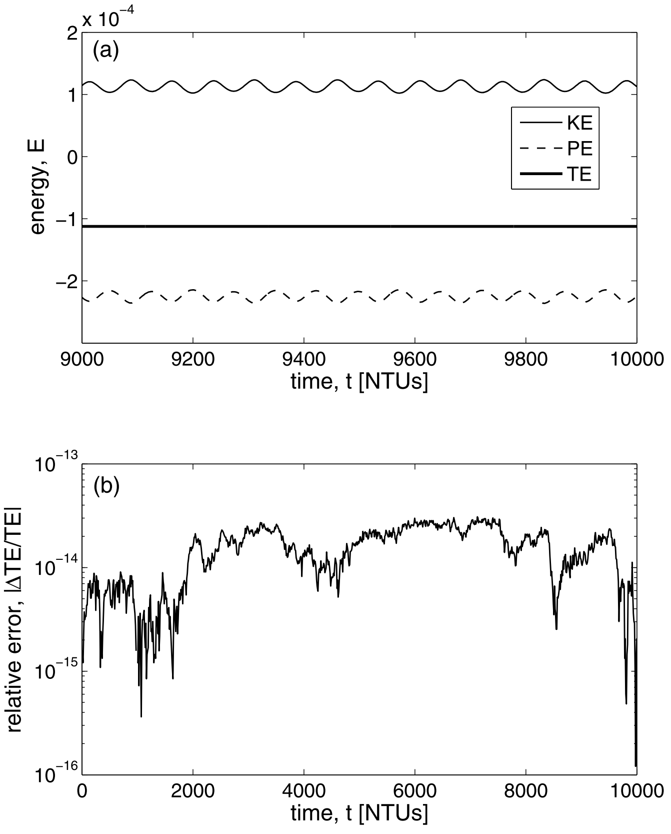

Fig. 6. Energy conservation by present scheme for 10,000 NTU solar-system evolution. Legend: (a) aggregate kinetic energy (KE, narrow solid line); aggregate potential energy (PE, dashed line); aggregate total energy (TE, thick solid line). (b) Scheme conserves total energy (TE) throughout integration to more than 13 significant digits.

Once again, the computational platform is that described above in Section 7.1. Because serial and parallel performance both improve by a factor of approximately three with aggressive optimization, the -O3 compiler option was set. The computation was conducted using the present parameter-free parallel algorithm on two processors ($p = 2$); that is, with five bodies per processor.

The current scheme is not explicitly designed to be symplectic; nevertheless, its high-order accuracy renders the scheme effectively symplectic. Fig. 6 presents aggregate kinetic energy (KE), aggregate potential energy (PE), and aggregate total energy (TE) for a preliminary integration of 10,000 NTUs using the parameter-free, parallel algorithm. Total energy is conserved to more than 13 significant digits. We note that the relationship of the time averages of KE and PE is as expected from the virial theorem. We further note that the variations in KE and PE on a 74.2 NTU (11.8-year) cycle are due primarily to the motion of Jupiter.

For this long-duration integration, one algorithmic concession was made. The relative global error tolerance was relaxed by several orders of magnitude under the assumption that several digits of accuracy would unavoidably be lost due to the accumulation of round-off errors. Specifically, the error tolerance was reset to 10^{-9}, and the algorithm did not run in its completely parameter-free mode. During the computation, which nevertheless invoked doubly adaptive optimization, the Maclaurin order varied from 6 to 24, with the usual range between 13 and 22. The total number of time steps was just over 239 million, or approximately 75 steps per Earth year on average, or a bit under 5 days per average step. The integration required 30.4 hours of clock time.

Table 4
Extrema of signed relative errors in total energy (E), total linear momentum (P_i), and total angular momentum (L_i).

	Signed rel. err.	Time [NTUs]
$\min(\Delta E/E)$	-5.024×10^{-10}	1.820×10^{7}
$\max(\Delta E/E)$	$+1.007 \times 10^{-9}$	1.439×10^{7}
$\min(\Delta P_1/P_1)$	-2.134×10^{-12}	1.858×10^{7}
$\max(\Delta P_1/P_1)$	$+1.600 \times 10^{-12}$	9.415×10^{6}
$\min(\Delta P_2/P_2)$	-4.198×10^{-13}	2.813×10^{6}
$\max(\Delta P_2/P_2)$	$+1.259 \times 10^{-12}$	1.783×10^{7}
$\min(\Delta P_3/P_3)$	-4.543×10^{-13}	1.713×10^{7}
$\max(\Delta P_3/P_3)$	$+1.060 \times 10^{-12}$	4.194×10^{6}
$\min(\Delta L_1/L_1)$	-4.341×10^{-10}	1.448×10^{8}
$\max(\Delta L_1/L_1)$	$+4.403 \times 10^{-10}$	1.729×10^{8}
$\min(\Delta L_2/L_2)$	-4.446×10^{-10}	6.291×10^{6}
$\max(\Delta L_2/L_2)$	$+2.497 \times 10^{-9}$	1.944×10^{7}
$\min(\Delta L_3/L_3)$	-4.027×10^{-10}	1.184×10^{7}
$\max(\Delta L_3/L_3)$	$+1.406 \times 10^{-9}$	1.878×10^{7}

Because the system is isolated, all components of aggregate linear momentum and aggregate angular momentum should be conserved. Table 4 presents the minimum and maximum relative errors incurred in all conserved quantities over the course of the integration, as well as the time [NTUs] at which each extremum occurred. The error extrema do not congregate at the end of the integration, which suggests that the dominant contributions to error are random rather than systematic. The largest relative error incurred for any conserved quantity was approximately 2.5×10^{-9}.

Orbital eccentricities vary over time, as do nominal radii and precession rates of perihelia. The planets with the largest eccentricities are Mercury, Mars, and Pluto, whose time-averaged precession rates are 572.4, 1628, and −34.88 arcseconds per century, respectively, as reported by the JPL Solar System Dynamics website (in Table 2a of "Keplerian Elements for Approximate Positions of the Major Planets" by E. M. Standish; http://ssd.jpl.nasa.gov/txt/p_elem_t2.txt). Of Mercury's precession rate, 43 arcseconds per century are attributable to relativistic effects. The remaining 529.4 arcseconds per century are due to the gravitational influences of the remaining planets, which are modeled here. (The calculated relativistic contributions to the precession rates for Mars and Pluto are much smaller: less than 2.0 and 0.1 arcseconds per century, respectively.)

We now summarize how well the present algorithm replicates time-averaged precession rates published by JPL. To this end, our calculated planetary positions were examined carefully during three 10,000-year "epochs" at the beginning, middle, and end, respectively, of the 3.18 million-year integration. During these epochs, orbital positions were output at every time step, and the data were probed to locate the perihelia of Mercury, Mars, and Pluto as functions of time, from which the precession rates were subsequently extracted.

Because the initial state of the system was not free of linear momentum, the entire system experienced significant drift. Thus, the initial step in data post-processing was to convert absolute positions to sun-relative positions. Subsequent determination of the position of planetary perihelia was numerically challenging, especially for Mercury. Because of the relatively large time steps taken by the high-order method, Mercury's position, for example, was recorded fewer than twenty times per orbit. However, during each epoch, Mercury completed more than 41,000 orbits. Of the relatively large number of position reports for each planet, a few hundred were determined numerically to closely approximate perihelion positions. Points representing perihelion positions are plotted in the nominal plane of the ecliptic in Figs. 7, 8, and 9. In sun-relative coordinates, the sun's position is fixed at the origin. Symbols identify perihelion positions near the beginning (squares)

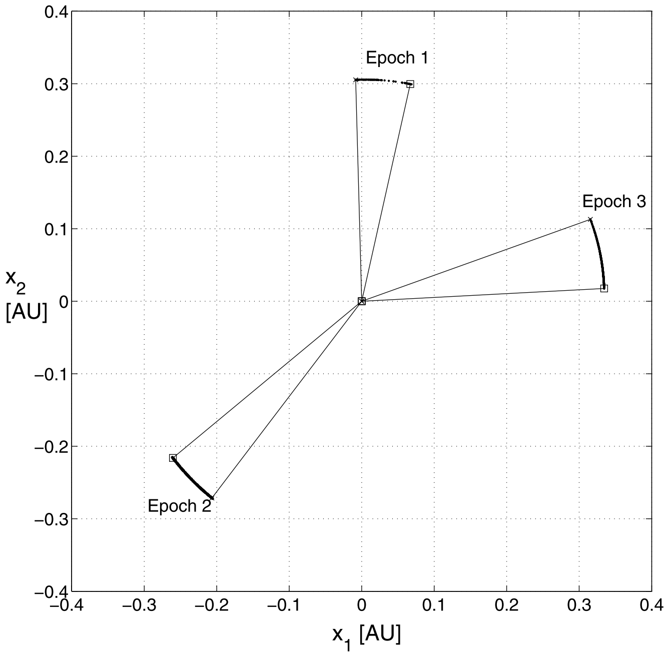

Fig. 7. Precession of perihelion during epochs 1, 2, and 3 for planet Mercury. Legend: dots (perihelion data); squares (location of perihelion near beginning of epoch); × (location of perihelion near end of epoch).

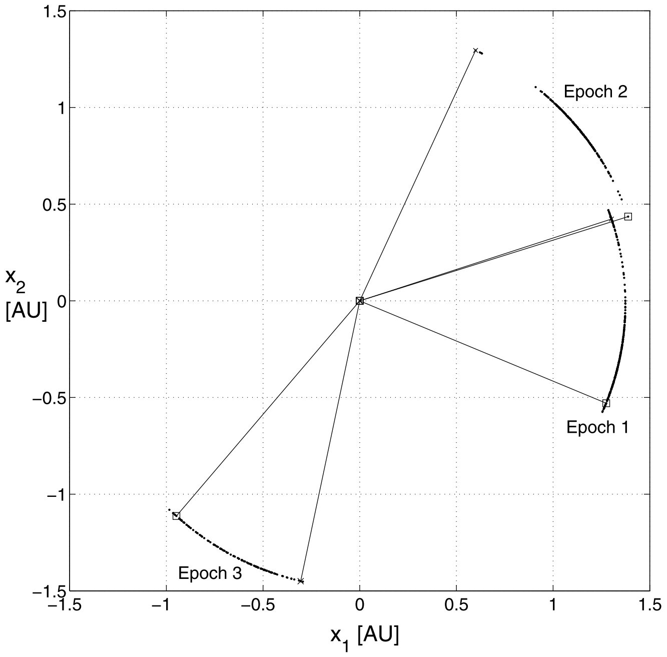

Fig. 8. Precession of perihelion during epochs 1, 2, and 3 for planet Mars. Legend: dots (perihelion data); squares (location of perihelion near beginning of epoch); × (location of perihelion near end of epoch).

and ending (×) of each epoch, respectively. The relative positions of the symbols indicate that Mercury and Mars precess counterclockwise. Pluto, in contrast, precesses clockwise. (This difference is attributable to the fact that Mercury and Mars orbit nearer the Sun than the massive planets Jupiter and Saturn, whereas Pluto's orbit is larger.) The mean precession rate for the epoch is simply the measure of the angle formed by solar rays through these two marked points, divided by the time that has elapsed between the two perihelion positions. Precession rates thus extracted numerically are presented in Table 5.

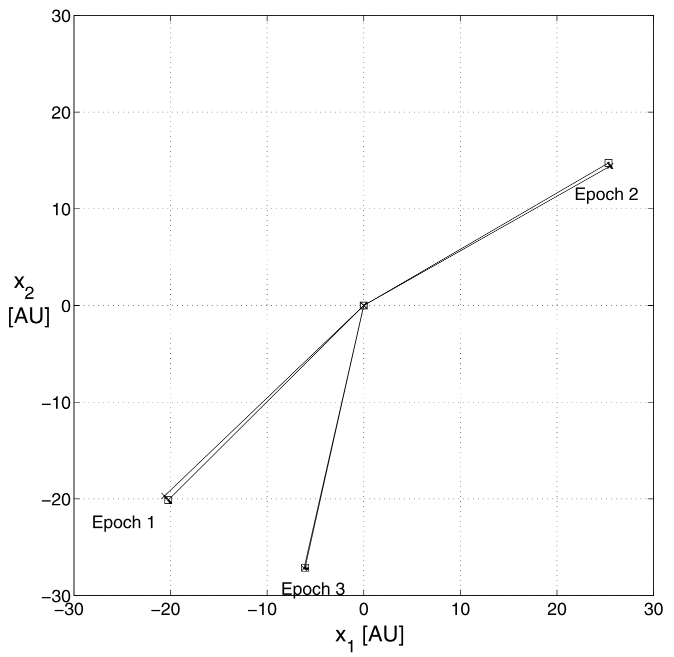

Fig. 9. Precession of perihelion during epochs 1, 2, and 3 for Pluto. Legend: dots (perihelion data); squares (location of perihelion near beginning of epoch); × (location of perihelion near end of epoch).

Table 5
Numerically calculated perihelion precession rates in arcseconds per century for Mercury, Mars, and Pluto. Legend: E1 (epoch 1, etc.); E1–2 (epochs 1–2, etc.); $[t_0, t_1]$ (time interval of integration in NTUs).

	t_0	t_1	Rate [s/Cy]	P. revolutions
Mercury				
E1	349.54	62,577.65	518.49	0
E2	10,000,362.40	10,062,319.64	484.06	0
E3	19,937,308.84	19,998,402.03	645.97	0
E1–2	349.54	10,000,362.40	520.47	6
E2–3	10,000,362.40	19,937,308.84	524.12	6
E1–3	349.54	19,937,308.84	**522.54**	12
Mars				
E1	35.84	62,776.27	1463.37	0
E2	10,000,211.07	10,062,727.14	1726.29	0
E3	19,937,204.19	19,999,873.29	1050.46	0
E1–2	35.84	10,000,211.07	1637.65	20
E2–3	10,000,211.07	19,937,204.19	1605.36	19
E1–3	35.84	19,937,204.19	**1621.49**	39
Pluto				
E1	1506.12	62,244.02	−41.72	0
E2	10,000,800.85	10,042,919.15	−37.61	0
E3	19,938,270.11	19,992,687.61	−7.55	0
E1–2	1506.12	10,000,800.85	−48.27	0
E2–3	10,000,800.85	19,938,270.11	−28.32	0
E1–3	1506.12	19,938,270.11	**−37.29**	0

Table 5 reveals that precession rates are subject to considerable variability on the time scale of a 10,000-year epoch. In contrast, precession rates for the inner planets are relatively stable for time scales on the order of one-million years, as shown in the table by lines E1–2, E2–3, and E1–3, which report mean precession rates during the first "half," second "half," and "full" duration of the integration, respectively. Note that the numerically derived mean precession rate of Mars for the full duration of the integration is 1621.5 arcseconds per century, a value within 1/2 percent of the JPL value. The corresponding value for Mercury's precession rate, 522.54 arcseconds per century, is somewhat over one percent off the JPL value (excluding the general-relativistic contribution which was not part of our study). Pluto's numerically derived precession rate is of the correct sign and within seven percent of the JPL value. Note that Pluto's perihelion did not complete even one full revolution during the 3.18-million year integration; a much longer integration time would be necessary to obtain a good approximation for the true long-term-average for Pluto's precession rate.

These results strongly suggest that the method is suitable for long-term solar-system integrations.

7.3. Core–halo instability

Having validated the parallel N-body algorithm with two venerable and relatively gentle problems, we now assess its performance on a tougher problem: the integration of "swarms" of particles such as that described in Section 5. Although such systems are easy to specify, they present extreme challenges for N-body integrators. Such systems exhibit "core-halo instability," in which gravitational forces occasionally gang up to eject a particle. Over time, as particles "boil off," the remaining system contracts in size. As it contracts, mean density increases, and close encounters increase in frequency. Numerical workhorses such as RK4 fail miserably because the time step needed to maintain tight error tolerances becomes so small that round-off errors overwhelm the calculation. Even a state-of-the-art integrator like Bulirsch–Stoer struggles, as indicated by a high percentage of failed steps [3].

For greater physical realism than that of the system considered in Section 5, the particle swarms we now examine are spherical and the initial velocity distributions are Maxwellian.

Fig. 10 shows the execution time of the fully optimized parallel algorithm for particle swarms that vary in number from $N = 16$ to $N = 480$. In all cases, the integration interval is $t \in [0, 0.5]$. Execution time depends upon many factors: number of particles N, particle density, specific initial conditions, and optimal Maclaurin polynomial order M and time increment Δt_l at each time level l. To remove two degrees of freedom, the particle density and the energy density are kept constant as the number of particles N is increased. Specifically, for each given value of N, a swarm radius and a velocity scale is predetermined by numerical experimentation such that the solution is non-singular during the finite integration interval. Equivalently, swarm radius and N establish particle density. Density and energy density are then both kept constant as N is varied by appropriately adjusting the spherical radius and "temperature" (velocity scale) of the particle swarm. This procedure tends also to somewhat stabilize the optimal order M.

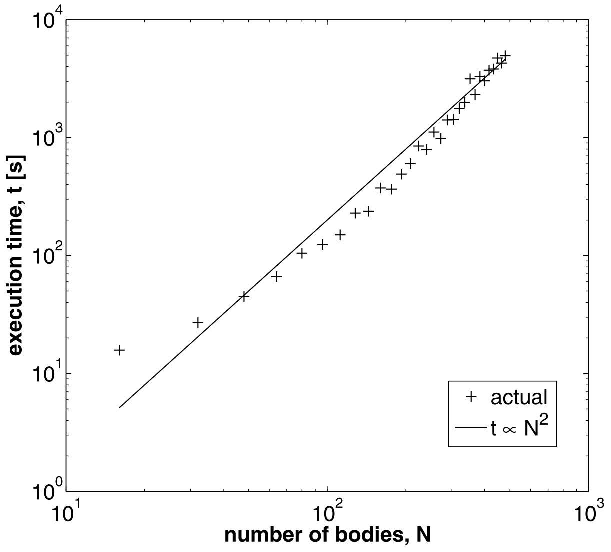

Fig. 10. Execution time in seconds vs. number of bodies N.

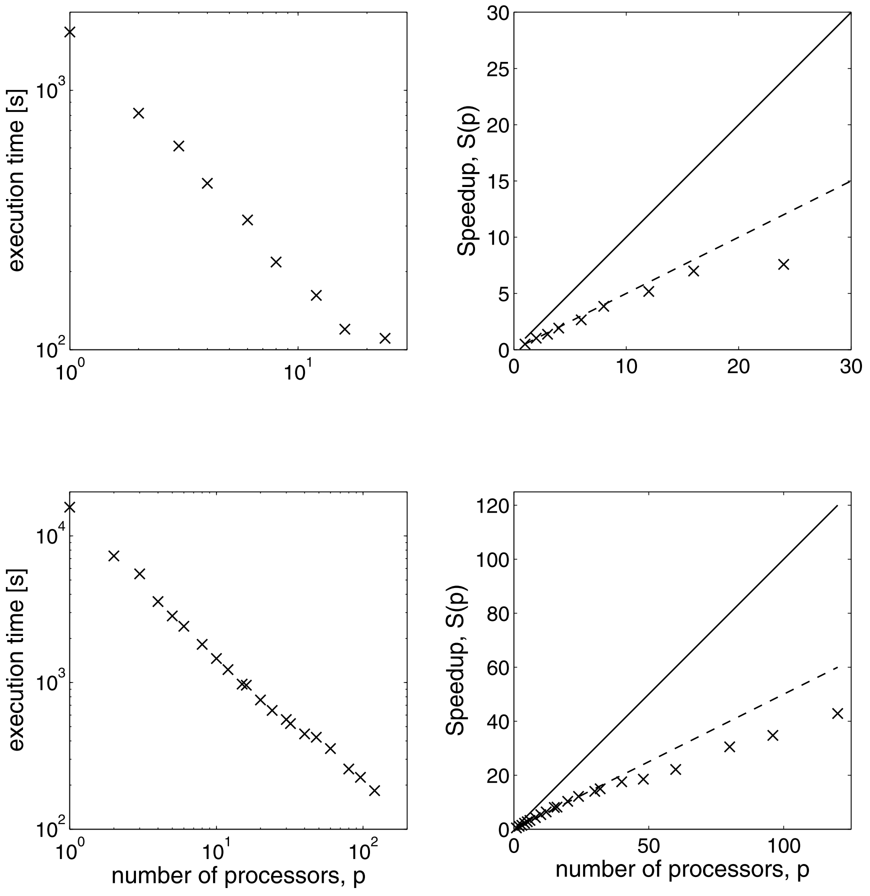

Fig. 11. Execution time in seconds and speedup $S(p)$ vs. number of processors p for 96-body system (upper plots) and 480-body system (lower plots). Legend: idealized linear speedup vs. p relative to process time for serial algorithm (solid line); idealized linear speedup vs. p relative to process time for parallel algorithm with $p = 1$ (dashed line); actual speedup (symbols).

Despite the considerable variability that remains due to randomized initial conditions, particle systems standardized in this way show predictable behavior. Execution times increase in close relationship to the asymptotic operation count, which scales as N^2, as shown in Fig. 10.

Finally, Fig. 11 demonstrates parallel performance. The computational platform used for parallel benchmarking is the Cedar Linux Cluster at the University of Virginia, which consists of several hundred nodes, each with 2 GB of RAM and two AMD Opteron CPUs each running at 2 GHz. Interconnections are by gigabit Ethernet with 60–110 MB/s bandwidth and 50–200 µsecs latency. Routine requests are limited to 128 concurrent CPUs. The compiler is **pgf90** with default options (no aggressive optimization). Specifically, Fig. 11 presents execution times and speedup vs. p for two N-body systems: the 96-particle system described in Section 5, and a spherical 480-particle system with a Maxwellian velocity distribution as described above. For integration of the former system, the global error tolerance is set at $\epsilon = 10^{-13}$. For integrating the larger system, the algorithm is run in parameter-free mode, where $\epsilon = 10u$, and u, determined during execution, is the machine unit round-off error. For a double-precision calculation on the specific operational platform, $u = 2.2 \times 10^{-15}$.

Here, speedup $S(p)$ is defined as the ratio of execution time of the serial algorithm to the execution time of the parallel algorithm on p processors. For comparison, ideal (linear) speedup, namely $S(p) = p$, is also shown.

Because the parallel algorithm cannot efficiently exploit symmetries (as discussed in the previous section), its computational workload is twice that of the baseline serial algorithm. Hence, ideal scalability for the parallel algorithm is $S(p) = \frac{p}{2}$. For reference, this linear relationship is also provided in the figure. On this system and with these compiler options, scalability is linear but somewhat less than ideal for $p \leqslant 120$. Mildly superlinear performance is sometimes observed and attributed to cache issues; each node has 1024 K cache.

As expected, parallel performance degrades when the distribution of work becomes too fine-grained; that is, when the number of bodies per processor is fewer than a handful. However, so long as $p \leqslant N/5$ (as suggested previously), the current parallel algorithm enjoys linear scalability for the range of values of N considered. We anticipate linear scalability with p for arbitrarily large N-body systems so long as the “handful” rule is obeyed. This optimistic projection is based on the following argument. If, at minimum granularity, $p = N/5$, then p is removed as a parameter, and R (see Eq. (23)) is independent of N.

8. How to use

Advantages of the current algorithm include practical simplicity, high accuracy, and linear scaling with increasing N. A disadvantage is that it is memory intensive because of large polynomial order M. However, processor memory continues to grow even as chip speeds have plateaued. Hence, algorithms that favor memory use to diminish operation counts may be riding a wave for the foreseeable future.

The current algorithm is compact; implemented in Fortran 90 and MPI, it runs to only 723 lines. Moreover, in its parameter-free mode, the present algorithm is extraordinarily easily to use. For all practical purposes, the only input values are the problem data itself: the number of bodies; the masses, initial positions, and initial velocities of those bodies; and the integration interval. What follows below is the header of a typical input data file:

```
32 3 /number of bodies (n); number of bodies output (nout)
28   /maximum Maclaurin polynomial order (mo)
0.E+0, 0.5D0, -.025 /time interval [a,b] and print interval (dtout)
-1.0E-14, .F. /global error tolerance (epsilon); diagnostics trigger
1.000000 -0.451781 -0.668711  2.130823 -0.390649  0.170636 -0.139339
1.000000  0.139627  2.417780 -0.184771  0.059581  0.064517  0.278170
```

The program is compiled and executed with commands **mpif90** and **mpirun**, respectively.

The first algorithmic step during parallel execution is performed by the MPI command **MPI_init**. Among other functions, the command establishes the number of processors p as specified in the **mpirun** command-line option and assigns integer identifiers 0 to $p-1$ to each processor.

After initialization, input data, in the format above, are read. Lines 1 and 2 are read first. Line 1 specifies the number of bodies N. Line 2 specifies the maximum allowed size M of the Maclaurin polynomial approximations. The three integers N, M, and p determine per-processor memory requirements. Memory is then allocated dynamically for all processors.

The integration time interval is defined in line 3; the global relative error tolerance is established in line 4. Wherever negative values are encountered as input data, default values are used. For example, the default value for the error tolerance is $\epsilon = 10u$. A positive print interval (**dtout** in line 3) ensures regular output intervals in time. Otherwise results are output at the natural (adaptive) time intervals. In either case, the time step is adaptive and the parameter effects only the regularity of the output. If desired, setting the diagnostics trigger to 'true' in line 4 invokes diagnostics routines that compute total energy and linear and angular momenta, from which, for example, the data in Fig. 6 and Table 4 were obtained. Each line following the four header lines specifies the mass, initial position coordinates, and initial velocity coordinates, respectively, of a body.

Although the algorithm appears to retain a few free parameters, this is largely illusion. By following the "handful of bodies per processor" guideline, the parameter p can be eliminated. If desired, M (Fortran variable **mo**) can be hard coded to some maximum. Based on our experience with double-precision calculations, this maximum should be somewhat less than 30. These steps, along with selecting the defaults for **nout**, ϵ, and **dtout**, render the algorithm essentially parameter-free, as advertised.

Unless otherwise specified by **nout**, an output file containing the time history of position and velocity is generated for each of the N bodies. These files are numbered sequentially starting with **particle0001**.

9. Conclusions

The current parallel and adaptive algorithm is appropriate for highly accurate integration of relatively large N-body systems. Moreover, the parallel algorithm, which has been demonstrated for systems of up to several hundred bodies, enjoys essentially linear speedup so long as there are at least a handful of bodies per processor. Finally, the methodology is readily applied to any deterministic system whose generator can be written in polynomial form. To date, we know of *no* deterministic system that cannot be cast as polynomial.

Acknowledgements

The first two authors are grateful to the University of Virginia (UVA) and Virginia Tech for underwriting their participation in the High Performance Parallel Computing Bootcamp, held at UVA, August 7–10 and 13–16, 2007. Special thanks is due M. Rosen and A. Grimshaw of UVA for extending the invitation to participate and for sponsoring off-site access to UVA's high-performance clusters. Without the expert guidance of the staff of UVA's Information Technology & Communications Division, this work could not have been satisfactorily completed. The authors are particularly indebted to J.E. Chambers of the Carnegie Institution of Washington for granting access to his hybrid symplectic integrator MERCURY6 and for gracious assistance in its use. Some computations were conducted on a parallel platform at James Madison University made possible by NSF SCREMS grant DMS-0821309. Finally, the authors are grateful to their colleague J. Martin for his wisdom in all things computing.

References

[1] G.E. Parker, J.S. Sochacki, Neural Parallel Sci. Comput. 4 (1996) 97.
[2] R.D. Stewart, W. Blair, J. Comput. Neurosci. 27 (2009) 115.
[3] C.D. Pruett, J.W. Rudmin, J.M. Lacy, J. Comput. Phys. 187 (2003) 298.
[4] E. Fehlberg, NASA TN D-2356 (1964).
[5] R. Broucke, Celestial Mech. 4 (1971) 110.
[6] J. Makino, Astrophys. J. 369 (1991) 200.
[7] W.H. Press, D.N. Spergel, Astrophys. J. 325 (1988) 715.
[8] E. Hairer, C. Lubich, G. Wanner, Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations, 2nd ed., Springer, Berlin, 2002, p. 179.
[9] H. Van de Vyver, Int. J. Mod. Phys. C 19 (2008) 1257.
[10] J.E. Chambers, Mon. Not. Roy. Astron. Soc. 304 (1999) 793.
[11] T.R. Quinn, S. Tremaine, M. Duncan, Astron. J. 101 (1991) 2287.