PASJ: Publ. Astron. Soc. Japan **44**, 141–151 (1992)

On a Hermite Integrator with Ahmad-Cohen Scheme for Gravitational Many-Body Problems

Junichiro MAKINO
Department of Information Science and Graphics, College of Arts and Sciences, The University of Tokyo, 3-8-1 Komaba, Meguro-ku, Tokyo 153
and
Sverre J. AARSETH
Institute of Astronomy, Madingley Road, Cambridge, CB3 0HA, England

(Received 1991 September 18; accepted 1992 February 7)

Abstract

We describe the implementation of the Ahmad-Cohen scheme based on a fourth-order Hermite integrator. With the fourth-order Hermite scheme, we calculate the force and the time derivative of the force analytically, and construct a third-order interpolation polynomial using two points in time. Compared with the standard scheme (Aarseth 1985) which is widely used, it allows a longer stepsize for the same accuracy, and the program is much simpler. In the case of the Ahmad-Cohen scheme, which uses different stepsizes for the forces from neighboring particles and that from distant particles, the difference in the programming complexity is even larger, since the Hermite scheme does not require corrections of the higher order divided differences for the forces from distant particles. On scalar computers the Hermite schemes are marginally faster than the standard scheme for the same level of accuracy, both with and without the Ahmad-Cohen scheme. On vector machines or special-purpose hardware, such as GRAPE, the Hermite scheme would be significantly faster since the number of scalar operation is much smaller. The gain in computing speed using the Ahmad-Cohen scheme is $(N/3.8)^{1/4}$ for both the standard and Hermite schemes, where N is the total number of particles. However, this gain can be significantly smaller on vector or parallel machines.

Key words: N-body simulation — Numerical method — Collisional systems

1. Introduction

The programs developed by Aarseth (1963, 1985) have been widely used for numerical integration of gravitational N-body systems. These programs are based on a fourth-order Adams-Bashforth-Moulton predictor-corrector scheme, modified so as to allow a variable stepsize and different stepsizes over particles. In addition, it allows different stepsizes for the force from neighboring particles and distant particles, so that the force from distant particles is calculated over much longer time intervals than the force from the neighbors (Ahmad and Cohen 1973). In the following, we denote the standard scheme with and without the Ahmad-Cohen scheme as the individual timestep scheme (ITS) and the Ahmad-Cohen scheme (ACS), respectively.

In this paper we compare the standard scheme with the Hermite scheme (Makino 1991a). In the Hermite scheme we use higher order derivatives which are explicitly calculated, in order to construct interpolation polynomials of the force. Makino (1991a) compared ITS and its equivalent Hermite schemes (HITS for short) with different orders, and found that for all orders tested (4 through 10) HITS allows timesteps longer than for ITS by roughly a factor of 2 for the same accuracy. However, this comparison is limited to an integrator without the Ahmad-Cohen scheme. Thus, it remains to be investigated how the Hermite integrator with the Ahmad-Cohen scheme (HACS for short) compares with the Ahmad-Cohen scheme. In this paper we describe the implementation of HACS and compare its speed and accuracy with that of ACS.

In section 2 we describe the Hermite integrator. According to Makino (1991a), the fourth-order scheme seems to be the best choice, unless we require extremely high accuracy. We therefore concentrate here on the fourth-order schemes. The fourth-order Hermite scheme has a very important advantage in that it can be implemented as a self-starting scheme. Therefore, both HITS and HACS are much simpler than ITS and ACS.

In section 3 we present the results of numerical tests. The integration error in the total energy of HITS is about 1/10 that of ITS for the same number of timesteps. With HITS, we can use timesteps that are about twice as long as that we require with ITS for the same accuracy, since the error is proportional to the fourth power of the timestep size. For Ahmad-Cohen schemes, we first determine the optimal set of parameters that minimizes the calculation cost for a given accuracy. The optimal number of neighbors is found to be $(N/4)^{3/4}$, confirming the analytical prediction by Makino and Hut (1988), where N is the total number of particles. For the Ahmad-Cohen schemes, the gain in accuracy by using the Hermite scheme is also about a factor of 10. The gain in computational speed that we obtain by using the Ahmad-Cohen scheme is $(N/3.8)^{1/4}$ for the case of a scalar computer.

In section 4 we discuss the efficiency of integration schemes for real hardware. If N is large, the cost of force calculation determines the total cost. The number of floating-point operations for Hermite schemes per timestep is about twice that for standard schemes, since we explicitly calculate the time derivative of the gravitational acceleration. The difference in the computation time per timestep ranges from less than 30% to nearly a factor of 2, depending on the hardware. In the case of vector/parallel computers or special-purpose machines, however, the calculation cost of the time integration itself cannot be neglected; in this case the Hermite scheme is better, since the number of timesteps is smaller and the calculation cost per timestep is also smaller. In the extreme case, the difference can be a factor 3 or larger.

2. Hermite Integration Scheme

In this section we describe the fourth-order Hermite scheme in detail. In section 2.1 we describe the individual timestep scheme (HITS). In section 2.2 we describe the implementation of the Ahmad-Cohen scheme. In section 2.3 we describe the algorithm to synchronize particles at the output time.

2.1. The Hermite Individual Timestep Scheme

In the Hermite individual timestep scheme, particle i has its own time (t_i), timestep (Δt_i), position ($\boldsymbol{x}_i$) and velocity ($\boldsymbol{v}_i$) at time t_i, and acceleration ($\boldsymbol{a}_i$) and time derivative of acceleration ($\dot{\boldsymbol{a}}_i$) calculated at time t_i. The integration proceeds according to the following steps:

(a) Select particle i with a minimum $t_i + \Delta t_i$. Set the global time (t) to be this minimum, $t_i + \Delta t_i$.

(b) Predict the positions and the velocities of all particles at time t using $\boldsymbol{x}$, $\boldsymbol{v}$, $\boldsymbol{a}$ and $\dot{\boldsymbol{a}}$.

(c) Calculate the acceleration ($\boldsymbol{a}_i$) and its time derivative ($\dot{\boldsymbol{a}}_i$) for particle i at time t, using the predicted positions and velocities.

(d) Calculate $\boldsymbol{a}_i^{(2)}$ and $\boldsymbol{a}_i^{(3)}$ using a Hermite interpolation based on $\boldsymbol{a}$ and $\dot{\boldsymbol{a}}$. Add the corrections to the position and velocity of particle i. Calculate the new timestep and update t_i.

(e) Go back to step (a).

The structure of the algorithm for HITS is exactly the same as that of ITS. Thus, both ITS and HITS are examples of a PEC (predictor-evaluator-corrector) scheme. There are, however, several differences in the details. In step (b), we predict the velocities as well as positions, since we require velocities to calculate $\dot{\boldsymbol{a}}$. The formulas used are

$$\boldsymbol{x}_{p,j} = \frac{(t-t_j)^3}{6}\dot{\boldsymbol{a}}_j + \frac{(t-t_j)^2}{2}\boldsymbol{a}_j + (t-t_j)\boldsymbol{v}_j + \boldsymbol{x}_j$$

and (1)

$$\boldsymbol{v}_{p,j} = \frac{(t-t_j)^2}{2}\dot{\boldsymbol{a}}_j + (t-t_j)\boldsymbol{a}_j + \boldsymbol{v}_j,$$

where j runs through all particles, including particle i. Since $\dot{\boldsymbol{a}}_j$ is calculated directly, all of the quantities used for the prediction are obtained by direct calculation, rather than by interpolation. The predictor of HITS requires no memory of the previous timesteps, and is therefore self-starting.

In ITS, the position of particle i (the particle to be updated) is predicted up to the order of $\boldsymbol{a}^{(3)}$ before the force calculation, while the other particles are predicted up to the order $\dot{\boldsymbol{a}}$. In HITS, we predict $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$ only up to $\dot{\boldsymbol{a}}$. The inclusion of higher order terms in the predictor does not change the order of the integrator and, therefore, it has only a small effect on the accuracy.

Using the position and velocity predicted by equation (1), we evaluate the acceleration and its time derivative for particle i according to the following:

$$\boldsymbol{a}_i = \sum_j Gm_j \frac{\boldsymbol{r}_{ij}}{(r_{ij}^2+\epsilon^2)^{3/2}}$$

and (2)

$$\dot{\boldsymbol{a}}_i = \sum_j Gm_j \left[\frac{\boldsymbol{v}_{ij}}{(r_{ij}^2+\epsilon^2)^{3/2}} + \frac{3(\boldsymbol{v}_{ij}\cdot\boldsymbol{r}_{ij})\boldsymbol{r}_{ij}}{(r_{ij}^2+\epsilon^2)^{5/2}}\right],$$

where

$$\boldsymbol{r}_{ij} = \boldsymbol{x}_{p,j} - \boldsymbol{x}_{p,i},$$
$$\boldsymbol{v}_{ij} = \boldsymbol{v}_{p,j} - \boldsymbol{v}_{p,i}, \quad (3)$$

and ϵ is the softening parameter. The corrector is based on the third-order Hermite interpolation constructed using $\boldsymbol{a}$ and $\dot{\boldsymbol{a}}$ at times t_i and $t_i + \Delta t_i$. The third-order Hermite interpolation polynomial is expressed as

$$\boldsymbol{a}_i(t) = \boldsymbol{a}_{0,i} + \Delta t\dot{\boldsymbol{a}}_{0,i} + \frac{\Delta t^2}{2}\boldsymbol{a}_{0,i}^{(2)} + \frac{\Delta t^3}{6}\boldsymbol{a}_{0,i}^{(3)}, \quad (4)$$

where $\Delta t = t - t_i$, $\boldsymbol{a}_0$ and $\dot{\boldsymbol{a}}_0$ are the acceleration and its time derivative calculated at time t_i; $\boldsymbol{a}_{0,i}^{(2)}$ and $\boldsymbol{a}_{0,i}^{(3)}$ are given by

$$\boldsymbol{a}_{0,i}^{(2)} = \frac{-6(\boldsymbol{a}_{0,i} - \boldsymbol{a}_{1,i}) - \Delta t_i(4\dot{\boldsymbol{a}}_{0,i} + 2\dot{\boldsymbol{a}}_{1,i})}{\Delta t_i^2},$$

and (5)

$$\boldsymbol{a}_{0,i}^{(3)} = \frac{12(\boldsymbol{a}_{0,i} - \boldsymbol{a}_{1,i}) + 6\Delta t_i(\dot{\boldsymbol{a}}_{0,i} + \dot{\boldsymbol{a}}_{1,i})}{\Delta t_i^3},$$

where $\boldsymbol{a}_{1,i}$ and $\dot{\boldsymbol{a}}_{1,i}$ are the acceleration and its derivative at time $t_i + \Delta t_i$. The correction formulae for the position and velocity are expressed as

$$\boldsymbol{x}_i(t_i + \Delta t_i) = \boldsymbol{x}_{p,i} + \frac{\Delta t_i^4}{24}\boldsymbol{a}_{0,i}^{(2)} + \frac{\Delta t_i^5}{120}\boldsymbol{a}_{0,i}^{(3)},$$

and (6)

$$\boldsymbol{v}(t + \Delta t_i) = \boldsymbol{v}_{p,i} + \frac{\Delta t_i^3}{6}\boldsymbol{a}_{0,i}^{(2)} + \frac{\Delta t_i^4}{24}\boldsymbol{a}_{0,i}^{(3)}.$$

For the timestep, we use the standard formula (Aarseth 1985), which is expressed as

$$\Delta t_i = \sqrt{\eta \frac{|\boldsymbol{a}_{1,i}||\boldsymbol{a}_{1,i}^{(2)}| + |\dot{\boldsymbol{a}}_{1,i}|^2}{|\dot{\boldsymbol{a}}_{1,i}||\boldsymbol{a}_{1,i}^{(3)}| + |\boldsymbol{a}_{1,i}^{(2)}|^2}}, \tag{7}$$

where η is a parameter that controls the accuracy. The timestep is proportional to $\sqrt{\eta}$, and the integration error is expected to be proportional to η^2. For fourth-order schemes the timestep formula (7) shows quite good behavior (Makino 1991a). The time derivatives in formula (7) have values at the new time, $t_i + \Delta t_i$. The values of $\boldsymbol{a}_{1,i}$ and $\dot{\boldsymbol{a}}_{1,i}$ are already known, since we calculated them directly. The value of $\boldsymbol{a}_{1,i}^{(3)}$ is the same as that of $\boldsymbol{a}_{0,i}^{(3)}$, because we use third-order interpolation. The second-order derivative, $\boldsymbol{a}_{1,i}^{(2)}$, is calculated as

$$\boldsymbol{a}_{1,i}^{(2)} = \boldsymbol{a}_{0,i}^{(2)} + \Delta t_i \boldsymbol{a}_{0,i}^{(3)}. \tag{8}$$

When we start the time integration, these higher order terms are not available. In our implementation of the Hermite scheme, we calculate the higher order terms explicitly and use them to determine the timestep, following Aarseth (1985). To implement the Hermite integration scheme, we use NBODY1 and NBODY2 (Aarseth 1985) as a base (ITS and ACS, respectively). Since these codes calculate the higher order derivatives in the startup routine, we were able to implement the starting procedure without any extra programming effort.

It is also possible to use a special formula for the initial timestep. A simple formula is

$$\Delta t = \eta_s \frac{|\boldsymbol{a}|}{|\dot{\boldsymbol{a}}|}, \tag{9}$$

with a sufficiently small value of η_s. We found that formula (9) gives sufficient accuracy with $\eta_s \sim 0.01$, provided that the velocities of the particles are not very small. The increase in the calculation cost is rather small, since the timestep is adjusted to the appropriate value fairly quickly. Even when we impose an upper limit to the increase of the stepsize as, say, a factor of 1.2, which is used in NBODY1/2, the timestep can increase by a factor of 10 in about 10 timesteps.

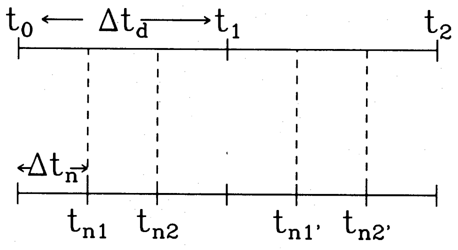


Fig. 1. Time sequence for the Hermite Ahmad-Cohen scheme.

2.2. *Hermite Ahmad-Cohen Scheme*

Figure 1 shows how the Ahmad-Cohen scheme (ACS) works for one particle. The gravitational force on a particle is divided into two components, one from its neighbors and the other from distant particles. In ACS, these two components are updated using different timesteps, Δt_n for the neighbors and Δt_d for distant particles. Thus, at certain timesteps we calculate only the forces from neighbors (neighbor step), while at other timesteps we calculate both the forces from neighbors and distant particles (distant step). Whether a particle is a neighbor or not is determined by the distance. If this particle is within the distance r_n from the particle for which we calculate the acceleration, it is included in the list of neighbors.

When we start the time integration, we first create a list of the neighbors; at the same time we calculate the acceleration due to neighbors $\boldsymbol{a}_n$, and that due to distant particles $\boldsymbol{a}_d$. In HACS, we also calculate their time derivatives. At the first neighbor step (time t_{n1}), we predict the position of the particle using the total acceleration and its derivatives, which are the sum of the contributions from the neighbors as well as those from distant particles. At time t_{n1}, we calculate $\boldsymbol{a}_n$ only, and not $\boldsymbol{a}_d$. In order to apply the corrector, we need the total acceleration at time, t_{n1}. This is obtained by extrapolating the acceleration ($\boldsymbol{a}_d$) by

$$\boldsymbol{a}_d(t + \Delta t) = \boldsymbol{a}_d(t) + \Delta t \dot{\boldsymbol{a}}_d(t). \tag{10}$$

We apply the corrector in the same way as in HITS. At the next neighbor step, time t_{n2}, we again predict the position and velocity of the particle. The total acceleration at time t_{n1} is approximated as the sum of $\boldsymbol{a}_n$ calculated at time t_{n1} and $\boldsymbol{a}_d$ predicted using equation (10). The time derivative of the total acceleration at t_{n1} is just the sum of $\dot{\boldsymbol{a}}_n(t_{n1})$ and $\dot{\boldsymbol{a}}_d(t_0)$, since we neglect the change in $\dot{\boldsymbol{a}}_d$ during the interval Δt_d, which is larger than $t_{n1} - t_0$. Using the predicted position we calculate the new $\boldsymbol{a}_n$ and $\dot{\boldsymbol{a}}_n$, and apply the corrector.

When we reach time $t_1 = t_0 + \Delta t_d$ (distant step), we need to calculate the following:

i) The acceleration and its derivative due to neighbors based on the list of neighbors calculated at time t_0. We denote these $\boldsymbol{a}_{n,\text{old}}$ and $\dot{\boldsymbol{a}}_{n,\text{old}}$, respectively.

ii) The acceleration $\boldsymbol{a}_{d,\text{old}}$ and its derivative $\dot{\boldsymbol{a}}_{d,\text{old}}$ due to distant particles, also based on the list of neighbors calculated at time t_0.

iii) The new neighbor list calculated using the positions of particles at time t_1.

iv) The acceleration $\boldsymbol{a}_{n,\text{new}}$ and its derivative $\dot{\boldsymbol{a}}_{n,\text{new}}$ due to neighbors based on the new list of neighbors created at time t_1.

v) The acceleration $\boldsymbol{a}_{d,\text{new}}$ and its derivative $\dot{\boldsymbol{a}}_{d,\text{new}}$ due to distant particles based on the new list of neighbors created at time t_1.

The first two are necessary in order to apply the corrector for both accelerations due to neighbors and distant particles. The corrector for the acceleration due to neighbors is calculated in the same way as in the neighbor steps. The corrector for the acceleration due to distant particles is also calculated in a similar way. We construct the third-order Hermite interpolation of the acceleration, $\boldsymbol{a}_d$, using the acceleration and its time derivative at time t_0 and t_1, and integrate it from t_0 to t_1.

In our present scheme, the accelerations based on the old and new neighbor list are calculated in the following steps, in the same way as described in Ahmad and Cohen (1973) and Aarseth (1985):

step 1: Calculate $\boldsymbol{a}_{n,\text{old}}$ and $\dot{\boldsymbol{a}}_{n,\text{old}}$ in the same way as a neighbor step.

step 2: Apply the corrector calculated for the acceleration due to neighbors.

step 3: Construct a new neighbor list and calculate the accelerations and their derivatives based on this list.

step 4: Calculate the acceleration and its derivative due to the distant particles based on the old neighbor list. They are calculated as

$$\begin{aligned} \boldsymbol{a}_{d,\text{old}} &= (\boldsymbol{a}_{d,\text{new}} + \boldsymbol{a}_{n,\text{new}}) - \boldsymbol{a}_{n,\text{old}} \\ \dot{\boldsymbol{a}}_{d,\text{old}} &= (\dot{\boldsymbol{a}}_{d,\text{new}} + \dot{\boldsymbol{a}}_{n,\text{new}}) - \dot{\boldsymbol{a}}_{n,\text{old}}. \end{aligned} \tag{11}$$

step 5: Apply the corrector for the acceleration due to distant particles using $\boldsymbol{a}_{d,\text{old}}$ and $\dot{\boldsymbol{a}}_{d,\text{old}}$.

In ACS, higher order derivatives, or more precisely the divided differences used to generate higher order derivatives, must be corrected so that they reflect any change in the membership of the neighbor list. With HACS, we do not need this correction as far as the time integration is concerned. ACS is based on a Newtonian four-point interpolation. Therefore, we need to know the acceleration calculated for the updated neighbor list at three previous times, separately for $\boldsymbol{a}_n$ and $\boldsymbol{a}_d$. However, the quantities calculated in the previous timesteps are based on the old neighbor list. In order to correct the old accelerations, first we make a list of new neighbors which are not present in the old neighbor list, and a list of old neighbors which are not present in the new neighbor list. For each particle in these two lists, we calculate the analytical time derivatives of its contribution to the acceleration up to $\boldsymbol{a}^{(3)}$, and correct the time derivatives of $\boldsymbol{a}_n$ and $\boldsymbol{a}_d$ using them. In HACS, we do not need to correct the higher order derivatives for the time integration. All of the complicated procedures described above are not necessary. Thus, a HACS integrator is much simpler than an ACS integrator.

The timestep for the next distant step, and the size of new neighbor sphere, r_n, are calculated in the standard way (Aarseth 1985). The timesteps for neighbor and distant steps are calculated as

$$\Delta t_n = \sqrt{\eta_n \frac{|\boldsymbol{a}||\boldsymbol{a}_n^{(2)}| + |\dot{\boldsymbol{a}}_n|^2}{|\dot{\boldsymbol{a}}_n||\boldsymbol{a}_n^{(3)}| + |\boldsymbol{a}_n^{(2)}|^2}}$$

and (12)

$$\Delta t_d = \sqrt{\eta_d \frac{|\boldsymbol{a}_d||\boldsymbol{a}_d^{(2)}| + |\dot{\boldsymbol{a}}_d|^2}{|\dot{\boldsymbol{a}}_d||\boldsymbol{a}_d^{(3)}| + |\boldsymbol{a}_d^{(2)}|^2}}.$$

These criteria have the same form as formula (7), except that the terms are those of the acceleration due to the neighbors or distant particles. For the neighbor step, we use the total acceleration, $|\boldsymbol{a}|$, instead of $|\boldsymbol{a}_n|$. Compared to the formula that uses $|\boldsymbol{a}_n|$, we found that this formula requires a fewer number of timesteps for roughly the same accuracy. In addition, it makes comparisons between ITS and ACS easier, since with equation (12) the size of neighbor step, Δt_n, is roughly the same as the stepsize of ITS, if $\eta_n = \eta$. With our formula, the stepsize of the neighbor step is controlled so that the error of the neighbor force compared to the total force remains constant. With the formula that uses $|\boldsymbol{a}_n|$, the relative error of the neighbor force, itself, is kept constant. Thus, the timesteps become unnecessarily small if the neighbor force is small compared to the total force.

For a distant step, we use the distant acceleration, $\boldsymbol{a}_d$, in order to ensure the stability of the integrator. In principle, we could also use the total acceleration here. However, such a formula tends to result in a too large distant timestep, since the distant force can become much smaller than the total force. For example, when a particle undergoes a close encounter with another particle, the total force is dominated by the contribution from the nearest particle. If we were to determine the distant step using the total force, the stepsize would be far too large. We therefore adopted a formula that uses $\boldsymbol{a}_d$. With our formula the timestep of the distant step is adjusted so that the relative error of the distant force, itself, is constant. If the distant force is much smaller than the total force, the distant force is integrated with unnecessarily high accuracy. However, it is difficult to avoid this loss without causing a stability problem.

In the original ACS described in Aarseth (1985), the number of neighbors is stabilized to a value that depends on the local density. In our test calculation described in section 3, however, we stabilize the number of neighbors to a constant value which does not depend on the local density. The difference in the calculation cost and accuracy caused by this modification is rather small, but it simplifies the analysis of the result.

2.3. Synchronization of Particles at Output Time

When we calculate some physical quantity of the system, such as the total energy at a certain time, we need the positions and velocities of all particles to an order consistent with the order of the time integration. Since we are using individual timesteps, the time to produce output does not synchronize with the times of particles. Thus, we have to predict the position and velocity of particles at the output time to the order of $\boldsymbol{a}^{(3)}$.

In ITS or ACS, it is simple to perform this prediction, since the prediction procedure is the same as that used in the integrator, itself. In Hermite schemes, however, we do not have the necessary higher order derivatives, since they are not used in the time integration.

In HITS, we can easily calculate the higher order derivatives. In fact, as described in section 2.1, we have already calculated them to determine the new timestep. Therefore, all we need to do is to save these higher order derivatives in memory.

In HACS, it is slightly more difficult to obtain higher order derivatives. At a distant step (t_1 in figure 1), we calculate the accelerations $\boldsymbol{a}_n$ and $\boldsymbol{a}_d$, for both the old and new neighbor lists. At time t_0 we calculate $\boldsymbol{a}_d$, and at time t_{nk} we calculate $\boldsymbol{a}_n$, both based on the old neighbor list constructed at time t_0. Thus, we can construct higher order derivatives separately for $\boldsymbol{a}_d$ and $\boldsymbol{a}_n$ at time t_1.

When we reach the first neighbor step after the distant step, time $t_{n1'}$, we calculate $\boldsymbol{a}_n$ using the new neighbor list, and apply the corrector using $\boldsymbol{a}_n$ at time t_1 calculated using the new neighbor list. Therefore, higher order derivatives of $\boldsymbol{a}_n$ at neighbor steps are calculated using the new neighbor list, while the higher order derivatives of $\boldsymbol{a}_d$ are still based on the old neighbor list constructed at time t_0. After the first neighbor step, therefore, the higher order derivatives of $\boldsymbol{a}_d$ and $\boldsymbol{a}_n$ are not consistent.

At present, we apply a correction of the higher order derivatives in the same way as in ACS, but only when the next distant time exceeds the next output time. Since these higher order derivatives are used only to calculate diagnostics, they are not used if we do not require diagnostics before the next distant timestep. In that case, therefore, we do not have to calculate the higher order derivatives.

If we use the hierarchical timestep algorithm (McMillan 1986; Makino 1991b), all particles are synchronized without any extra cost, at the longest timesteps. If the interval to calculate the output is longer than the average size of timesteps, the hierarchical timestep is, perhaps, a better choice, since in that case we do not have to calculate the 2nd- and 3rd-order derivatives analytically.

3. Performance and Accuracy

In this section we present the results of numerical comparisons between the standard schemes (ITS and ACS) and their equivalent Hermite schemes (HITS and HACS). We use the Plummer model as an initial condition, integrate it for one crossing time and measure the total energy at every 1/8 crossing time; we then calculate the r.m.s. change of energy. We use the standard system of units where the gravitational constant (G) and the total mass of the system (M) are both unity and the total energy of the system is $E = -1/4$. In this system, the crossing time is $2\sqrt{2}$. We use a softening parameter of size $4/N$, where N is the number of particles. For all calculations, we used a SPARC station 1 and 1+. All calculations are performed in 64-bit precision.

In section 3.1 we present the result for ITS and HITS. In section 3.2 we compare ACS and HACS.

3.1. Individual Timestep Schemes

The calculation cost and accuracy of ITS and HITS are controlled by only one parameter, the accuracy parameter (η) used to determine the timestep in equation (7). We vary η from 0.04 to 0.0025 and plot the calculated energy error as a function of n, the number of timesteps per particle per crossing time. Figure 2 shows the result for $N = 25, 100, 400$. For all values of N, the error is inversely proportional to the fourth power of the number of timesteps, as expected from the order of the integrator. For the same number of timesteps, the error of HITS is typically 1/10 that of ITS. In other words, HITS allows $10^{1/4} \simeq 1.8$ times larger timestep than ITS for the same accuracy.

Figure 3 shows the number of timesteps (n) per particle as a function of the total particle number (N), for $\eta = 0.02$. Roughly speaking, the dependence is $n \propto N^{1/3}$. In other words, for this range of N, the size of timestep is determined by the interparticle distance.

3.2. Ahmad-Cohen Schemes

With the Ahmad-Cohen scheme, three parameters control the accuracy and calculation cost: η_n, the accuracy parameter for the stepsize of neighbor steps, η_d, the accuracy parameter for the stepsize of distant steps, and N_n, the number of particles in the neighbor list.

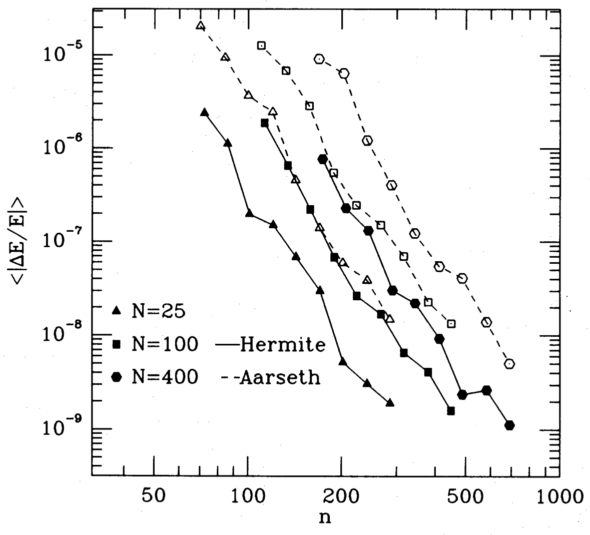


Fig. 2. Average energy error as a function of the number of timesteps per particle per crossing time. The individual timestep scheme is used. The solid and dashed curves indicate HITS and ITS, respectively. The triangles are for $N = 25$, squares for $N = 100$, and hexagons for $N = 400$.

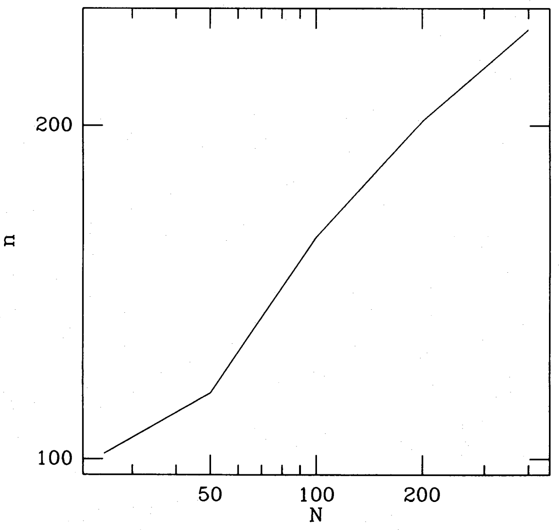


Fig. 3. Average number of timesteps per particle per crossing time as a function of the number of particles (N). The accuracy parameter (η) is 0.02. Both axes are logarithmic. The number of particles (N) used is 25, 50, 100, 200 and 400.

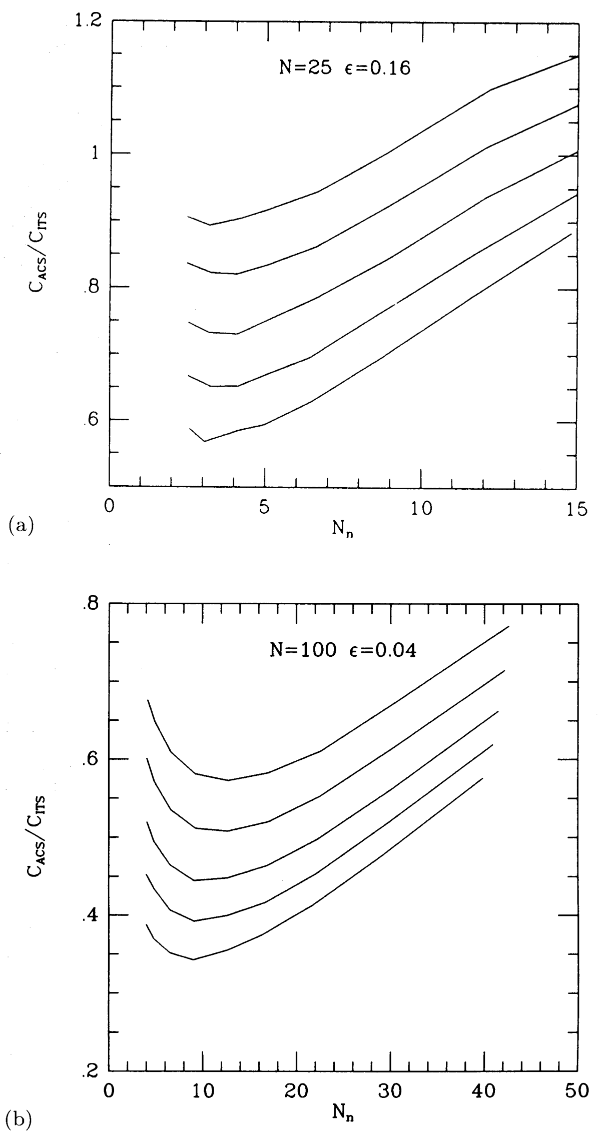
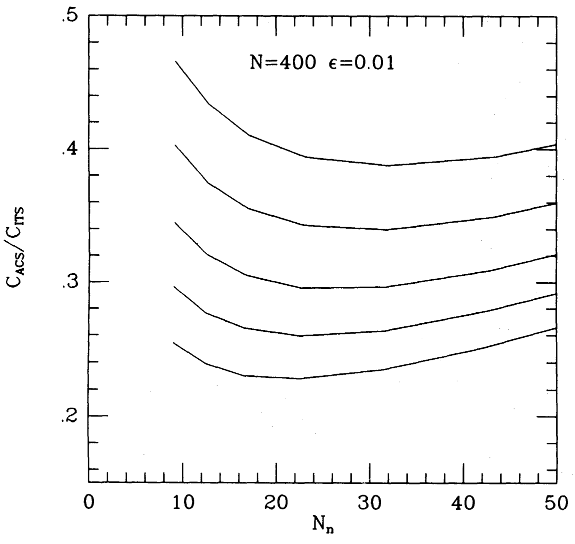


Fig. 4. Relative cost of ACS over ITS, plotted as a function of the number of neighbors. The accuracy parameter for neighbor steps (η_n) is 0.02. The accuracy parameter for distant steps (η_d) is 0.01, 0.014, 0.02, 0.028, 0.04, from top to bottom. (a) $N = 25$, (b) $N = 100$, (c) $N = 400$.

The number of neighbors is known to have only a weak effect on the accuracy, while η_n and η_d control the accuracy through the size of the timestep. In the following, therefore, we first determine the optimal N_n which minimizes the calculation cost for some given η_n and η_d. Then, with this N_n, we determine the relation between η_n and η_d that provides the best accuracy for a given calculation cost, and finally for this optimal set of parameters, compare the calculation cost of ACS and of HACS, as well as those of (H)ITS. For the optimal neighbor number, we use the value obtained for the ACS scheme. The value obtained for the HACS scheme is quite similar. For the ratio between η_n and η_d, we show the results for both schemes, which are again similar to each other.

3.2.1. *Optimal neighbor number*

Figure 4 shows the ratio between the calculation cost per timestep of ACS and that of ITS as a function of the



(c) Fig. 4. (Continued)

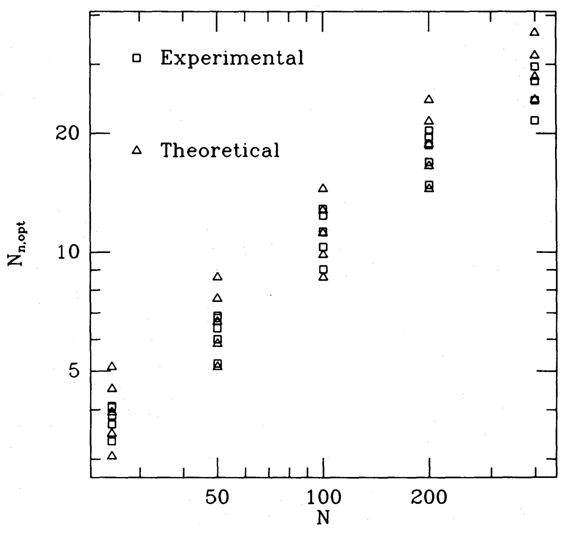


Fig. 5. Optimal number of neighbors as a function of N. The squares are obtained experimentally. The triangles indicate the theoretical prediction of equation (14). The accuracy parameters are same as in figure 4. For the same N, a larger $N_{n,opt}$ corresponds to a smaller η_d.

number of neighbors (N_n), for several different values of N and η_d. We have used $\eta_n = 0.02$ for all runs. The calculation cost per timestep is defined as

$$C_{\rm ITS} = k_1 N$$
and
$$C_{\rm ACS} = k_1 \left(N_n + N\frac{< n_d >}{< n_n >}\right), \tag{13}$$

where k_1 is the CPU time to perform one prediction and one force calculation; $< n_d >$ is the average number of neighbor steps per crossing time, and $< n_d >$ is the average number of distant steps per crossing time. We neglect here the cost of calculations other than those of the prediction and force calculation. The relative cost ($C_{\rm ACS}/C_{\rm ITS}$) has a minimum at a certain value of N_n. This optimal value of N_n, ($N_{n,\rm opt}$), is larger for larger N and smaller η_d. In figure 5 we plot $N_{n,\rm opt}$ as a function of N for several different η_d. The triangles in figure 5 indicate the theoretical prediction of the optimal N_n by Makino and Hut (1988), which is expressed as

$$N_n = \left(\sqrt{\frac{\eta_n}{\eta_d}}\frac{N}{4}\right)^{3/4} \tag{14}$$

in the case of timestep formula (12). The experimental result and the theoretical prediction show fairly good agreement.

3.2.2. Ratio between neighbor and distant steps

Figure 6 shows the energy error as a function of the ratio between the accuracy parameters (η_d/η_n), for several values of η_n. The number of neighbors (N_n) is chosen according to equation (14). Solid curves are for HACS, and the dashed curves are for ACS. Roughly speaking, if $\eta_n > \eta_d$, the error becomes smaller as we reduce η_n. If $\eta_n < \eta_d$, the error is determined by η_d and does not depend on η_n. For small N, we can clearly see this tendency. For the case of $N = 400$, however, this tendency is difficult to see. This is probably because the timestep formula for the distant step [equation (12)] is not optimal, as discussed in section 2.2.

We set η_n and η_d to be equal in order to compare the different schemes. Though for large N it might not be the true optimal point, the dependence of the calculation cost is relatively weak, as will be shown in section 3.2.4.

3.2.3. Comparison of HACS and ACS

Figure 7 shows the energy error as a function of the number of timesteps per particle. For these runs, we set $\eta_n = \eta_d$. The number of neighbor particles is chosen according to equation (14). The error of HACS is a factor of 5–10 smaller than that of ACS for the same number of timesteps. This behavior is roughly the same as that shown in figure 2, though the gain by the Hermite scheme is slightly smaller for ACS than for ITS.

3.2.4. Relative merit of (H)ACS over (H)ITS

Here, we take into account only the cost of the force calculation and the prediction. The contribution of the other calculations is discussed in the next section. In the following, we compare the calculation cost per timestep for (H)ITS and (H)ACS, neglecting the difference in the accuracy obtained for the same number of timesteps. In

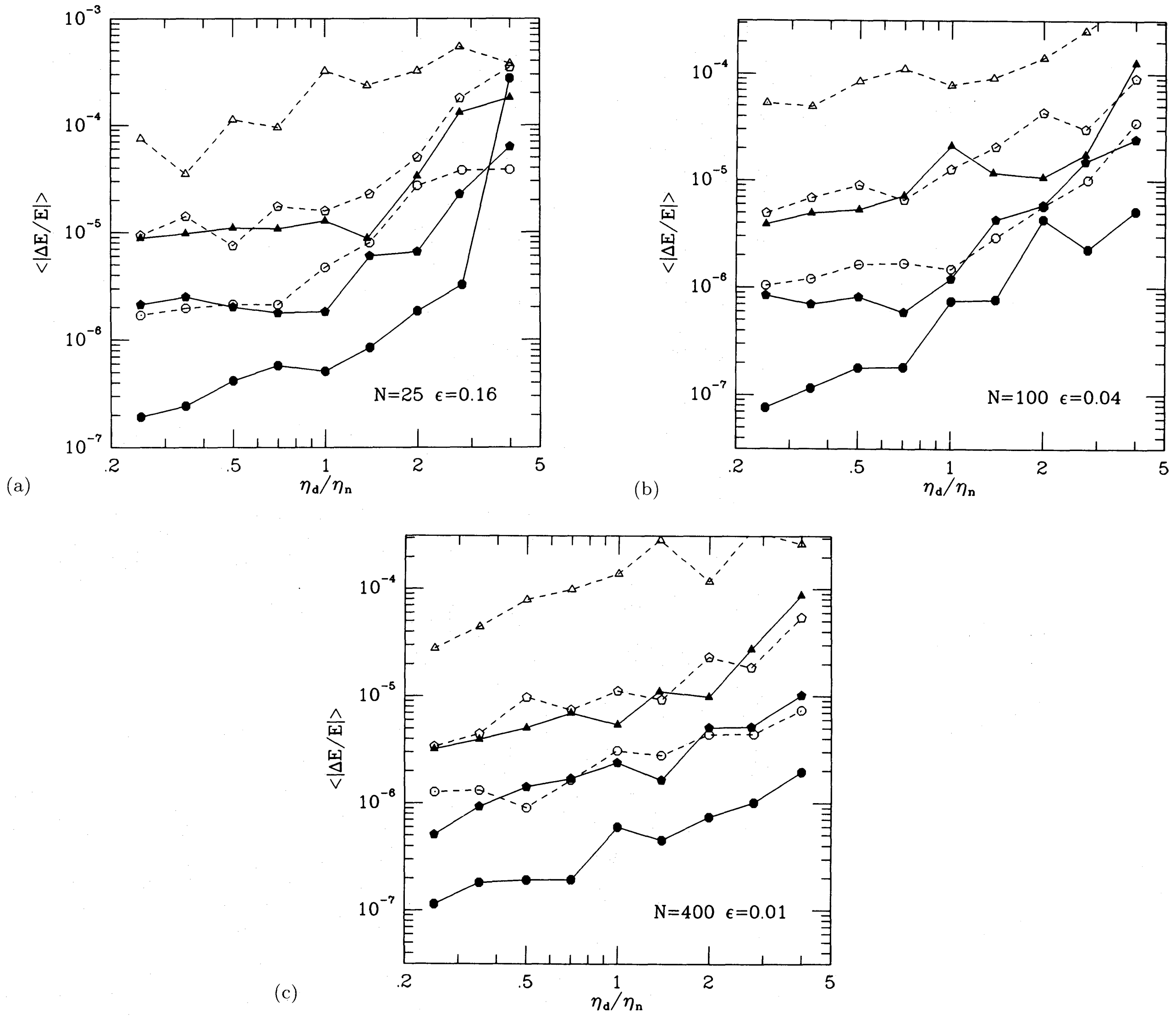


Fig. 6. Energy error as a function of the ratio between the accuracy parameters (η_d/η_n). The solid curves represent the results of HACS, and the dashed curves are for ACS. The triangles, pentagons and octagons indicate for $\eta_n = 0.08, 0.02$, and 0.005, respectively. (a) $N = 25$, (b) $N = 100$, (c) $N = 400$.

the case of ITS and ACS, the obtained accuracy is practically the same. In the case of HITS and HACS, it seems that HACS is slightly less accurate. Anyway, the difference is rather small and does not make a significant difference in the conclusion.

Figure 8 shows the relative cost of (H)ACS over (H)ITS, defined as $C_{\rm ACS}/C_{\rm ITS}$ (see section 3.2.1), as function of N. The accuracy parameters are $\eta = \eta_n = \eta_d = 0.02$. The number of neighbors is chosen according to equation (14). Makino and Hut (1988) predicted that this relative cost is given by

$$\frac{C_{\rm ACS}}{C_{\rm ITS}} = \left\{\frac{\sqrt{2}}{\Gamma(2/3)} + \frac{1}{[3\Gamma(2/3)]^{3/4}}\right\}\left(\frac{\eta_n}{\eta_d}\right)^{3/8} N^{-1/4}$$
$$= \left(\frac{\eta_n}{\eta_d}\right)^{3/8}\left(\frac{N}{3.8}\right)^{-1/4}. \qquad (15)$$

As can be seen from figure 8, this prediction is in good agreement with the experimental results, except for the very small N.

The relative cost of (H)ACS over (H)ITS given in equation (15) is significantly higher than previously reported. Both Ahmad and Cohen (1973) and Aarseth (1985) experimentally obtained the scaling of the calculation cost per timestep to be proportional to $N^{1.6}$, for N up to several hundred. In the limit $N \to \infty$, this scaling must

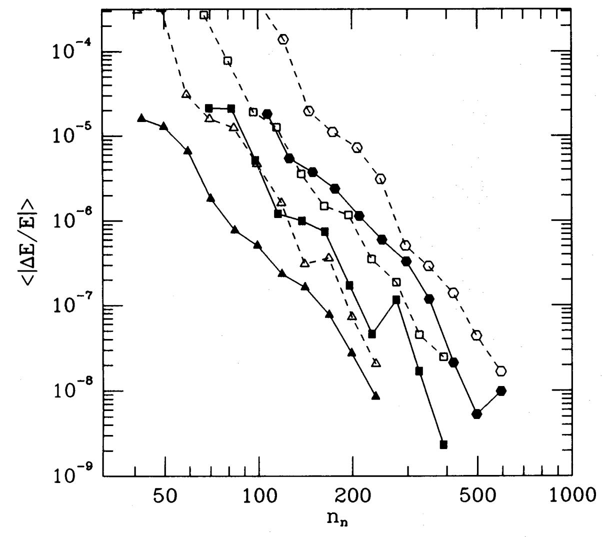


Fig. 7. Same as figure 2 but for Ahmad-Cohen schemes with an optimal parameter choice.

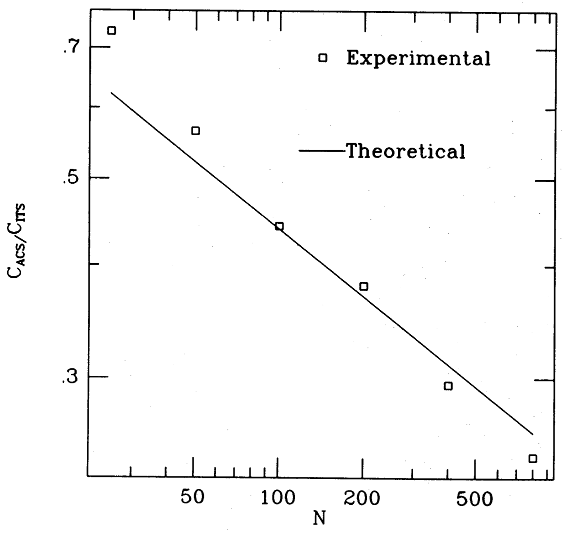


Fig. 8. Ratio between the calculation cost of (H)ACS and that of (H)ITS, plotted as a function of N for the optimal parameter choice for (H)ACS. The solid line represents the analytical prediction of equation (15).

break down, since the timestep for the distant step cannot exceed the crossing time. The difference between our result and the previous results is mainly because we measure only the force calculation cost, in units of the number of force calculations. For small N, the contribution of calculations other than the force calculation is relatively large and, therefore, the scaling of CPU time obtained for small N is different from the asymptotic behavior in the large-N limit. On a SPARC station 1+, the CPU time for ACS is 78% and 48% of those for ITS for $N = 50$ and $N = 200$, respectively. They are somewhat larger than the relative cost shown in figure 7. The actual CPU time to integrate the system for 1 crossing time is 2.80 minutes for $N = 200$, using ACS.

4. Efficiency on Actual Hardware

In section 3 we compared the efficiency of different integration schemes in terms of the number of force calculations per crossing time. In practice, however, the computing time on actual hardware is not determined solely by the number of force calculations. In this section, we first discuss the relative merit of Hermite schemes compared to standard schemes. We then discuss the gain that we can expect from the Ahmad-Cohen schemes on actual hardware.

4.1. Efficiency of Hermite Schemes

One force calculation of the Hermite scheme is more expensive than that of the standard scheme, simply because we must calculate $\dot{\boldsymbol{a}}$. The number of floating-point operations required to predict the position of one particle is 19, and that to calculate the acceleration is 16, excluding one division and one square-root operation. To predict the velocity, there are 12 more operations. To calculate $\dot{\boldsymbol{a}}$, 22 more operations are required. Therefore, depending on the relative speed of the square-root and the division, the calculation of one interaction of the Hermite scheme is, in the worst case, about a factor 2 more expensive than that of the standard scheme.

The actual difference in speed is strongly hardware dependent, simply because the relative speed of a square root or a division compared to that of an addition or a multiplication depends on the hardware. One extreme case is 68020- or 80386-based workstations or personal computers, on which a square root operation is only a few times slower than an addition. On these machines one step of the Hermite integrator would take twice as long. The other extreme is recent RISC-based workstations, on which it is not unusual that a square root operation is 20–50 times more expensive. On these machines the difference in the CPU time for one step is rather small.

The number of timesteps required to attain the same accuracy is about a factor 2 smaller for Hermite schemes. Therefore, even for the worst case, the Hermite scheme is only marginally slower than the standard schemes, and nearly a factor 2 faster for the best case.

We must take into account the cost of calculations other than the force calculation and the prediction. These calculations include the selection of the particle to update, the prediction of its position and velocity (ITS/ACS), the correction of the predicted position and velocity, determination of the next timestep, and correction of the higher order derivatives (ACS). For simplicity, we call this cost the cost of time integration.

The number of floating-point operations for time integration is about 130 per timestep for ITS, and about 90 for HITS, if we neglect the calculation cost of the search for the next particle to be advanced and the timestep determination. The number of timesteps needed to achieve the same accuracy is about a factor 2 smaller for HITS. Thus, there is a difference of about a factor of 3 for the calculation cost. With the Ahmad-Cohen scheme, this difference is even larger, since we do not need to correct the higher order derivatives at distant steps in the case of HACS.

On a scalar machine, the difference in the cost of the time integration does not change the total calculation cost, since the total CPU time is dominated by the force calculation. However, on vector/parallel machines or on a special-purpose system, such as GRAPE (Sugimoto et al. 1990), this difference actually changes the speed of the total calculation. For example, for $N = 1000$ and (H)ACS scheme, the average number of force calculations per timestep is about 200. The calculation cost of the time integration is roughly equal to that of 10 force calculations on a scalar computer. Therefore, if the speedup of the force calculation by vectorization or parallelization is a factor of 20, the time integration already accounts for 50% of the total CPU time, if this part is not vectorized at all. In principle, we can speed up the time integration by the hierarchical timestep scheme (McMillan 1986; Makino 1991b). However, the gain is rather limited since the vector length is not large.

4.2. Efficiency of Ahmad-Cohen Scheme

On some machines, the costs of the force calculation for (H)ITS and that for (H)ACS are quite different. For example, on vector supercomputers, such as a Cray Y-MP or a Fujitsu VP-2600, both the calculation of the distant force and that of the neighbor force are considerably slower than the total force calculation of (H)ITS. On vector machines, the calculation of the neighbor force is slow because (a) the vector length is short, and (b) it requires indirect addressing. The calculation of the distant force is slower because it requires one conditional statement and indirect addressing to construct the neighbor list.

To see the effect of these differences, let us assume (a) that the startup time for a vector operation is equivalent to the time to process N_s elements, (b) that the speed of the neighbor force calculation is slower than that of the force calculation of (H)ITS by a factor of α, and (c) that the speed of distant force calculation is slower by another factor β. Following Makino and Hut (1988), the calculation time per timestep is now expressed as

$$C_{\rm ACS} = k_1 \frac{\alpha(N+N_s)}{\Gamma(2/3)N_n^{1/3}} + k_1\beta(N_n+N_s). \qquad (16)$$

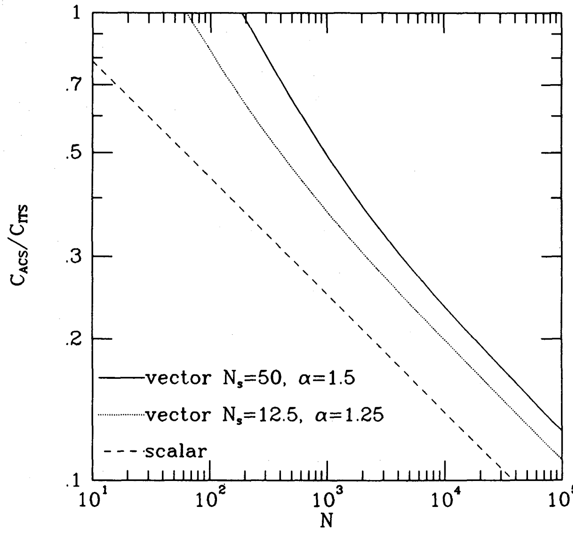


Fig. 9. Same as figure 8, but showing the theoretical gain for various vector processors.

By differentiating equation (16), we obtain the optimal N_n which minimizes the calculation cost,

$$N_{n,\rm opt} = \left[\frac{\alpha(N+N_s)}{3\Gamma(2/3)\beta}\right]^{3/4}. \qquad (17)$$

The calculation cost per timestep for this optimal N_n is expressed as

$$\begin{aligned} C_{\rm ACS} &= k_1[(3^{1/4}+3^{-3/4}) \\ &\quad \Gamma(2/3)^{-3/4}\alpha^{3/4}\beta^{1/4}(N+N_s)^{3/4}+\beta N_s] \\ &= 1.40k_1[\alpha^{3/4}\beta^{1/4}(N+N_s)^{3/4}+\beta N_s]. \qquad (18) \end{aligned}$$

The calculation cost of ITS is expressed as

$$C_{\rm ITS} = k_1(N+N_s). \qquad (19)$$

The ratio of calculation cost of ACS and that of ITS then becomes

$$\frac{C_{\rm ACS}}{C_{\rm ITS}} = 1.40\left[\frac{\alpha^{3/4}\beta^{1/4}}{(N+N_s)^{1/4}} + \frac{\beta N_s}{N+N_s}\right]. \qquad (20)$$

On vector processors with multiple pipelines such as a Fujitsu VP-2600, we expect $n_s \sim 50$, $\alpha \sim 2$, and $\beta \sim 1.5$. In this case, equation (20) becomes

$$\frac{C_{\rm ACS}}{C_{\rm ITS}} \simeq \frac{2.25}{(N+50)^{1/4}} + \frac{140}{N+50}. \qquad (21)$$

Figure 9 shows the speedup factor ($C_{\rm ACS}/C_{\rm ITS}$), as a function of the total number of particles (N) for the ideal case of $N_s = 0$, $\alpha = \beta = 1$ [equation (15)] and a realistic case of equation (21). For the realistic case, the crossover point is significantly higher than that for the ideal case; even for the limit $N \rightarrow \infty$, the speedup factor is smaller by a factor of 1.6. We also show an intermediate case of $N_s = 12.5$, $\alpha = 1.25$, and $\beta = 2$, which would represent a single-pipeline vector processor.

5. Conclusion

We have implemented HACS, a fourth-order Hermite integrator with the Ahmad-Cohen scheme. Its implementation is significantly simpler than the original ACS, since HACS is a self-starting scheme. Compared to ACS, HACS allows timesteps twice as long for the same accuracy, and the increase of the calculation cost per timestep is not very large. The actual gain in speed depends on the hardware, but will range between a factor 1 and 2. The gain by using the Ahmad-Cohen scheme is expressed as $(N/3.8)^{1/4}$ for both the fourth-order standard and Hermite schemes, but would be significantly smaller on vector or parallel machines.

We thank Steve McMillan for many stimulating discussions. J. M. thanks the Institute of Astronomy for its hospitality during his visit when this work was carried out.

References

Aarseth, S. J. 1963, *Monthly Notices Roy. Astron. Soc.*, **126**, 223.

Aarseth, S. J. 1985, in *Multiple Time Scales*, ed. J. U. Brackhill and B. I. Cohen (Academic Press, New York), p. 377.

Ahmad, A., and Cohen, L. 1973, *J. Comput. Phys.*, **12**, 389.

Makino, J. 1991a, *Astrophys. J.*, **369**, 200.

Makino, J. 1991b, *Publ. Astron. Soc. Japan.*, **43**, 859.

Makino, J., and Hut, P. 1988, *Astrophys. J. Suppl.*, **68**, 833.

McMillan, S. L. W. 1986, in *The Use of Supercomputers in Stellar Dynamics*, ed. P. Hut and S. L. W. McMillan (Springer-Verlag, Berlin), p. 156.

Sugimoto, D., Chikada, Y., Makino, J., Ito, T., Ebisuzaki, T., and Umemura, M. 1990, *Nature*, **345**, 33.