ACADEMIC PRESS

Available online at www.sciencedirect.com

SCIENCE DIRECT®

Journal of Computational Physics 185 (2003) 484–511

JOURNAL OF COMPUTATIONAL PHYSICS

www.elsevier.com/locate/jcp

Systolic and hyper-systolic algorithms for the gravitational N-body problem, with an application to Brownian motion

Ernst Nils Dorband, Marc Hemsendorf, David Merritt [*]

Department of Physics and Astronomy, Rutgers University, New Brunswick, NJ 08903, USA

Received 19 December 2001; received in revised form 2 October 2002; accepted 9 December 2002

Abstract

A systolic algorithm rhythmically computes and passes data through a network of processors. We investigate the performance of systolic algorithms for implementing the gravitational N-body problem on distributed-memory computers. Systolic algorithms minimize memory requirements by distributing the particles between processors. We show that the performance of systolic routines can be greatly enhanced by the use of non-blocking communication, which allows particle coordinates to be communicated at the same time that force calculations are being carried out. The performance enhancement is particularly great when block sizes are small, i.e., when only a small fraction of the N particles need their forces computed in each time step. Hyper-systolic algorithms reduce the communication complexity from $O(Np)$, with p the processor number, to $O(N\sqrt{p})$, at the expense of increased memory demands. We describe a hyper-systolic algorithm that will work with a block time step algorithm and analyze its performance. As an example of an application requiring large N, we use the systolic algorithm to carry out direct-summation simulations using 10^6 particles of the Brownian motion of the supermassive black hole at the center of the Milky Way galaxy.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Systolic algorithm; Gravitational N-body problem; Brownian motion

1. Introduction

Numerical algorithms for solving the gravitational N-body problem have evolved along one of two lines in recent years. Direct-summation codes compute the complete set of N^2 interparticle forces at each time step; these codes were designed for systems in which the finite-N graininess of the potential is important, and are limited by their $O(N^2)$ scaling to moderate ($N \lesssim 10^5$) particle numbers. The best-known examples are the NBODY series of codes introduced by Aarseth [1]. These codes typically use high-order schemes for integration of particle trajectories and avoid the force singularities at small interparticle separations either by softening, or by regularization of the equations of motion [2]. A second class of N-body algorithms replace the direct summation of forces from distant particles by an approximation scheme. Examples are

[*] Corresponding author: Fax: 1-732-445-4343.
E-mail address: merritt@physics.rutgers.edu (D. Merritt).

doi:10.1016/S0021-9991(02)00067-0

tree codes [3], which reduce the number of direct force calculations by collecting particles in boxes, and algorithms which represent the large-scale potential via a truncated basis-set expansion (e.g. [4]) or on a grid (e.g. [5]). These algorithms have a milder, $O(N \log N)$ scaling for the force calculations and can handle much larger particle numbers although at some expense in decreased accuracy [6].

The efficiency of both sorts of algorithm can be considerably increased by the use of individual time steps for advancing particle positions, since many astrophysically interesting systems exhibit a "core-halo" structure characterized by different regions with widely disparate force gradients. An extreme example of a core-halo system is a galaxy containing a central black hole [7]. The efficiency of individual time steps compared with a global time step has rendered such schemes standard elements of direct-summation codes (e.g. [8]).

Here we focus on direct-summation algorithms as implemented on multi-processor, distributed-memory machines. Applications for such codes include simulation of globular star clusters [9], galactic nuclei [10], or systems of planetesimals orbiting a star [11]. In such systems, the time scale over which the energy of a particle changes due to near-encounters with other particles is comparable to or shorter than the lifetime of the system. In most cases, values of N exceeding 10^5 would be desirable and it is natural to investigate parallel algorithms. There are two basic ways of implementing a parallel force computation for $O(N^2)$ problems for a computer having p processing elements.

1. *Replicated data algorithm.* All of the particle information is copied onto each processor at every time step. Computing node i, $1 \leqslant i \leqslant p$, computes the forces exerted by the entire set of N particles on the subset of $n_i = N/p$ particles assigned to it.
2. *Systolic algorithm.* At the start of each time step, each computing node contains only N/p particles. The sub-arrays are shifted sequentially to the other nodes where the partial forces are computed and stored. After $p-1$ such shifts, all of the force pairs have been computed.

(The term "systolic algorithm" was coined by H.T. Kung [12,13] by analogy with blood circulation.) Both types of algorithm exhibit an $O(Np)$ scaling in communication complexity and an $O(N^2)$ scaling in number of computations. The advantage of a systolic algorithm is its more efficient use of memory: since each processor stores only a fraction $1/p$ of the particles, the memory requirements are minimized and a larger number of particles can be integrated. Other advantages of systolic algorithms include elimination of global broadcasting, modular expansibility, and simple and regular data flows [13].

The performance of a systolic algorithm suffers, however, whenever the number of particles on which forces are being computed is less than the number of computing nodes. This is often the case in core-halo systems since only a fraction of the particles are advanced during a typical time step. As an extreme example, consider the use of a systolic algorithm to compute the total force on a *single* particle due to N other particles. Only one processor is active at a given time and assuming $p \gg 1$ then the total computing time for the blocking algorithm is

$$t_{\rm sb} = N\tau_{\rm f} + p(\tau_{\rm l} + \tau_{\rm c}), \tag{1}$$

where $\tau_{\rm f}$ is the time for one force calculation, $\tau_{\rm l}$ is the latency time required for two processors to establish a connection, and $\tau_{\rm c}$ is the interprocessor communication time for transmitting positions, velocities, and forces. Thus the algorithm is essentially linear and no advantage is gained from having multiple processors.

An efficient way to deal with the problem of small group sizes in systolic algorithms is via *nonblocking communication*, a feature of MPI that allows communication to be put in the background so that the computing nodes can send/receive data and calculate at the same time [14]. In a nonblocking algorithm, the time per force loop for a single particle $t_{\rm sn}$ becomes assuming $p \gg 1$

$$t_{\rm sn} = \frac{N\tau_{\rm f}}{p} + p(\tau_{\rm l} + \tau_{\rm c}). \tag{2}$$

The second term is the waiting time for the last computing node to receive the particle after p shifts. The first term is the time then required to compute the forces from the subset of N/p particles associated with the last node. As long as the calculation time is not dominated by interprocessor communication, the speedup is roughly a factor of p compared with the blocking algorithm.

Here we discuss the performance of systolic algorithms as applied to systems with small group sizes, i.e., systems in which the number of particles whose positions are advanced during a typical time step is a small fraction of the total. Section 2 presents the block time step scheme and its implementation as a systolic algorithm. Section 3 discusses the factors which affect the algorithm's performance, and Section 4 presents the results of performance tests on multi-processor machines of blocking and nonblocking algorithms. Section 5 presents a preliminary discussion of "hyper-systolic" algorithms with block time steps, which achieve an $O(N\sqrt{p})$ communication complexity at the cost of increased memory requirements. Finally, Section 6 describes an application of our systolic algorithm to a problem requiring the use of a large N: the gravitational Brownian motion of a supermassive black hole at the center of a galaxy.

2. Algorithm

In a direct-force code, the gravitational force acting on particle i is

$$\mathbf{F}_i = m_i\mathbf{a}_i = -Gm_i\sum_{k=1}^{N}\frac{m_k(\mathbf{r}_i-\mathbf{r}_k)}{|\mathbf{r}_i-\mathbf{r}_k|^3}, \tag{3}$$

where m_i is the mass of the ith particle and $\mathbf{r}_i$ its position; G is the gravitational force constant. The summation excludes $k = i$.

The integration of particle orbits is based on the fourth-order Hermite scheme as described by Makino and Aarseth [15]. We adopt their formula for computing the time step of an individual particle i at time t_{now},

$$\Delta t_i = \sqrt{\eta\frac{|\mathbf{a}(t_{\text{now}})||\mathbf{a}^{(2)}(t_{\text{now}})| + |\dot{\mathbf{a}}(t_{\text{now}})|^2}{|\dot{\mathbf{a}}(t_{\text{now}})||\mathbf{a}^{(3)}(t_{\text{now}})| + |\mathbf{a}^{(2)}(t_{\text{now}})|^2}}. \tag{4}$$

Here $\mathbf{a}$ is the acceleration of the ith particle, the superscript (j) denotes the jth time derivative, and η is a dimensionless constant of order 10^{-2}; we typically set $\eta = 0.02$. With a definition of individual time steps as in Eq. (4) the computing time for the integration of typical gravitational systems is significantly reduced in comparison with codes which do a full force calculation at each integration step [20]. Since the particle positions $\mathbf{r}_k$ must be up-to-date in Eq. (3) they are predicted using a low order polynomial. This prediction takes less time if groups of particles request a new force calculation at large time intervals, rather than if single particles request it in small time intervals. For this reason, an integer n is chosen such that

$$\left(\frac{1}{2}\right)^n \leqslant \Delta t_i < \left(\frac{1}{2}\right)^{n-1} \tag{5}$$

with Δt_i given by Eq. (4). The individual time step is replaced by a block time step $(\Delta t_i)_\text{b}$, where

$$(\Delta t_i)_\text{b} = \left(\frac{1}{2}\right)^n. \tag{6}$$

We implement the systolic force calculation in either blocking or nonblocking mode. For the non-blocking calculation, we have to add a buffer for storing the incoming positions and masses, one for the outgoing positions and masses, and one compute buffer. The maximal size of these buffers is defined by the largest particle number on one processing element. We also need input, output and compute buffers for the forces and the time-derivatives of the forces. Since data synchronization is not critical with blocking communication, extra input data buffering is not necessary. A positive side effect of the buffering strategy is that data access is far more ordered than in other implementations of the Hermite scheme. As a result, the number of cache misses is reduced, optimizing performance.

We arrange all processors in a ring-like structure, so that each processor has a right and a left neighbor. For the integrator to work, the individual block time step $(\Delta t_i)_{\mathrm{b}}$ and the time of the last integration $t_{\mathrm{last},i}$ must be defined for each particle. Either the initialization or the integration method are required to compute these two quantities.

Algorithm 1 (*Find new group*).

```
Search all particles $i$ for the smallest $t_{\min,p} = (\Delta t_{i,p})_{\mathrm{b}} + t_{\mathrm{last},i,p}$
  on each computing node $p$.
Do a global reduce so that each processor knows the global
  minimum $t_{\min} = \min(t_{\min,p})$. Set the simulated
  time to $t_{\mathrm{now}} = t_{\min}$.
Find the particles with $(\Delta t_{i,p})_{\mathrm{b}} + t_{\mathrm{last},i,p} = t_{\mathrm{now}}$
  and store their index $i$ in a list.
Predict the positions and velocities at the time
  $t_{\mathrm{now}}$ for the local particles on each node.
```

Each processor will select a subgroup s_p of the block size $s = \sum s_p$. The systolic shift with blocking communication is implemented as follows:

Algorithm 2 (*Force loop* (*blocking communication*)).

```
Copy the positions of the subgroups $s_p$ into the
   compute buffer on each node.
foreach $j$ in $s_p$
 Do the force calculation with respect to all local particles.
end foreach
Wait for all processors to finish their work.
Copy the masses, positions, and partial forces to the output
   buffer.
Send the output buffer to the right neighbor and store the
   data from the left neighbor in the compute buffer.
Do another local force calculation unless $p$ shifts have
   been done and the forces returned to their originating processor.
```

Utilizing nonblocking communication, the systolic loop can gain significant performance since it is possible to transfer data while the force calculation is ongoing. The partial forces follow one cycle behind the positions.

Algorithm 3 (*Force loop* (*nonblocking communication*)).

```
Copy the positions of the subgroups $s_p$ into the
   compute buffer and into the output buffer on each node.
if nloops between 1 and $p-1$:
 Initiate the data transfer for the positions and masses to
    the right node and allow data to be sent from the left neighbor
    to the input buffer.
end if
if nloops between 2 and $p$:
 Initiate the data transfer for the forces to the right node
     and allow data to be sent from the left neighbor to the input buffer.
end if
foreach $j$ in $s_p$
 Do the force calculation with respect to all local particles.
end foreach
if nloops between 1 and $p-1$:
 Wait for the data transfer of particles to finish.
end if
if nloops between 2 and $p$:
 Wait for the data transfer of the forces to finish.
 Add the partial forces to the incoming forces.
end if
Increment nloops and initiate another data transfer
    unless $p$ shifts have been made.
Shift the forces to the right neighbor.
```

Fig. 1 shows an idealized workgraph of the nonblocking systolic force computation, assumed to be completely calculation-dominated, which would be the case for a computer system with zero latency and infinite bandwidth. In the example shown in Fig. 1, processor one finds $s_1 = 30$, processor two $s_2 = 10$, processor three $s_3 = 40$, and processor four $s_4 = 20$ particles due for the force calculation. The thin vertical

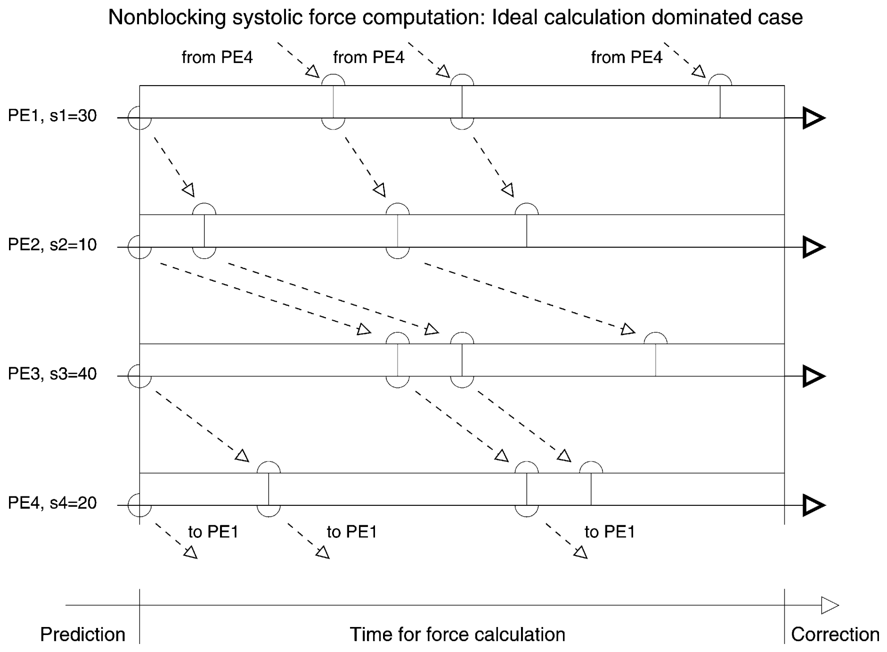

Fig. 1. Workflow diagram for an ideal, calculation-dominated nonblocking systolic force computation. Time increases to the right; the bold-faced arrows represent the work of each of the computing nodes, assumed here to be four. The dashed arrows indicate the flow of the position information between the nodes. The steepness of the arrows indicates how much time is allowed for the data transfer. Circles indicate the points when each processor switches from computing the partial forces from one subgroup to the next subgroup, including the time to finalize one communication thread and initialize the next one. Vertical lines represent barriers which can only be passed when all processes reach the same state of execution.

lines represent barriers which all processes can only pass together. The circles represent points in time at which a process changes from the computation of subgroup s_i to the next subgroup. Since all communication is in the background, switching between the subcalculations is very fast and the processing elements do not idle. The dashed arrows represent the data flow between processors; they begin at the sending point and terminate where the reception is finalized. The steepness indicates the maximal time allowed for data transfer between two nodes. In Fig. 1, communication between processing elements one and two must be carried out in a very short time as indicated by the steep inclination of the arrows. This is why, of all communications in the force loop, the transfer of s_3 between these two processors requires the highest bandwidth. In the ideal case, however, all processes finalize the force calculation at the same time as is shown by the second vertical bar. We discuss in the following section how closely we can reach this ideal in typical applications.

After computing the forces we finalize the integration in the same manner as described by Makino and Aarseth [15].

Algorithm 4 (*Hermite interpolation*).

```
    **Interpolate** higher force derivatives $\mathbf{F}_i^{(2)}(t_{\text{last},i})$ and $\mathbf{F}_i^{(3)}(t_{\text{last},i})$
        for all particles $i$ in group $s$ from $\mathbf{F}_i(t_{\text{last},i})$, $\dot{\mathbf{F}}_i(t_{\text{last},i})$, $\mathbf{F}_i(t_{\text{now}})$, and $\dot{\mathbf{F}}_i(t_{\text{now}})$.
    **Correct** the positions of the particles $i$ in group $s$ using
        a fourth-order Taylor polynomial.
    **Interpolate** higher force derivatives
    $\mathbf{F}_i^{(2)}(t_{\text{now}}) = \mathbf{F}_i^{(2)}(t_{\text{last},i}) + \mathbf{F}_i^{(3)}(t_{\text{last},i}) \times (\Delta t_i)_{\text{b}}$ and
    $\mathbf{F}_i^{(3)}(t_{\text{now}}) = \mathbf{F}_i^{(3)}(t_{\text{last},i})$.
    **Compute** the new time step according to Eqs. (4)–(6).
    **Set** $t_{\text{last},i} = t_{\text{now}}$.
```

We now have defined new block time steps $(\Delta t_i)_{\text{b}}$ and have set $t_{\text{last},i}$ for the particles in group s and can continue the integration by selecting a new group according to Algorithm 1.

3. Factors affecting the performance

3.1. General performance aspects

The performance of systolic force calculations is dependent on the calculation speed of each processing element and on the bandwidth of the inter-process communication network. Also important is the parallelism of the calculation and the load imbalance. However, predicting these latter two quantities requires precise knowledge of the distribution of work on the nodes and of numerous machine parameters. Some of these parameters might also be dependent on the overall usage of the parallel computer. For these reasons we restrict our estimates to an optimal and a worst-case scenario.

Let N be the total number of particles and s the number of particles whose coordinates are to be advanced during the current integration step. The particles to be integrated are distributed over p subgroups of size s_i, $1 \leqslant i \leqslant p$, such that the ith processor contributes s_i particles. Thus

$$\sum_{i=1}^{p} s_i = s, \tag{7}$$

$$\langle s_i \rangle = \frac{s}{p}. \tag{8}$$

Note that the s_i need not be equal, since the number of particles due to be integrated may be different on different processors. We assume that the time spent on force calculations is a linear function of the group size s_i, and similarly that the communication time is a linear function of the amount of transported data.

The time required for the force calculation can be estimated as follows. Let τ_f be the time required to do one pairwise force calculation. The total number of force calculations that one processor has to do per full force loop is $(N/p)s$, since the processor calculates the force of its (N/p) own particles against the s particles of the group. The total time for one full force loop in the calculation-dominated regime is therefore

$$t_\mathrm{f} = \frac{Ns}{p}\tau_\mathrm{f}. \tag{9}$$

The time necessary for communicating a subset of particles s_i from one processor to its neighboring one can be estimated as follows. Let τ_c be the time required to transfer the information for one particle to the next processor. The latency time τ_l is the time it takes to set up the communication between two processors. Within a single integration step, each processor has to establish p connections and has to send s particles. The total communication time is therefore:

$$t_\mathrm{c} = p\tau_\mathrm{l} + s\tau_\mathrm{c}. \tag{10}$$

3.2. Blocking communication

If the communication is not delegated to a communication thread, the force calculation phase for each subgroup s_i is followed by a communication phase. We first consider the case that all group sizes s_i are the same. Then the total time is just the sum of t_f and t_c, or

$$t = \frac{Ns}{p}\tau_\mathrm{f} + p\tau_\mathrm{l} + s\tau_\mathrm{c}. \tag{11}$$

The optimal processor number is obtained when $\mathrm{d}t/\mathrm{d}p = 0$, or

$$p_\mathrm{opt} = \sqrt{\frac{\tau_\mathrm{f}}{\tau_\mathrm{l}}Ns} \tag{12}$$

and

$$t_\mathrm{opt} = s\tau_\mathrm{c} + 2\sqrt{\tau_\mathrm{f}\tau_\mathrm{l}Ns}. \tag{13}$$

We now consider the case that the block sizes s_i are not equal. Each shift of particles must now wait for the processor with the largest block size s_max to complete its calculation and communication. So for each shift, the time t_s is

$$t_\mathrm{s} = \frac{N}{p}s_\mathrm{max}\tau_\mathrm{f} + \tau_\mathrm{l} + s_\mathrm{max}\tau_\mathrm{c} \tag{14}$$

and after p shifts,

$$t = Ns_\mathrm{max}\tau_\mathrm{f} + p\tau_\mathrm{l} + ps_\mathrm{max}\tau_\mathrm{c}. \tag{15}$$

Since $s = ps_i$ for equal s_i, Eq. (15) follows from Eq. (11) by substituting s_i with s_max. Defining

$$s_\mathrm{max} = \langle s_i \rangle + (\Delta s_i)_\mathrm{max} = \frac{s}{p} + (\Delta s_i)_\mathrm{max}, \tag{16}$$

we can write:

$$t = \frac{Ns}{p}\tau_\mathrm{f} + p\tau_\mathrm{l} + s\tau_\mathrm{c} + N(\Delta s_i)_{\max}\tau_\mathrm{f} + p(\Delta s_i)_{\max}\tau_\mathrm{c}. \tag{17}$$

Setting $\mathrm{d}t/\mathrm{d}p = 0$,

$$p_\mathrm{opt} = \sqrt{\frac{\tau_\mathrm{f} Ns}{\tau_\mathrm{l} + \tau_\mathrm{c}(\Delta s_i)_{\max}}}, \tag{18}$$

and the time is

$$t_\mathrm{opt} = 2\sqrt{Ns\tau_\mathrm{f}(\tau_\mathrm{l} + \tau_\mathrm{c}(\Delta s_i)_{\max})} + s\tau_\mathrm{c} + N(\Delta s_i)_{\max}\tau_\mathrm{f}. \tag{19}$$

3.3. Nonblocking communication

On a system which allows concurrent communication and calculation, the systolic algorithm can become more efficient, since the effective cost of communication is reduced. In this case the communication routines return immediately after initializing the communication process. As described in more detail in Section 2, each processor has to perform the following steps p times for each integration step:

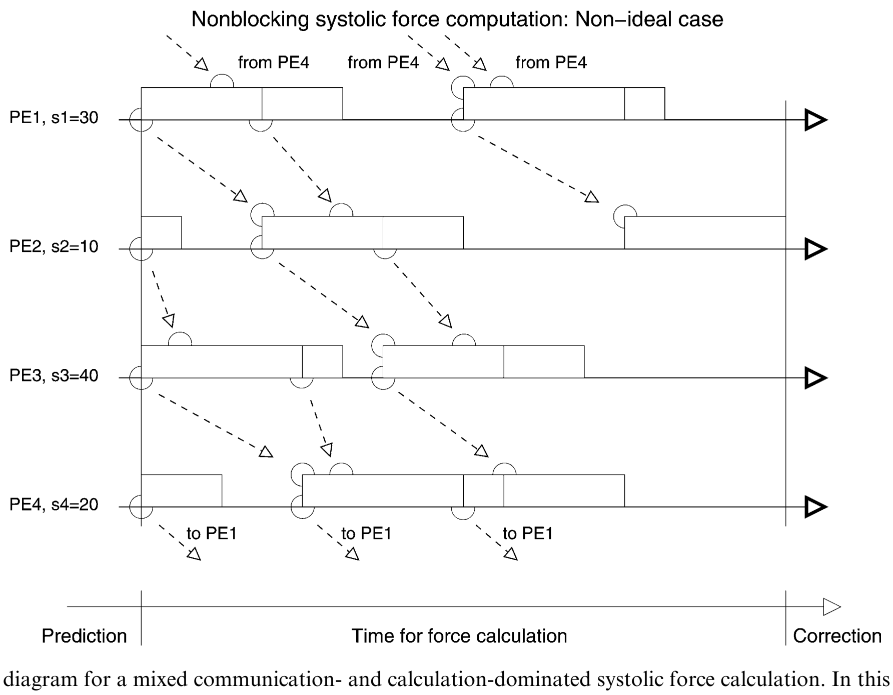

Fig. 2. Workflow diagram for a mixed communication- and calculation-dominated systolic force calculation. In this example we have chosen the communication time to be a linear function of the calculation time with $t_{\mathrm{c},i} = t_{\mathrm{f},i}$ and $\tau_\mathrm{l} = 0$. The hatched blocks symbolize times when the processors perform the force calculations. The circles at the lower ends of these blocks denote times when the communication of positional data is initiated. The ones at the upper end denote the times when the communication has completed. The dashed arrows show the dataflow. Their steepness indicates the time needed for each communication with constant bandwidth. In order to simplify this graph, the communication of partial forces is omitted.

(1) Do the force calculation for the s_i particles of one subgroup.
(2) Simultaneously send the s_i particles to the next processor.
(3) Simultaneously receive the s_{i-1} particles from the previous processor.

Each of these steps starts at the same time, but they might take different amounts of time to finish. Since (2) and (3) behave in a quite similar way, we treat them together and call them *communication*. Step (1) is called *calculation*. A system in which the calculation takes more time than the communication is called a *calculation-dominated system*, and a system in which communication is dominant we call a *communication-dominated system*. Our goal is to derive approximate expressions for the total time, calculation plus communication, per integration step and to minimize this time.

Fig. 2 shows a communication-dominated system. In this example, $\tau_l = 0$ and $\tau_c = \tau_f$. With this communication speed and the very uneven distribution of subgroups s_i on the processors, it is not possible to ensure a continuous force calculation. The time to compute the forces is dominated by the communication of the largest group (s_3 in Fig. 2). With this scenario we are able to define a threshold for the communication speed, which ensures that processors are not waiting for the communication to finish. Let $t_{c,\max}$ be the processor-to-processor communication time for the largest subgroup $s_{\max}$ and $t_{f,\min}$ the force calculation time for the smallest subgroup $s_{\min}$. Continuous calculation can be expected when

$$t_{c,\max} \leqslant t_{f,\min}. \tag{20}$$

The values for the latency time and the throughput are dependent on the specific MPI implementation. Cray T3Es typically have a two-mode implementation with algorithms for small and for large messages. On the T3E-900 in Stuttgart, the sustained average latency and throughput has been measured to be $\tau_l = 6\,\mu s$ for the routines `MPI_Send` and `MPI_Recv` [16]. The MPI implementation on this machine actually has a four-mode scheme which provides the following measured bandwidths β:

$$\beta_1 \geqslant 220\,\text{Mbyte/s} \quad \text{for } L_m \geqslant 8\,\text{kbyte},$$

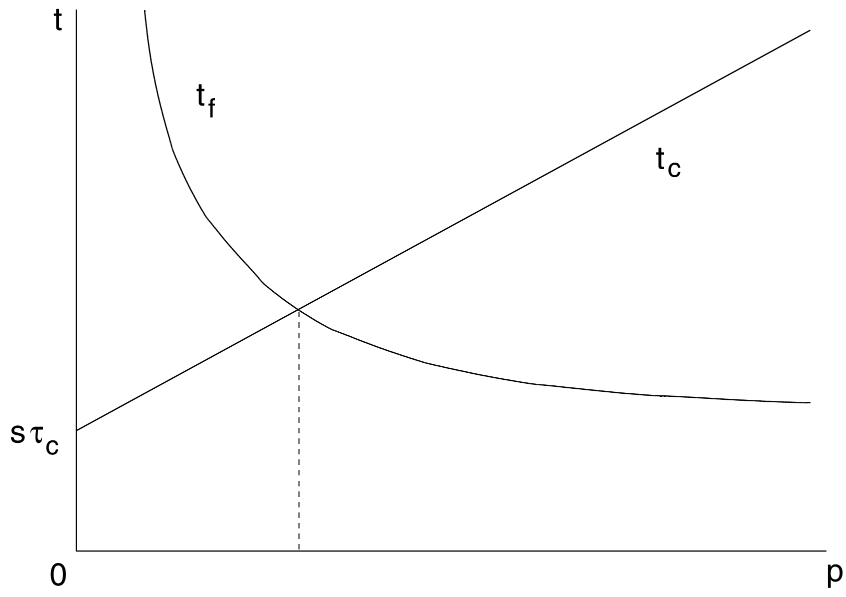

Fig. 3. Communication and calculation time as a function of processor number p for fixed N as derived from Eqs. (9) and (10). This figure is only valid under the assumption of equal s_i.

$$\beta_2 \geqslant 300\,\text{Mbyte/s} \quad \text{for } L_\text{m} \geqslant 64\,\text{kbyte},$$

$$\beta_3 \geqslant 315\,\text{Mbyte/s} \quad \text{for } L_\text{m} \geqslant 256\,\text{kbyte}.$$

The quantity L_m denotes the message length. These figures show that the actual throughput is dependent on the machine architecture, load and MPI implementation [16].

Assuming all s_i to be equal, a rough estimate for an optimal processor number can be given. Fig. 3 shows that there is an optimum value $p = p_\text{opt}$ which minimizes the total time in this case. We find p_opt by setting (9) equal to (10):

$$p_\text{opt} = \frac{-s\tau_\text{c} + \sqrt{s^2\tau_\text{c}^2 + 4\tau_\text{l}\tau_\text{f}Ns}}{2\tau_\text{l}} \tag{21}$$

and the total time is

$$t_\text{opt} = \frac{s\tau_\text{c}}{2} + \frac{1}{2}\sqrt{s^2\tau_\text{c}^2 + 4\tau_\text{l}\tau_\text{f}Ns}. \tag{22}$$

Eqs. (22) and (21) become far simpler on machines having zero latency. With $\tau_\text{l} = 0$ we find

$$p_\text{opt} = N\frac{\tau_\text{f}}{\tau_\text{c}}, \tag{23}$$

$$t_\text{opt} = s\tau_\text{c}. \tag{24}$$

This means that the systolic force loop parallelizes extremely well on such an idealized computer so that the computing time is only dominated by the performance of the intercommunication network. However this equation also allows the use of more processors than particles if the communication is faster than the force calculation. We give a more detailed picture of the situation in the following.

In N-body systems with a low central concentration, i.e. small core density, s is usually proportional to $N^{2/3}$ [15]. In our benchmarks discussed below, however, we find power-law indices of 0.55 ± 0.05 for the Plummer model, 0.437 ± 0.105 for the Dehnen model, and 0.492 ± 0.110 for the Dehnen model with a black hole. For this reason, we estimate the group sizes to be $s \approx \xi N^{1/2}$. The constant ξ can be derived for each type of dataset from the measured values for s in Table 1.

In our scheme, a given processor may be calculation- or communication-dominated depending on the value of s_i that is currently assigned to it. We can compute the time per integration loop in this more general case by again focussing on just one processor, which carries out p operations of force calculation and communication per loop. Each of these p operations will be either calculation- or communication-dominated, requiring either a time of $t_{\text{f},i}$ or $t_{\text{c},i}$. With $1 \leqslant i \leqslant p$,

$$t_{\text{f},i} = \frac{N}{p}s_i\tau_\text{f}, \tag{25}$$

$$t_{\text{c},i} = \tau_\text{l} + s_i\tau_\text{c}, \tag{26}$$

Table 1
Mean group sizes for the benchmarks

Model	131 072	65 536	32 768	16 384
Plummer	172	128	89	54
Dehnen, $M_\bullet = 0$	13.1	11.2	9.2	5.1
Dehnen, $M_\bullet = 0.01$	6.7	4.1	2.6	2.5

With the notion of the threshold in Eq. (20), which guarantees a continuous calculation phase, we have a completely calculation-dominated system. Thus

$$t_{\text{calc}} = \sum_{i=1}^{p} t_{\text{f},i}, \tag{27}$$

$$= \frac{N}{p} s\tau_{\text{f}}, \tag{28}$$

where t_{calc} is the computing time for calculation-dominated force loops. It is not trivial, though, to define an optimal processor number, since the distribution of the s_i and the size of s is a dynamical quantity of the integrated particle set. The result from Eq. (28) describes the ideal scaling for the force calculation which we use below as a basis of comparison with the benchmark data.

If the threshold (20) is not fulfilled, a significant share of the computation time t comes from the communication. Assuming a worst-case scenario, where all particles of the group s are found on only one processor, the computation time becomes maximal:

$$t_{\text{comm}} = t_{\text{f,max}} + \sum_{i=1}^{p} t_{\text{c,max}}, \tag{29}$$

$$= t_{\text{f,max}} + p t_{\text{c,max}}, \tag{30}$$

where t_{comm} is the computing time for communication-dominated force loops. The overall calculation time for the situation shown in Fig. 2 follows Eq. (30). We use this equation below to estimate the minimal scaling behavior expected from our code.

3.4. Comparing the performance of the two methods

In comparison with the nonblocking algorithm, the blocking scheme does not fall behind in terms of the parallelism. However, as long as the force computations remain calculation-dominated in the nonblocking scheme, minor variations between the s_i can be levelled out so that perfect load balancing is guaranteed. The blocking scheme is not flexible in this regard so that it builds up a penalty of the order $N(\Delta s_i)_{\text{max}}\tau_{\text{f}}$ per force loop. Since Eq. (17) shows that there are a few particles in the blocking scheme which are treated with a serial performance, the overall parallel efficiency is reduced. This means that a systolic loop will only give satisfying performance when load balancing is guaranteed either on a per processor level by keeping all s_i the same, or by applying nonblocking communication. This is why a nonblocking communication scheme is superior to the blocking one.

Depending on the type of system integrated, the number of available processors, the performance of the computer, and the total number of particles, the work load for the processors might become very small. Small group sizes are less likely to be distributed evenly on the processes; we expect an approximately Poisson distribution, or

$$(\Delta s_i)_{\text{max}} \approx \sqrt{s_i}. \tag{31}$$

For the systolic algorithm, large problem sizes, together with group sizes that are larger in the mean than the number of available processors, ensure efficient parallel performance with a systolic algorithm. In this case one can expect an ideal linear scaling of the performance with processor number. For small problem sizes, the amount of communication governs the the performance.

3.5. More elaborate codes

Parallelizing an individual block time step scheme is not trivial, as the discussion above has shown. However, is has also been shown that a fairly simple scheme can profit enormously from threaded, and therefore nonblocking, communication. Since the overall communication time is a linear function of the processor number in our simple scheme, attempts have been made to reduce the number of communications. The hyper-systolic codes proposed by Lippert et al. [17,18] and the broadcast method proposed by Makino [19] can perform the force calculation by using only $\sqrt{p}$ communication events. However, both methods require identical group sizes on each computing node in order to have efficient load balancing.

Individual block time step schemes select their particles to be integrated under physical criteria defined by the simulated system. This is why a load imbalance is unavoidable in general. Assuming there is no extra algorithm that provides a perfect distribution of the group, the expected imbalance time would be dependent on the particle distribution statistics in Eq. (31). Thus

$$t_{\mathrm{imb}} = \frac{N(\Delta s_i)_{\max}}{\sqrt{p}}\tau_{\mathrm{f}}. \tag{32}$$

This implies that the nonblocking systolic code will outperform nonbalanced hyper-systolic or broadcast methods unless the processor number is very large.

4. Measured performance

We evaluated the performance of the algorithms in realistic applications by using them to evolve particle models of spherical stellar systems. The evolution was carried out for approximately one crossing time, and the benchmarks were based on timings for 2000 integration steps. The Hermite integration scheme adjusts the group sizes automatically as described above. All experiments were carried out on the Cray T3E at the Goddard Space Flight Center. We compiled the executable from our C sources using the Cray standard C compiler version 6.2.0.0 with no explicit optimization.

4.1. Initial conditions

We consider three models representing spherical stellar systems in equilibrium. The Plummer model [21] has mass density and gravitational potential

$$\rho(r) = \frac{3GM}{4\pi}\frac{b^2}{(r^2+b^2)^{5/2}}, \tag{33}$$

$$\Phi(r) = -\frac{GM}{\sqrt{r^2+b^2}}. \tag{34}$$

Here M is the total mass and b is a scale length. The many analytic properties of this model make it a common test case for benchmarking. For the present application, the most important feature of the Plummer model is its low degree of central concentration and its finite central density, similar to the density profiles of globular star clusters. The Dehnen family of models [22] are characterized by a parameter γ that measures the degree of central concentration. The density profile is

$$\rho(r) = \frac{(3-\gamma)M}{4\pi}\frac{a}{r^\gamma(r+a)^{4-\gamma}} \tag{35}$$

with M the total mass as before and a the scale length. We chose a centrally condensed Dehnen model with $\gamma = 2$, yielding a density profile similar to those of elliptical galaxies with dense stellar nuclei. The gravitational potential for $\gamma = 2$ is

$$\Phi(r) = -\frac{GM}{a} \ln\left(\frac{r}{r+a}\right). \tag{36}$$

The central density diverges as r^{-2} and the gravitational force diverges as r^{-1}. Our third model had a density equal to that of (35) with $\gamma = 2$ and an additional mass component consisting of a point particle at the center representing a supermassive black hole. The mass $M_\bullet$ of the "black hole" was 1% of the stellar mass of the model. This is similar to the black-hole-to-galaxy mass ratios observed in real galaxies [23].

The initial particle velocities for the N-body realizations of our models were selected from the unique, isotropic velocity distribution functions that reproduce $\rho(r)$ in the corresponding potentials $\Phi(r)$. For all of the models we adopted units in which the gravitational constant G and the total mass of the the system M were set to unity. The scale length a of the Dehnen model was also set to unity, while the Plummer radius b was chosen to be $3\pi/16$. We computed two realizations for each particle number N. The total number of stars was $N = (16\,384, 32\,768, 65\,536, 131\,072)$.

After the integration over 2000 steps we computed the average group size during the benchmark. Table 1 summarizes the results. The centrally condensed Dehnen model systems require the use of far smaller groups than the Plummer model. This poses a severe challenge to systolic algorithms.

4.2. Performance of the force calculation

Since the complexity of the force calculation is $O(N^2)$, compared to $O(N)$ for the integrator, it is critical to measure the impact of parallelism on this task. We present comparisons of blocking and nonblocking systolic codes on the three datasets described above. We count 70 floating point operations per pairwise force calculation in our implementation.

The particle blocks may have very different sizes, and accordingly the efficiency will vary from force step to force step. For this reason we measure performance by summing the time required to carry out all force computations in a time interval corresponding to 2000 steps, then averaging.

Figs. 4–6 show the measured speedups, expressed in terms of the number of force calculations per second. In the case of the nonblocking algorithm, the expectation from Eq. (28), assuming a calculation-dominated system, is

$$t_{\mathrm{calc}} \propto \frac{1}{p}, \tag{37}$$

corresponding to a linear trend in Figs. 4–6. This dependence is recovered for each of the three models when N is large. In the Dehnen-model runs with small N, the force calculation becomes communication-dominated due to the small group sizes (Table 1) and the performance is described by Eq. (30):

$$t_{\mathrm{comm}} \propto p \tag{38}$$

corresponding to a $1/p$ dependence of the speedup in Figs. 4–6. This effect is less pronounced for the Plummer models since the mean group size is larger (Table 1). The effect is strongest for the Dehnen models with "black hole" since they have the smallest group sizes (Table 1).

In comparison with the runs involving Dehnen models, the Plummer model runs show a peculiar behavior. For very low processor number, the performance increases linearly. However for an intermediate number of computing nodes, the performance shows a *super*-linear gain until the curve becomes shallow for

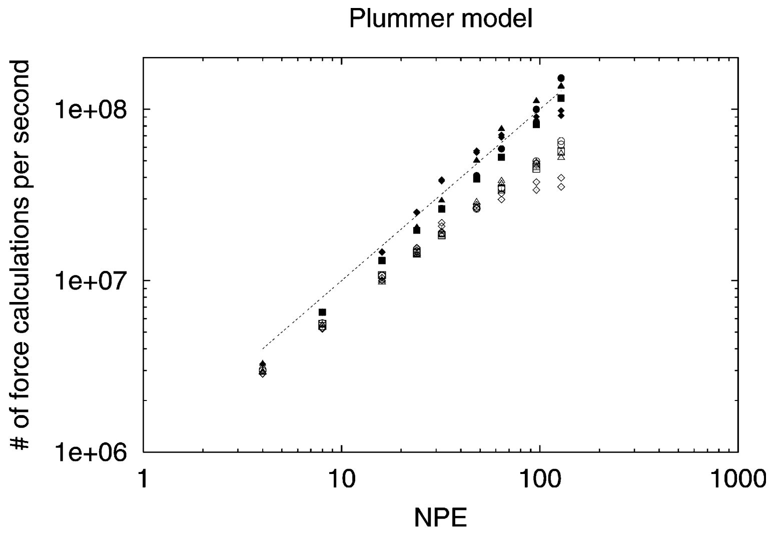

Fig. 4. Results of the benchmarks for the Plummer model initial conditions. The number of pairwise force calculations per second has been plotted as a function of processor number *NPE*. Non-blocking and blocking algorithms are indicated via filled and open symbols respectively, for the four different particle numbers N listed in Table 1. The dashed line shows the expected performance in the case of an ideal, calculation-dominated system (linear speedup).

large processor numbers. This super-linearity is a result of the way in which message passing is implemented on the T3E, which switches to a different protocol for small messages. The mean message size is inversely proportional to the total number of processes so that we can profit from this change. As Table 1 shows, the mean group size (which is proportional to the mean message size) is significantly smaller for the Dehnen models explaining why the super-linearity does not occur.

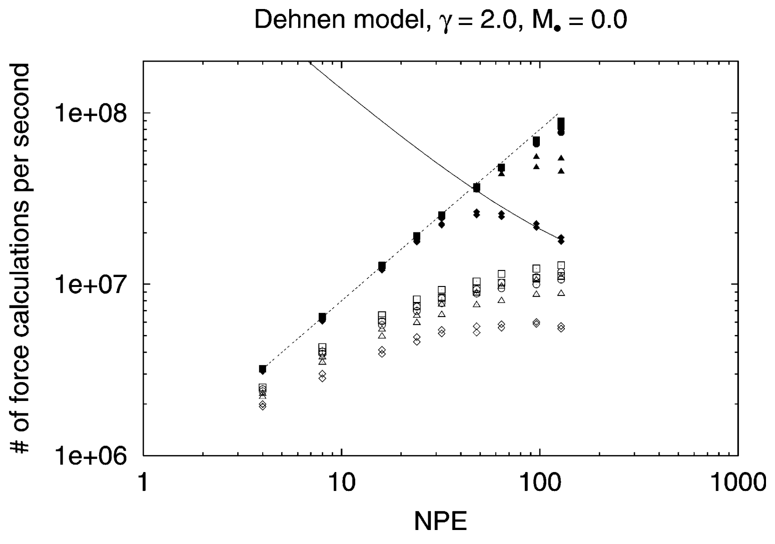

Fig. 5. Same as Fig. 4, for the Dehnen model initial conditions. The dashed and solid lines show the expected scaling for calculation- and communication-dominated systems. The performance of the non-blocking algorithm is reduced for small N due to the small group sizes.

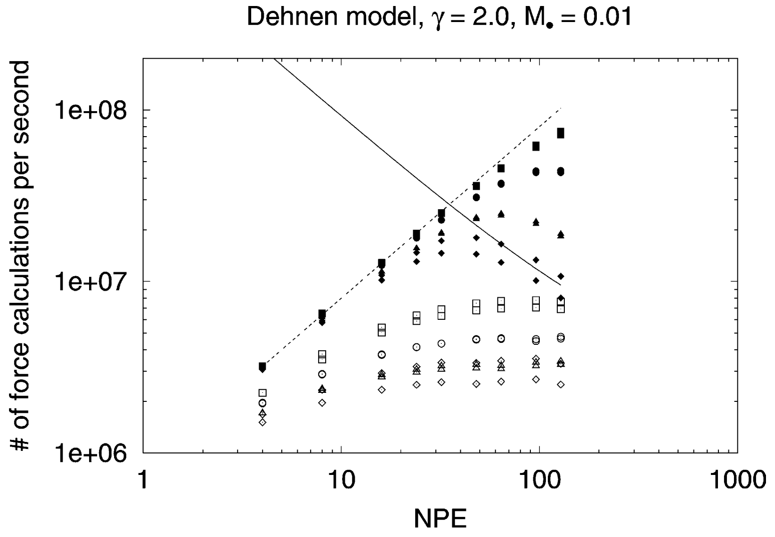

Fig. 6. Same as Fig. 5, for the Dehnen model with a central point particle ("black hole"). The influence of small group sizes is even more striking.

The open symbols in Figs. 4–6 show the results for the blocking systolic algorithm. In the Plummer model runs, the blocking code managed to calculate a little more than half the number of force pairs per second compared with the nonblocking code. The performance difference is much more dramatic in the Dehnen model runs due to the small group sizes: the blocking code can be as much as a factor of 10 slower than the non-blocking code. The computing time is asymptotically constant in the Dehnen model runs since the force calculation becomes increasingly serial for small s, as given by Eq. (17). For very large N we expect the $1/p$ scaling to take over. This does not appear in the plots because of the inefficiency of the blocking force calculation for small group sizes.

4.3. Performance of the integration

The performance of the N-body code depends also on the efficiency of the algorithm that advances the particle positions, which we call the "integrator". Here we give benchmarks for the full code including the integrator. Our performance goal is to reach a linear increase of computer wall-clock time as a function of problem size for a constant simulation interval and optimal processor number. The unit of time in the simulations is fixed by the gravitational constant (which is set to $G = 1$), the total mass, and the adopted length scale via $[T] = (GM/R^3)^{-1/2}$ [24]. Our performance results are based on an integration using 2000 force loops. The integrated time in model units is summarized in Table 2. We carried out two integrations for each model based on different random realizations of the initial conditions.

The performance of the blocking scheme is shown in Fig. 7, for the same values of N and NPE shown in Figs. 4–6. We now plot performance as a function of N; runs with the same N but different NPE are plotted with the same symbol and their vertical scatter is an indication of how well the code benefits from parallelism. In the blocking scheme, the points are very localized vertically revealing that the code can not profit much from large processor numbers. Larger problem sizes, however, are able to optimize the computations in such a way that the scale-up is better than quadratic, the scaling in traditional direct-force codes. With the individual time step scheme described here we expect a behavior of Ns. Since we observe an $N^{1/2}$ behavior for the mean group size, we can expect a scale-up following a $N^{3/2}$ power law in our benchmarks. We compare the maximal performance with increasing work load in Figs. 7 and 8 for our code

Table 2
Integrated time after 2000 force loops for the datasets used in the benchmarks

Dataset	1	2
Plummer		
16 384	4.57×10^{-2}	5.40×10^{-2}
32 768	3.35×10^{-2}	3.08×10^{-2}
65 536	1.90×10^{-2}	2.09×10^{-2}
131 072	1.08×10^{-2}	9.95×10^{-3}
$\gamma = 2.0, M_\bullet = 0.0$		
16 385	1.75×10^{-3}	1.59×10^{-3}
32 769	1.08×10^{-3}	1.35×10^{-3}
65 537	5.35×10^{-4}	6.68×10^{-4}
131 073	4.11×10^{-4}	3.26×10^{-4}
$\gamma = 2.0, M_\bullet = 0.01$		
16 385	3.99×10^{-4}	2.63×10^{-4}
32 769	1.53×10^{-4}	1.49×10^{-4}
65 537	1.30×10^{-4}	1.36×10^{-4}
131 073	4.52×10^{-5}	8.51×10^{-5}

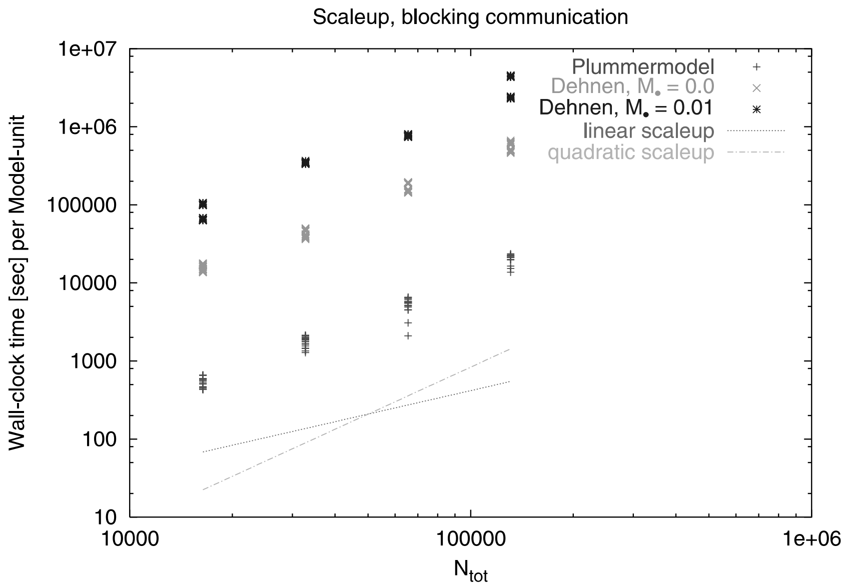

Fig. 7. Wall clock time in seconds per integrated model unit as a function of particle number N for the blocking N-body code. Different points at given N show the performance for the various values of NPE of Figs. 4–6. For comparison we plot lines following a linear and a quadratic N-dependence. The lines are positioned in such a way that they mark the maximal speed of the nonblocking code.

which shows a scale up between $O(N^2)$ and $O(N^{3/2})$ on a serial machine. In our benchmarks we consider any scaling better than that as indicating a benefit from parallel computing.

In order to quantify the scale-up, we fit the two, best-performing runs for each data set. We fit for a and b in the following function:

$$\frac{t_{\text{wc}}}{T} = \frac{N_{\text{tot}}^b}{a}. \tag{39}$$

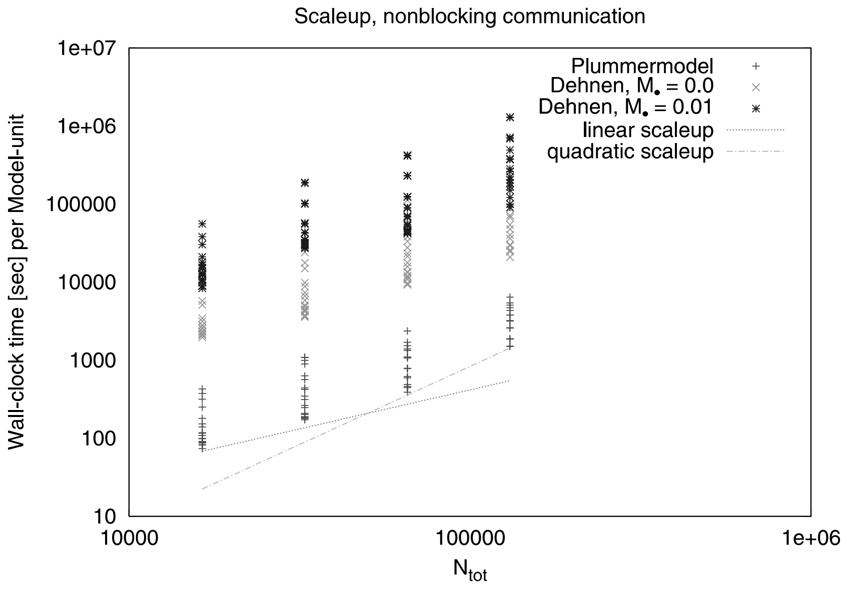

Fig. 8. Same as Fig. 7 for the code with nonblocking communication.

The quantity T the integrated time in model units and t_{wc} is the wall clock time for the run. Table 3 summarizes the results for b. Since the benchmarks using the Plummer model with the nonblocking code did not reach their maximum speedup for $N = 131\,072$ and $65\,536$, we estimated the scale-up from the runs with smaller particle sets only. These values are italicized in Table 3.

When measuring the overall performance of the integrator, the nonblocking communication scheme makes much better use of the T3E than the blocking scheme: the broad vertical bands in Fig. 8 show that the integration speed profits from increasing processor number. The two lines representing a linear and a quadratic behavior are placed at the optimal speed levels for the runs integrating a Plummer model. For all three types of model, a nearly linear scale-up can be observed. The exceptions are the datasets representing a Plummer model with 131 072 and 65 536 particles. The reason for this is no bottleneck. In fact, the nonblocking code scales so well that we could not reach the optimal performance with our maximal number of 128 processing elements in the benchmarks.

4.4. Nonblocking vs. blocking communication

While the implementation of a systolic force calculation is simpler and more memory-efficient using blocking communication, we have found significant performance gains using a nonblocking algorithm. The results of our force calculation benchmarks, displayed in Figs. 4–6, show up to 10 times better performance

Table 3
Power-law fits of the two best performing runs in each benchmark for the blocking and nonblocking communication runs

Dataset	Blocking		Nonblocking	
	b	Δb	b	Δb
Plummer	1.65	0.03	*1.174*	*0.112*
$\gamma = 2.0, M_\bullet = 0.0$	1.702	0.006	1.16	0.03
$\gamma = 2.0, M_\bullet = 0.01$	1.612	0.003	0.9	0.1

for a code implementing a nonblocking systolic force calculation over one which applies a blocking scheme. The gain is greatest when the typical particle group size is smallest. The integration as a whole can profit more from the parallelization in the nonblocking scheme than in the blocking: we observe a scale-up close to $N^{3/2}$ for the blocking scheme, while the nonblocking scheme clearly reaches a linear scaling. This means by choosing an appropriate number of processors for a specific problem size, we can effectively reduce the complexity of the direct force integration to $\mathrm{O}(N)$.

5. Hyper-systolic force calculation with individual block time steps

5.1. Systolic and hyper-systolic matrices

Introduced by Lippert et al. [17,18], the hyper-systolic algorithm reduces communication costs in $\mathrm{O}(N^2)$ problems by improving the communication complexity from $\mathrm{O}(Np)$ (systolic algorithm) to $\mathrm{O}(N\sqrt{p})$. The price paid is an increased need for memory, of order $\mathrm{O}(\sqrt{p})$ [18].

The standard systolic algorithm is performed on a one-dimensional ring of p processors. In order to transfer all positions and forces, p shifts of all N particles are required for each force calculation. Thus the communication complexity is $\mathrm{O}(Np)$.

By analogy with the systolic case, we define the vector of input data $\mathbf{x} \equiv (x_1, x_2, \ldots, x_p)$. In this definition the x_i are all data initially stored on processor p_i. These data are copied to a second vector $\tilde{\mathbf{x}}$ that is shifted between the processors. (Note our assumption here that the blocksize is N/p, i.e., that every particle needs its forces updated. We relax this assumption below.) To calculate the total forces one has to take into account all possible pairs of x_i with $\tilde{x}_i$. To visualize the systolic algorithm, we write down a matrix that shows the intermediate states of $\tilde{\mathbf{x}}$ at each step of the force calculation. This matrix is called the "systolic matrix" S. The first line shows $\tilde{\mathbf{x}}$ at shift zero, the ith line shows $\tilde{\mathbf{x}}$ at shift i:

$$S = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 8 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 8 & 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 7 & 8 & 1 & 2 & 3 & 4 & 5 \\ 5 & 6 & 7 & 8 & 1 & 2 & 3 & 4 \\ 4 & 5 & 6 & 7 & 8 & 1 & 2 & 3 \\ 3 & 4 & 5 & 6 & 7 & 8 & 1 & 2 \\ 2 & 3 & 4 & 5 & 6 & 7 & 8 & 1 \end{pmatrix}. \tag{40}$$

Local computations are done within one column of this matrix. One notices that there is some redundancy of vertical pairs: for instance, the pair $(1,2)$ occurs 8 times. As a result of this redundancy it is not necessary to perform all of the shifts in order to get every possible pair of elements of $\mathbf{x}$. In fact it would be sufficient to perform just three shifts, so that we get a matrix with just four rows. This matrix is called the "hyper-systolic matrix" H and is

$$H = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 8 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 8 & 1 & 2 & 3 & 4 & 5 & 6 \\ 5 & 6 & 7 & 8 & 1 & 2 & 3 & 4 \end{pmatrix}. \tag{41}$$

In contrast to the matrix S, which allows us to get all pairs by comparing the first row with one of the other rows, not all of the pairs in H have one member in the first row; for example, the pair (x_1, x_4) requires comparing rows 2 and 4. Therefore all of the shifted data in the hyper-systolic algorithm must be stored on each node in order to compute the forces. This is the reason for the increased memory requirements in comparison with the systolic algorithm.

The hyper-systolic matrix H can be characterized by a vector called the hyper-systolic base $A_k = (0, a_1, \ldots, a_k)$, where a_i gives the stride of the ith shift and k is the number of shifts that have to be done. In our example of H for $p = 8$, A_3 would be:

$$A_3 = (0, 1, 1, 2). \tag{42}$$

To minimize the communication costs, one wants to minimize k, the number of shifts. This is a nontrivial problem and optimization techniques have been described for achieving this aim [25].

In the following we show that the complexity of communication is $\mathrm{O}(n\sqrt{p})$ [17]. The minimum number of pairings required for the total force calculation is $p(p-1)/2$. The number of possible combinations within a column of $k+1$ elements is $\binom{k+1}{2}$, thus:

$$\frac{p(p-1)}{2} \leqslant \binom{k+1}{2} p = \frac{(k+1)k}{2} p. \tag{43}$$

The solution for this inequality for $k \geqslant 0$ is

$$k \geqslant \sqrt{p - \frac{3}{4}} - \frac{1}{2}, \tag{44}$$

which we wanted to prove. However, since we do $\mathrm{O}(\sqrt{p})$ shifts and we have to save the intermediate data, we need $\mathrm{O}(\sqrt{p})$ more memory on each processor than with the standard systolic algorithm.

5.2. Hyper-systolic matrix with block time steps

The hyper-systolic algorithm described by Lippert et al. [17,18] assumes that all $N(N-1)/2$ force pairs are to be computed at each time step. When dealing with block time steps, however, only a subset of the full N particles are shifted at each time step, and a minimal basis like that of Eq. (41) is not sufficient to compute the required forces. This is because the full data are only stored in the first row and data pairs constructed using other rows only contain information about the block particles.

Nevertheless one can construct a different hyper-systolic matrix $\tilde{H}$ that ensures complete force calculations. For $p = 8$ a possible matrix would be:

$$\tilde{H} = \begin{pmatrix} \mathbf{1} & \mathbf{2} & \mathbf{3} & \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} \\ 8 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 7 & 8 & 1 & 2 & 3 & 4 & 5 & 6 \\ 6 & 7 & 8 & 1 & 2 & 3 & 4 & 5 \\ 5 & 6 & 7 & 8 & 1 & 2 & 3 & 4 \\ \mathbf{4} & \mathbf{5} & \mathbf{6} & \mathbf{7} & \mathbf{8} & \mathbf{1} & \mathbf{2} & \mathbf{3} \end{pmatrix}. \tag{45}$$

The bold figures represent full data sets and the non-bold figures indicate the block data sets. With this matrix of shifts, one can calculate the complete forces. For example, for the first block data set, one uses columns 2, 3, 4, and 5. The block data can be overwritten as in the systolic algorithm. The number of shifts needed for this algorithm is 6. First the block data have to be shifted (4 shifts); then they have to be shifted back to their home processor to add the forces (1 shift); and finally the full data sets have to be updated (1 shift). With the standard systolic algorithm one would need 8 shifts. We pay for this advantage with a higher amount of memory required: one more full data set has to be saved on each node.

One can construct matrices $\tilde{H}$ for an arbitrary number of processors p. We call the number of total data sets (bold rows) κ and the number of block data sets $\tilde{\kappa}$. The number of rows in the matrix is

$$\kappa + \tilde{\kappa} = k. \tag{46}$$

To get the full force on each block data set, its contents must be compared with every full dataset. So each block data set has to meet each full data set during one of the shifts. For each block data set there are $p - 1$ full data sets to be met. Thus

$$\kappa \times \tilde{\kappa} \geqslant p - 1. \tag{47}$$

Eq. (47) allows us to find all possible combinations of κ and $\tilde{\kappa}$. Table 4 gives the complete list for $p = 9$ processing elements. For all these combinations, one can find a possible hyper-systolic matrix that guarantees a complete force calculation. The $\tilde{H}$ matrix is constructed by using the following basis vector:

$$A_k = (0, \overbrace{\underbrace{1, 1, \ldots, 1}_{\tilde{\kappa}\ \text{times}}}^{\text{blockdata}}, \overbrace{1, \underbrace{\tilde{\kappa}, \tilde{\kappa}, \ldots, \tilde{\kappa}}_{\kappa-2\ \text{times}}}^{\text{fulldata}}). \tag{48}$$

The quantity κ determines the amount of memory needed, while k determines the amount of communication. This is because the number of shifts needed is $(\tilde{\kappa} + 1) + (\kappa - 1) = k$ (one shift for each $\tilde{\kappa}$-line, one shift for bringing the data to their home processors and $\kappa - 1$ shifts to update the full data sets).

The combination of values in the second line of Table 4 provides the fastest communication. The number of shifts k is minimal as is the redundancy of data κ. For optimal memory usage, the first line is the best choice which leads actually to a systolic force loop. The last line in Table 4 represents the shared memory approach where a copy of the whole dataset is kept on each processor.

We now compute the minimum number of shifts we have to perform with these algorithms. $\kappa\tilde{\kappa}$ is approximately $p - 1$ (Eq. (47)). Substituting into Eq. (46) we get:

$$k = \frac{p-1}{\kappa} + \kappa. \tag{49}$$

The quantity k is the number of shifts that are to be performed. Thus we want to minimize it:

$$\frac{\mathrm{d}k}{\mathrm{d}\kappa} = -\frac{p-1}{\kappa^2} + 1 = 0 \tag{50}$$

and so

$$\kappa = \sqrt{p-1}. \tag{51}$$

Table 4
κ, $\tilde{\kappa}$ and k for $p = 9$

κ	$\tilde{\kappa}$	k
1	8	9
2	4	6
3	3	6
4	2	6
5	2	7
6	2	8
7	2	9
8	1	9

Thus we need $\sqrt{p-1}$ full data sets on each processor. Substituting (51) into (49) we get

$$k = 2\sqrt{p-1} \tag{52}$$

and by virtue of (46),

$$\tilde{\kappa} = \kappa = \sqrt{p-1}. \tag{53}$$

The $\tilde{\kappa}+1$ shifts of the block data can be done with the nonblocking scheme described above. The $\kappa - 1$ update shifts have to be done, when the force calculation is completely finished and therefore cannot be moved to the background.

5.3. Performance

The performance of this class of algorithm will now be discussed for a blocking communication scheme. The calculation time, from Section 3, is:

$$t_\mathrm{f} = \frac{Ns}{p}\tau_\mathrm{f}. \tag{54}$$

The communication time for one shift is $\tau_\mathrm{l} + s\tau_\mathrm{c}/p$ (Eq. (25)) assuming that the s block particles are distributed equally over the processors, so that $s_i = s/p$. The number of shifts that have to be performed is $k = \kappa + \tilde{\kappa}$ (Eq. (46)). Thus the total communication time is

$$t_\mathrm{c} = k\left(\tau_\mathrm{l} + \frac{s}{p}\tau_\mathrm{c}\right). \tag{55}$$

Thus, using a blocking communication scheme, the total time for one integration step is

$$t = t_\mathrm{c} + t_\mathrm{f} = k\left(\tau_\mathrm{l} + \frac{s}{p}\tau_\mathrm{c}\right) + \frac{Ns}{p}\tau_\mathrm{f}. \tag{56}$$

In the case of minimal communication cost (Eq. (52)) the communication cost is

$$t_\mathrm{c} = 2\sqrt{p}\tau_\mathrm{l} + 2\frac{s}{\sqrt{p}}\tau_\mathrm{c}, \tag{57}$$

where we approximate k by $2\sqrt{p}$. We get the optimal number of processors that minimize the calculation time by setting the first derivative of p with respect to t to zero. This leads us to

$$p_\mathrm{opt}^{3/2}\tau_\mathrm{l} - p_\mathrm{opt}^{1/2}s\tau_\mathrm{c} - Ns\tau_\mathrm{f} = 0. \tag{58}$$

Since the hyper-systolic codes are aimed at very massive parallel machines, we assume that p is a number of order 100 or larger. The second term then becomes much smaller than the first one and:

$$p_\mathrm{opt} \approx \left(Ns\frac{\tau_\mathrm{f}}{\tau_\mathrm{l}}\right)^{2/3}. \tag{59}$$

As we measure in our benchmarks, the mean group size $s \propto N^{1/2}$. In this case $p_\mathrm{opt} \approx N(\tau_\mathrm{f}/\tau_\mathrm{l})^{2/3}$. So the optimal processor number is independent of the communication bandwidth of the host computer while this is not the case for the systolic algorithm in Eq. (21).

The performance just described is similar to that of the two-dimensional ring algorithm introduced by Makino [19]. If we choose $k = 2\sqrt{p}$ we get the same amount of calculation time and the same amount of communication as his algorithm needs. Makino's algorithm however is not strictly hyper-systolic. It is designed for a 2D network of processors and the systolic shifts are carried out within rows and columns of this 2D array. Such an algorithm is far less flexible than the hyper-systolic algorithm described above, for the following reasons. First, it only works for processor numbers p for which $\sqrt{p}$ is an integer. Second, Makino's algorithm corresponds to our hyper-systolic algorithm with $\kappa = \tilde{\kappa} = \sqrt{p}$. Therefore it requires that $\sqrt{p}$ full data arrays are saved on each processor. Our algorithm works with less memory without losing communication efficiency. For example, for $p = 9$, Makino's algorithm needs to save 3 datasets on each processor, while with the hyper-systolic algorithm, we can choose $\kappa = 2$ and $\tilde{\kappa} = 4$ (Table 4). k still equals 6. Thus the communication complexity is the same as in the case $\kappa = \tilde{\kappa} = 3$ while the memory requirements are reduced by one redundant full dataset. Also the number of shifts that cannot be done in a non-blocking way $(\kappa - 1)$ is reduced.

5.4. Small group sizes and the hyper-systolic algorithm

Above we discussed the effect of small group sizes on systolic algorithms. We defined a group to be small if not all processors provide particles to the group. We found that the systolic algorithm with nonblocking communication is able to deal with small group sizes in such a way that, over a large percentage of time, the force calculation is running parallel.

In a hyper-systolic scheme, the shifts of the block data can be done with blocking or nonblocking routines. Thus the same considerations as in Section 3.4 apply. In addition we encounter a new problem: in the hyper-systolic algorithm the particle data are not sent to each processor. For example in the case of $p = 8$ and a hyper-systolic matrix like in Eq. (45), calculations for particles provided by processor 1 are only done on processors 2, 3, 4, and 5. If we only have one particle in a group, and this is provided by processor 1, then only these four processors do calculations, while the remaining four processors wait until the next integration step. The performance would be comparable to a 4 processor parallel computation.

In general, if one chooses a hyper-systolic algorithm with the smallest communication complexity $\tilde{\kappa} = \sqrt{p-1}$ (Eq. (53)) and a group size of one, then only $\sqrt{p-1}$ processors are doing the computation.

If more than one but not all processors contribute particles to the group, the parallelization depends strongly on the specific processor number. For example, in the case of matrix (45), if processors 1 and 2 provide particles, 5 processors are working in parallel. If processors 1 and 5 provide particles, then all of the 8 processors are working in parallel.

6. An application: gravitational Brownian motion

6.1. The problem

The algorithms described here are ideally suited to problems requiring large N and small to moderate numbers of integration steps. One such problem is the Brownian motion of a massive point particle that responds to the gravitational force from N, less-massive particles. Let $M_\bullet$ and $\mathbf{R}$ be the mass and position of the Brownian particle, m the mass of a perturber, and $\mathbf{r}_j$ the position of the jth perturber particle. The equations of motion are

$$\ddot{\mathbf{R}} = -Gm \sum_{k=1}^{N} \frac{(\mathbf{R} - \mathbf{r}_k)}{|\mathbf{R} - \mathbf{r}_k|^3}, \tag{60}$$

$$\ddot{\mathbf{r}}_j = -Gm \sum_{k=1}^{N} \frac{(\mathbf{r}_j - \mathbf{r}_k)}{|\mathbf{r}_j - \mathbf{r}_k|^3} + GM_\bullet \frac{(\mathbf{R} - \mathbf{r}_j)}{|\mathbf{R} - \mathbf{r}_j|^3}, \tag{61}$$

where the summation in Eq. (61) excludes $k = j$. The total mass of the stellar system excluding the Brownian particle is $Nm \equiv M$.

This problem is relevant to the behavior of supermassive black holes at the centers of galaxies. Black holes have masses of order $10^6 M_\odot \lesssim M_\bullet \lesssim 10^9 M_\odot$, with $M_\odot$ the mass of the sun, compared with a total mass of a galactic nucleus of $\sim 10^9 M_\odot$ and of an entire galaxy, $\sim 10^{11} M_\odot$. By analogy with the fluid case, the RMS Brownian velocity of the black hole is expected to be of order

$$V_{\rm rms} \approx \sqrt{3}\sqrt{\frac{m}{M_\bullet}}\,\sigma, \tag{62}$$

where σ is the 1D velocity dispersion of stars in the vicinity of the black hole and m is a typical stellar mass. But Brownian motion in a self-gravitating system is expected to differ from that in a fluid, for a variety of reasons. A massive particle alters the potential when it moves and may excite collective modes in the background. Its effective mass may differ from $M_\bullet$ due to particles bound to it, and the force perturbations acting on it are not necessarily localized in time. For these reasons, it is important to treat the motion of the background particles in a fully self-consistent way, as in Eqs. (60) and (61).

Integrations involving large particle numbers are slow even with a parallel algorithm, but fortunately the time required for a massive particle (the "black hole") to reach a state of energy equipartition with the stars, $T_{\rm eq}$, is expected to be less than a single crossing time of the stellar system in which it is imbedded. We derive this result as follows. Starting from rest, the mean square velocity of the black hole should evolve approximately as

$$\langle V^2 \rangle \approx \langle \Delta v_\parallel^2 \rangle t, \tag{63}$$

where

$$\langle \Delta v_\parallel^2 \rangle = \frac{8\sqrt{2\pi}}{3}\frac{G^2 m\rho \ln \Lambda}{\sigma} \tag{64}$$

(e.g., [26]). Here ρ is the mass density of stars and $\ln \Lambda$ is the Coulomb logarithm, which is of order unity for this case [10] and will henceforth be set equal to one. The time $T_{\rm eq}$ required for $\langle V^2 \rangle$ to reach its expected equilibrium value of $\sim (m/M)\sigma^2$ is therefore

$$T_{\rm eq} \approx \frac{m}{M_\bullet}\frac{\sigma^2}{\langle \Delta v_\parallel^2 \rangle} \approx \frac{\sigma^3}{G^2 M_\bullet \rho}. \tag{65}$$

This may be written

$$T_{\rm eq} \approx T_D\left(\frac{M_{\rm gal}}{M_\bullet}\right)\left(\frac{\rho}{\langle\rho\rangle}\right)^{-1} \tag{66}$$

where $\langle\rho\rangle$ is the mean density of the galaxy within its half-mass radius $R_{1/2}$ and $T_D \equiv R_{1/2}/\sigma$, the crossing time within that radius.

A typical environment for a supermassive black hole is at the center of a galaxy in which the stellar density varies as

$$\rho \approx \langle\rho\rangle\left(\frac{r}{R_{1/2}}\right)^{-\gamma} \tag{67}$$

with $1 \lesssim \gamma \lesssim 2$ [27]. Thus

$$T_{\rm eq} \approx T_D\left(\frac{M_{\rm gal}}{M_\bullet}\right)\left(\frac{r}{R_{1/2}}\right)^{\gamma}. \tag{68}$$

To estimate the second factor in parentheses, we assume that the stars that most strongly affect the motion of the black hole are within a distance $r \approx GM_\bullet/\sigma^2 \approx (M_\bullet/M_{\rm gal})R_{1/2}$, the classical radius of influence of the black hole [29]. Then

$$T_{\rm eq} \approx T_D\left(\frac{M_\bullet}{M_{\rm gal}}\right)^{0.5}, \tag{69}$$

where γ has been set equal to 1.5, roughly the value characterizing the stellar density cusp surrounding the Milky Way black hole [28]. Setting $M_\bullet/M_{\rm gal} \sim 10^{-3}$ [23], we find $T_{\rm eq} \sim T_D/30$. We stress that this result is very approximate, since it is based on diffusion coefficient derived assuming an infinite homogenous medium. Eq. (69) nevertheless suggests that a massive particle will reach energy equipartition with the "stars" in less than one crossing time of the stellar system. Note that this result is independent of the masses of the perturbing particles and hence of N. However the predicted value of the equilibrium velocity dispersion of the black hole does depend on $m/M_\bullet$ (cf. Eq. (62)). We evaluate this dependence below via numerical experiments.

6.2. The experiments

We constructed initial conditions by distributing $N = 10^6$, equal-mass particles representing the stars according to Dehnen's law (35) with $\gamma = 1.5$,

$$\rho(r) = \frac{3M}{8\pi}\frac{a}{r^{1.5}(r+a)^{2.5}} \tag{70}$$

and a central point of mass $M_\bullet$ representing the black hole. The value chosen for the slope of the central density cusp, $\gamma = 1.5$, is similar to the value near the center of the Milky Way galaxy [28]. The velocities of the "star" particles were selected from the unique isotropic distribution function that reproduces a steady-state Dehnen density law in the presence of a central point mass [30]. The initial velocity of the black hole particle was set to zero. We carried out integrations for five different values of the black hole mass, $M_\bullet = (0.00001, 0.0001, 0.001, 0.01, 0.1)$ in units where the total mass M in "stars" is unity. The gravitational constant G and the Dehnen-model scale length a were also set to one. Integrations were carried out until a time of $t_{\rm max} = 0.1$, compared with a crossing time for the overall system of $a/\sigma \approx 1$. All runs were made using 128 processors on the Cray T3E 900 supercomputer in Stuttgart. The average wall clock computing time per simulation was 136 h.

Fig. 9 shows the time dependence of the squared velocity of the black hole particle, $V^2(t)$, for each of the runs. The motion appears to reach a statistical steady state in much less than one crossing time, as expected, and the inverse dependence of $\langle V^2\rangle$ on $M_\bullet$ is apparent. There were no discernable changes in the spatial or velocity distribution of the "star" particles during these runs outside of the region where the black hole wandered.

Fig. 10 shows $V_{\rm rms}$ as a function of $M_\bullet$ for each of the runs. $V_{\rm rms}$ was computed by averaging $V^2(t)$ over the full integration interval. From Eq. (62), and setting $m = 1/N = 10^{-6}$, we expect

$$V_{\rm rms} \approx 1.73 \times 10^{-3}\sigma M_\bullet^{-0.5}, \tag{71}$$

where σ is the 1D velocity dispersion of stars in the vicinity of the black hole particle. However it is not clear what value of σ should be used in this formula, since the velocity dispersion of the stars is a function of radius in the initial models: σ increases toward the center to a maximum of $\sim$0.42 at a radius of ~ 0.2, then drops slowly inward before rising again as $\sigma^2 \approx 2/5r$ within the radius of influence of the black hole. Using the peak value of σ in the black-hole-free Dehnen model, $\sigma = 0.42$, Eq. (71) predicts

$$V_{\rm rms} \approx 7.27 \times 10^{-4} M_\bullet^{-0.5}. \tag{72}$$

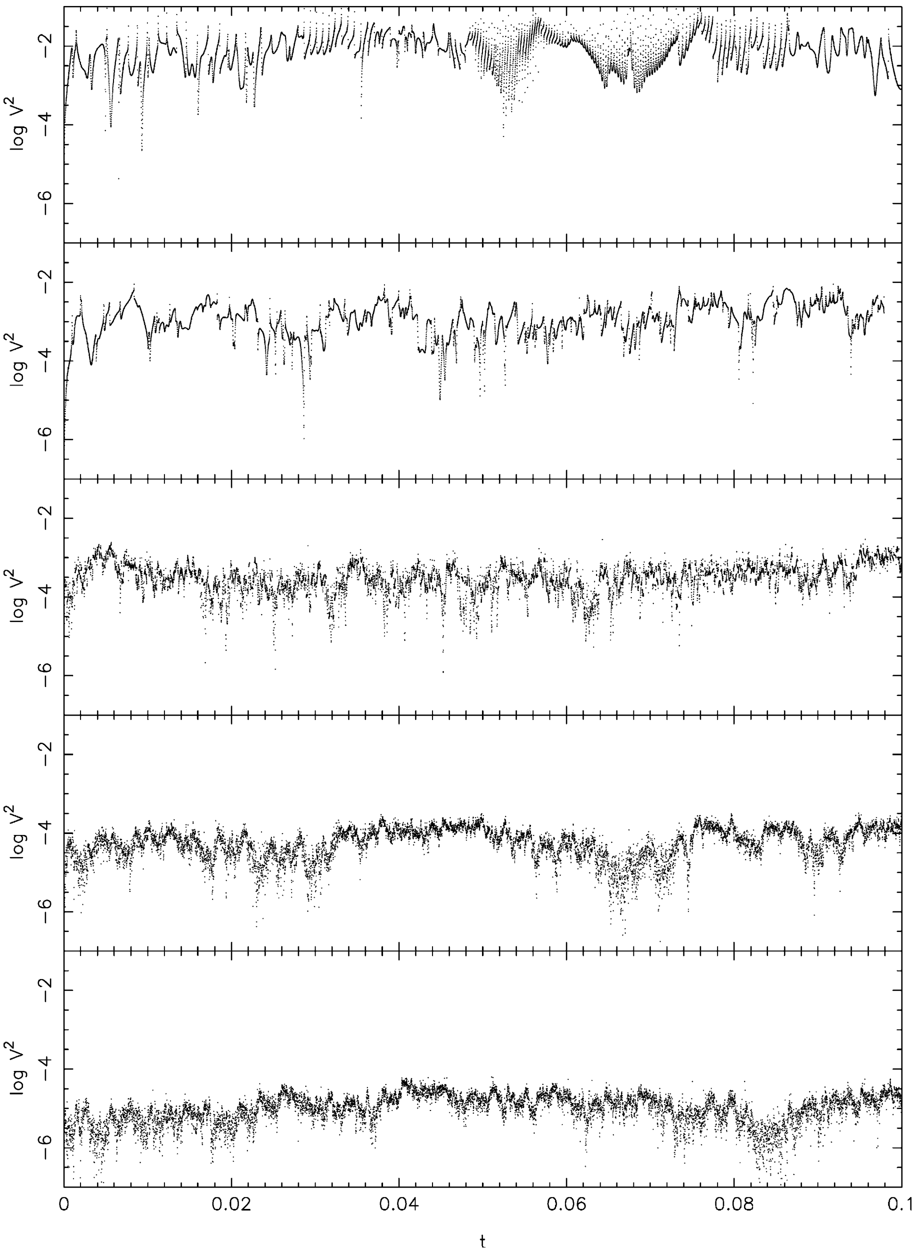

Fig. 9. Time evolution of the squared velocity of the massive ("black hole") particle in the Brownian motion experiments. The background stellar system consists of $N = 10^6$ particles of mass $m = 10^{-6}$ distributed according to a Dehnen density law, Eq. (70). The mass $M_\bullet$ of the black hole particle increases downward by a factor of 10 between each frame, from $M_\bullet = 10^{-5}$ at the top to $M_\bullet = 10^{-1}$ at the bottom. The black hole particle is started at rest and rapidly comes into energy equipartition with the lighter particles. The amplitude of its random motion varies inversely with its mass as expected from energy equipartition arguments.

This line is shown in Fig. 10. The measured values of $V_{\rm rms}$ are consistent with this prediction, although there is a hint that the dependence of $V_{\rm rms}$ on $M_\bullet$ is slightly less steep than $M_\bullet^{-0.5}$. This result suggests that gravitational Brownian motion may be very similar to its fluid counterpart. In particular, we note that $V_{\rm rms}$

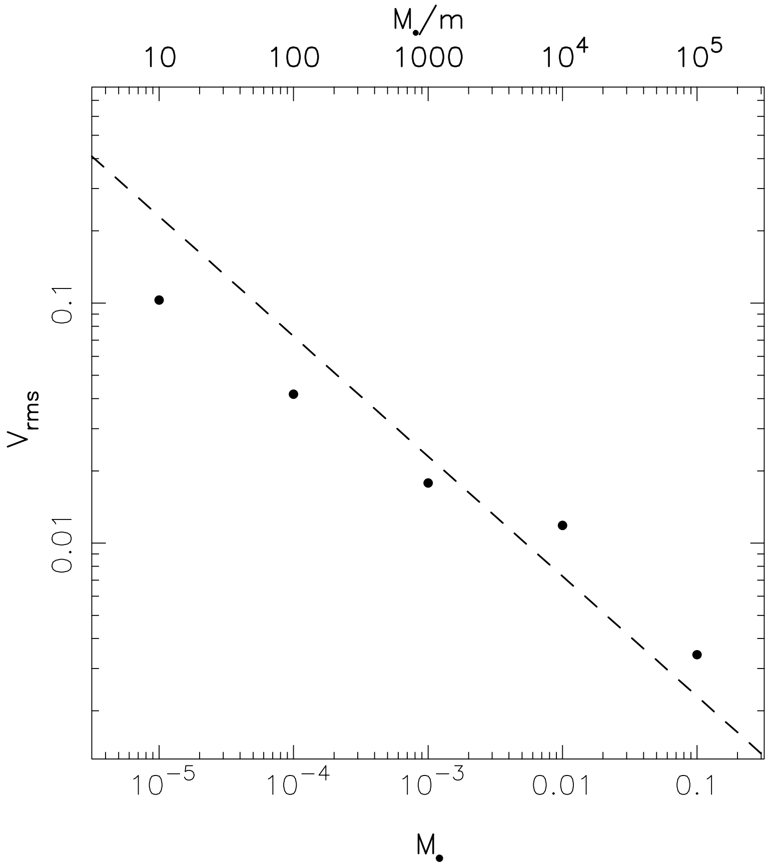

Fig. 10. RMS velocity of the "black hole" particle in the Brownian motion experiments. $V_{\rm rms}$ was computed using the average value of $V^2(t)$ between $t = 0$ and $t = 0.1$. Dashed line is the predicted relation based on energy equipartition arguments (see text). m is the mass of a "star" particle.

is correctly predicted by Eq. (62) if one uses a value for σ in that equation that is measured well outside of the region of gravitational influence of the black hole. This suggests that most of the perturbations leading to the black hole's motion come from distant stars.

We can use our results to estimate the random velocity of the supermassive black hole at the center of the Milky Way galaxy. It has recently become feasible to measure the motion of the Milky Way black hole [31,32], whose mass is $M_\bullet \approx 2.5$–$3.5 \times 10^6 M_\odot$ [33–35], roughly 10^{-3} times the mass of the Milky Way bulge [23]. The masses of stars in the cluster surrounding the black hole are of order $10 - 20 M_\odot$ [36]. We adopt $\sigma \approx 100\,\mathrm{km\,s^{-1}}$, roughly the peak velocity dispersion measured for the stars in the Milky Way bulge outside of the region of influence of the black hole [37]; here we make use of our result that the Brownian velocity in the N-body simulations is correctly predicted by Eq. (62) when σ is replaced by its peak value outside of the region of influence of the central black hole. The resulting prediction for the 3D random velocity of the Milky Way black hole is

$$V_{\rm rms} \approx 0.40\left(\frac{m_*}{15M_\odot}\right)^{1/2}\left(\frac{M_\bullet}{3\times10^6 M_\odot}\right)^{-1/2}\mathrm{km\,s^{-1}}. \tag{73}$$

The predicted velocity could be much greater than $0.4\,\mathrm{km\,s^{-1}}$ if the objects providing the force perturbations are much more massive than $10M_\odot$, e.g., giant molecular clouds. Current upper limits on the proper motion (2D) velocity of the Milky Way black hole are $\sim 20\,\mathrm{km\,s^{-1}}$ [31,32].

7. Conclusions

We have introduced two variants of a systolic algorithm for parallelizing direct-summation N-body codes implementing individual block time step integrators: the first with blocking communication, and the second with non-blocking communication. Performance tests were carried out using N-body models similar to those commonly studied by dynamical astronomers, in which the gravitational forces vary widely between core and halo and for which the particle block sizes are typically very small. The nonblocking scheme was found to provide far better performance than the blocking scheme for such systems, providing a nearly ideal speedup for the force calculations. By engaging a sufficient number of computing nodes, particle numbers in excess of 10^6 are now feasible for direct N-body simulations. For parallel machines with very large processor numbers, we describe the implementation of a hyper-systolic computing scheme which provides a communication scaling of $\mathrm{O}(\sqrt{p})$ at the expense of increased memory demands.

The algorithms described here are not well suited to dealing with the formation of close, bound pairs of particles (binaries). A standard way of dealing with binaries is via regularization [2]. In a future paper, we will discuss the implementation of regularization schemes in systolic algorithms.

The codes used to write this paper are available for download at: `http://www.physics.rutgers.edu/marchems/download.html`.

Acknowledgements

This work was supported by NSF Grant 00-71099 and by NASA Grants NAG5-6037 and NAG5-9046 to D.M. We thank Th. Lippert, W. Schroers, and K. Schilling for their encouragement and help. We are grateful to the NASA Center for Computational Sciences at NASA-Goddard Space Flight Center, the National Computational Science Alliance, the Center for Advanced Information Processing at Rutgers University, the John von Neumann Institut für Computing in Jülich, and the Höchstleistungsrechenzentrum in Stuttgart for their generous allocations of computer time.

References

[1] S.J. Aarseth, From NBODY1 to NBODY6: the growth of an industry, Pub. Astron. Soc. Pac. 111 (1999) 1333.
[2] P. Kustaanheimo, E. Stiefel, Perturbation theory of Kepler motion based on spinor regularization, J. Reine Angew. Math. 218 (1965) 204.
[3] J. Barnes, P. Hut, Error analysis of a tree code, Astrophys. J. Suppl. 70 (1989) 389.
[4] A.J. Allen, P.L. Palmer, J. Papaloizou, A conservative numerical technique for collisionless dynamical systems, Mon. Not. R. Astron. Soc. 242 (1990) 576.
[5] R.H. Miller, K.H. Prendergast, Stellar dynamics in a discrete phase space, Astrophys. J. 151 (1968) 699.
[6] R. Spurzem, Direct N-body simulations, J. Comp. Appl. Math. 109 (1999) 407.
[7] D. Merritt, Black holes and galaxy dynamics, in: D. Merritt, J. Sellwood, M. Valluri (Eds.), Galaxy Dynamics, Astronomical Society of the Pacific Conference Series, vol. 182, 1999, p. 164.
[8] S. Aarseth, Star cluster simulations: The state of the art.
[9] M. Giersz, D.C. Heggie, Statistics of N-body simulations – Part One – equal masses before core collapse, Mon. Not. R. Astron. Soc. 268 (1994) 257.
[10] M. Milosavljevic, D. Merritt, Formation of galactic nuclei, Astrophys. J. 563 (2001) 34.

[11] S.V.W. Beckwith, in: C.J. Lada, N. Kylafis (Eds.), The Physics of Star Formation and Early Stellar Evolution, Kluwer Academic Publishers, Dordrecht, 1999, p. 579.
[12] H.T. Kung, C.E. Leiserson, Systolic arrays (for vlsi), in: I.S. Duff, G.W. Stewart (Eds.), Sparse Matrix Processing 1978, SIAM, Philadelphia, 1979, p. 256.
[13] H.T. Kung, Why systolic architectures? Computer 15 (1982) 37.
[14] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Douglas, MPI – The complete reference, The MPI core, vol. 1, 2nd ed., MIT Press, Cambridge, MA, 1998, p. 68.
[15] J. Makino, S.J. Aarseth, On a hermite integrator with Ahmad–Cohen scheme for gravitational many-body problems, Pub. Astron. Soc. Jpn. 44 (1992) 141.
[16] R. Rabenseifner, Automatic MPI counter profiling of all users: First results on a CRAY T3E 900-512, in: Proceedings of the Message Passing Interface Developer's and User's Conference 1999 (MPIDC'99), Atlanta, USA, 1999, p. 77.
[17] T. Lippert, G. Ritzenhöfer, U. Glaessner, H. Hoeber, A. Seyfried, K. Schilling, Hyper-systolic processing on ape100/quadrics: N^2–loop computations, Int. J. Mod. Phys. C 7 (1998) 485 (hep-lat/9512020).
[18] T. Lippert, A. Seyfried, A. Bode, K. Schilling, Hyper-systolic parallel computing, IEEE Trans. Parallel Distrib. Syst. 9 (1998) 1.
[19] J. Makino, An efficient parallel algorithm for $O(N^2)$ direct summation method and its variations on distributed memory parallel machines, astro-ph/0108412 (2001).
[20] J. Makino, P. Hut, Performance analysis of direct N-body calculations, Astrophys. J. Suppl. Ser. 68 (1988) 833.
[21] H.C. Plummer, On the problem of distribution in globular star clusters, Mon. Not. R. Astron. Soc. 71 (5) (1911) 460.
[22] W. Dehnen, A family of potential–density pairs for spherical galaxies and bulges, Mon. Not. R. Astron. Soc. 265 (1993) 250.
[23] D. Merritt, L. Ferrarese, Black hole demographics from the $M_\bullet$–σ relation, Mon. Not. R. Astron. Soc. 320 (2001) L30.
[24] D.C. Heggie, R.M. Mathieu, Standardised units and time scales, in: P. Hut, S.L.W. McMillan (Eds.), The Use of Supercomputers in Stellar Dynamics (New York), 1986, p. 233.
[25] T. Lippert, P. Palazzari, K. Schilling, Automatic template generation for solving N^2 problems on parallel systems with arbitrary topology, in: Proceedings of the IEEE Workshop on Parallel and Distributed Software Engineering, IEEE, Boston, 1997.
[26] D. Merritt, Brownian motion of a massive binary, Astrophys. J. 556 (2001) 245.
[27] K. Gebhardt et al., The centers of early-type galaxies with HST. III. Non-parametric recovery of stellar luminosity distribution, Astron. J. 112 (1996) 105.
[28] T. Alexander, The distribution of stars near the supermassive black hole in the galactic center, Astrophys. J. 527 (1999) 835.
[29] P.J.E. Peebles, Star distribution near a collapsed object, Astrophys. J. 178 (1972) 371.
[30] S. Tremaine, D.O. Richstone, Y.-I. Byun, A. Dressler, S.M. Faber, C. Grillmair, J. Kormendy, T.R. Lauer, A family of models for spherical stellar systems, Astron. J. 107 (1994) 634.
[31] D.C. Backer, R.A. Sramek, Proper motion of the compact, nonthermal radio source in the galactic center, Sagittarius A*, Astrophys. J. 524 (1999) 805.
[32] M.J. Reid, A.C.S. Readhead, R.C. Vermeulen, R.N. Treuhaft, The proper motion of Sagittarius A*. I. First VLBA results, Astrophys. J. 524 (1999) 816.
[33] A.M. Ghez, B.L. Klein, M. Morris, E.E. Becklin, High proper-motion stars in the vicinity of Sagittarius A*: evidence for a supermassive black hole at the center of our galaxy, Astrophys. J. 509 (1998) 678.
[34] R. Genzel, C. Pichon, A. Eckart, O.E. Gerhard, T. Ott, Stellar dynamics in the galactic centre: proper motions and anisotropy, Mon. Not. R. Astron. Soc. 317 (2000) 348.
[35] D. Chakrabarty, P. Saha, A nonparametric estimate of the mass of the central black hole in the galaxy, Astron. J. 122 (2001) 232.
[36] R. Genzel, A. Eckart, T. Ott, F. Eisenhauer, On the nature of the dark mass in the center of the Milky Way, Mon. Not. R. Astron. Soc. 291 (1997) 219.
[37] S. Kent, Galactic structure from the Spacelab Infrared Telescope. III. A dynamical model for the Milky Way bulge, Astrophys. J. 387 (1992) 181.