THE ASTROPHYSICAL JOURNAL, 480:432–446, 1997 May 1

GRAPE-4: A MASSIVELY PARALLEL SPECIAL-PURPOSE COMPUTER FOR COLLISIONAL N-BODY SIMULATIONS

JUNICHIRO MAKINO
Department of Graphics and Information Science, College of Arts and Sciences, University of Tokyo, 3-8-1 Komaba, Meguro-ku, Tokyo 153, Japan

AND

MAKOTO TAIJI,[1] TOSHIKAZU EBISUZAKI, AND DAIICHIRO SUGIMOTO
Department of Graphics and Information Science, Department of Earth Science and Astronomy, College of Arts and Sciences, University of Tokyo, 3-8-1 Komaba, Meguro-ku, Tokyo 153, Japan
Received 1996 October 9; accepted 1996 November 25

ABSTRACT

In this paper, we describe the architecture and performance of the GRAPE-4 system, a massively parallel special-purpose computer for N-body simulation of gravitational collisional systems. The calculation cost of N-body simulation of collisional self-gravitating system is $O(N^3)$. Thus, even with present-day supercomputers, the number of particles one can handle is still around 10,000. In N-body simulations, almost all computing time is spent calculating the force between particles, since the number of interactions is proportional to the square of the number of particles. Computational cost of the rest of the simulation, such as the time integration and the reduction of the result, is generally proportional to the number of particles. The calculation of the force between particles can be greatly accelerated by means of a dedicated special-purpose hardware. We have developed a series of hardware systems, the GRAPE (GRAvity PipE) systems, which perform the force calculation. They are used with a general-purpose host computer which performs the rest of the calculation. The GRAPE-4 system is our newest hardware, completed in 1995 summer. Its peak speed is 1.08 TFLOPS. This speed is achieved by running 1692 pipeline large-scale integrated circuits (LSIs), each providing 640 MFLOPS, in parallel.

Subject headings: instrumentation: miscellaneous — methods: numerical

1. INTRODUCTION

The N-body simulation technique, in which the equations of motion of N particles are integrated numerically, has been one of the most powerful tools for the study of astronomical objects such as the solar system, star clusters, galaxies, clusters of galaxies and large-scale structures of the universe.

The calculation cost of direct N-body simulation increases rapidly as we increase the number of particles. Because of the following two reasons, the calculation cost of N-body simulations of star clusters is roughly proportional to N^3. The first is that the gravity is a long-range attractive force. We cannot neglect the contributions of distant particles to the force on a particle. In many other particle simulations, the force is of a short-range nature, as in the case of the van der Waals force or the contributions of the distant particles can be neglected.

The second reason is that for collisional systems, the timescale of the evolution of the system is proportional to the number of particles. For star clusters such as open clusters and globular clusters, the thermal relaxation timescale is considerably shorter than the age of the universe. Thus, the simulation of such systems must cover the thermal timescale. The ratio between the thermal timescale of the cluster and the orbital timescale of a typical star in the cluster is proportional to $N/\log N$.

Simulations with large N are very expensive. For example, a simulation of a 10^5-body system would take several hundred years, if performed on a supercomputer with the effective speed of 1 GFLOPS (Hut, Makino, & McMillan 1988). Very roughly speaking, the number of particles one can handle increased by a factor of 10 every 15 years, which is consistent with the fact that the speed of the computers has been improved by a factor of 100 every 10 years.

This slow progress has been the major obstacle to the study of the dynamical evolution of globular clusters. Though some of the important phenomena could be studied using more approximate approaches such as a conducting gas sphere or an orbit-averaged one-dimensional Fokker-Planck equation, many interesting problems could not. Just to give an example, the number of exotic objects such as millisecond pulsars is known to vary widely among different clusters. For example, 47 Tuc has about one-third of all millisecond pulsars found so far in globular clusters (Robinson et al. 1995). Thus, the formation rate of these objects seems to depend strongly on the dynamics of clusters. To model the formation and evolution of these objects in an evolving globular cluster is a very complex problem, since the behavior of these objects has rather strong effect on the evolution of the cluster as a whole. Thus, the only reliable way is to perform a direct N-body simulation of the whole cluster.

Even the evolution of an idealized cluster of point-mass particles cannot be fully understood by means of approximate methods. In the case of an isolated cluster, whether the core oscillation (Sugimoto & Bettwieser 1983) would take place or not needs to be tested by an N-body simulation. The number of the particles in the core at the maximum contraction is so small that the Fokker-Planck approximation breaks down. Moreover, the evolution of a cluster with a high fraction of primordial binaries is difficult to study with the Fokker-Plank method (Hut et al. 1992;

[1] Present address: The Institute of Statistical Mathematics, 4-6-7 Minami-Azabu, Minato-ku, Tokyo 106, Japan.

McMillan, Hut, & Makino 1990). If we take into account the tidal field of the parent galaxy, a recent argument by Weinberg suggests that the effect of the nonspherical, time-dependent tidal field is much more important than had been believed (Weinberg 1994a, 1994b, 1994c). All these can be easily studied if an N-body simulation with a sufficiently large number of particles is possible but are very difficult or impossible to study with more approximate methods.

The increase of the number of particles of a factor of 10 in more than 10 years appears to be quite slow, when compared with the advance in the number of particles in cosmological N-body simulations. In the late 1970s, the number of particles one could use for cosmological simulations was around 1000. At the present, many people use routinely more than 10 million particles. Thus, the increase of the number of particles by a factor of 10,000 in 20 years has been achieved for cosmological simulations (see, e.g., Warren et al. 1992).

The reason that the number of particles in cosmological simulations has increased so rapidly is that the calculation cost is $O(N \log N)$. The simulation covers the dynamical timescale of the universe. Therefore, the number of time steps does not depend strongly on the number of particles. Moreover, fast algorithms to calculate the gravitational potential such as the PM scheme or Barnes-Hut tree algorithm (Barnes & Hut 1986) can be used because the required accuracy is low. In particular, the invention of the tree algorithm made it possible to compute the gravitational potential of particles in a highly clustered distribution, which had been difficult with PM or traditional P^3M algorithms. The P^3M method with an adaptive multiple grid (Couchman, Thomas, & Pearce 1995), which has become available recently, seems to be able to provide performance comparable, if not superior, to the tree algorithms.

For simulations of star clusters, it is difficult to apply these fast techniques to calculate the interparticle forces. The simulation of the long-term evolution of star clusters requires very accurate calculation of the interparticle force. Moreover, the number density and orbital timescale of stars range over many orders of magnitude in a single cluster. As a result, the fixed resolution of the PM scheme is not appropriate. The P^3M scheme is not appropriate either, since the calculation cost of a grid cell with high density becomes very large. The tree algorithm does not help much, since the required accuracy of the force is high (McMillan & Aarseth 1993).

There is another reason that the fast algorithms are difficult to use. As stated above, the orbital timescale of particles ranges over many orders of magnitudes. In addition, the fraction of particles that require short time steps is relatively small. It is impractical to use the same time step for all particles.

To overcome this difficulty, an algorithm called the individual time step scheme has been used (Aarseth 1985). The basic idea is to allow particles to have their own times and time steps. As a result, integration proceeds by updating the particle with minimum $t_i + \Delta t_i$, where t_i and Δt_i are the current time and time step of particle i.

This individual time step scheme, sometimes referred to as the Aarseth scheme, is an extremely powerful method when applied to the late stage of the evolution of star clusters. The gain in the computational speed is $O(N)$ (Makino & Hut 1988) at the late stage of evolution. This $O(N)$ gain implies that only a small number of particles have small time steps.

For the individual time step algorithm, Hut et al. (1988) estimated that the maximum possible speed-up factor for a sophisticated scheme such as the tree algorithm over the direct summation would be around a factor of 100, even for a gigantic calculation using 10^5 particles. Such a calculation, if performed on a supercomputer with the sustained speed of 1 GFLOPS, would take several hundreds years.

At first sight, this factor of 100 looks pretty large. However, the efficiency of parallel/vector machines is much better for the direct summation because of its simplicity.

As parallel/vector computers become more widely available, the algorithmic gain of sophisticated algorithms becomes less important. In practice, it is already a significant challenge to implement the straightforward direct summation method with an individual time step algorithm on massively parallel computers because a naive implementation requires very fast interprocessor communication. The attempt to implement the Ahmad-Cohen neighbor scheme has not been very successful so far. The tree algorithm without individual time steps is perfectly vectorizable (Barnes 1990; Hernquist 1990; Makino 1990) as well as parallelizable (Barnes 1986; Salmon & Warren 1994). Most schemes, however, rely on the fact that forces on all particles can be evaluated in parallel. For the individual time step algorithm, the degree of the parallelism is much smaller than the number of particles. Thus, the efficient parallel implementation is very difficult.

In this paper, we describe a rather different approach to this problem, which is to build a dedicated special-purpose computing device tailored for the requirement of the simulation of collisional N-body simulation with individual time step algorithm.

The idea of building a computer by ourselves might sound rather silly, since to develop a computer is a very expensive and risky enterprise. Even if we limit our attention to the field of large-scale scientific computation, it seems very difficult to design and build a successful machine. Even the most successful company could not maintain its existence.

If we limit the application to a very narrow range, the design is extremely simplified, and the price-performance ratio is improved by several orders of magnitude. An example of such a special-purpose hardware is the FX processor for the radio interferometer (Chikada et al. 1987), which is a dedicated hardware to perform the FFT operation. In the early 1980s, it achieved an incredible 100 GOPs (giga operations per second) for the total budget of 200 M yen.

In this paper, we describe the GRAPE-4 system, which made it possible to perform a direct N-body simulation of small globular clusters ($N \leq 10^5$). It is a massively parallel computer that consists of 1692 processor chips. Each processor chip integrates about 15 floating point arithmetic units and one function evaluator. The peak speed of a chip is 640 MFLOPS, and that of the total machine is 1.08 TFLOPS. The hardware was completed in 1995 June, and it is the fastest computer in the world as of 1996 summer.

GRAPE-4 is now being used for the study of various problems, such as the core oscillation of the globular cluster (Makino 1996), the evolution of open clusters (Aarseth 1996), the evolution of the black hole binary in the center of a galaxy (Makino & Ebisuzaki 1996; Makino 1997), the formation of the dark matter halo (Fukushige & Makino

1997), and the evolution of the system of planetesimals (Kokubo & Ida 1996a, 1996b). In this paper we describe the basic idea, the hardware, the software, the achieved performance, and future prospects. In § 2, we describe the basic concept of GRAPE systems and how they work for the case of the individual time step algorithm. In § 3, we describe the GRAPE-4 hardware. In § 4, we describe the software. In § 5, we present the measured performance. In § 6, we discuss the future of the GRAPE system.

2. BASIC CONCEPTS

2.1. *GRAPE*

The basic idea of GRAPE (GRAvity PipE) is shown in Figure 1. A special-purpose hardware is connected to a general-purpose front end. The special-purpose hardware is used as a back-end processor, on which only the force (and related) calculations are performed. The rest of the computation, such as the actual orbit integration, is performed on the host computer. For almost all systems, we used a UNIX-based workstation as the host computer. In the simplest case, the host computer sends the positions and masses of all particles to GRAPE. Then GRAPE calculates the forces between particles and sends them back to the host computer.

We have developed several hardware systems, namely GRAPE-1 (Ito et al. 1990), GRAPE-1A (Fukushige et al. 1991), and GRAPE-3/3A (Okumura et al. 1993), which are straightforward realizations of the concept shown in Figure 1 (GRAPE-2 will be discussed later). All systems calculate the force using the specialized pipeline processor. The difference between GRAPE-1 and GRAPE-3 is that in GRAPE-1, the force calculation pipeline was implemented using off-the-shelf IC chips, while for GRAPE-3, we developed a custom LSI chip that implements a complete force calculation pipeline in a single chip. GRAPE-3 integrated 48 of these chips operating in parallel, each providing the speed of 300 MFLOPS. The peak speed of GRAPE-3 was 14.4 GFLOPS. In GRAPE-3, these 48 chips calculate the forces on 48 different particles in parallel. These chips calculate the forces *from* the same particles. Thus, all chips share one memory unit. In practice, GRAPE-3 consists of two boards, each with its own memory, but in the standard operation mode, the contents of the two memory units are identical. Figure 2 shows the structure of one board of GRAPE-3.

The direct summation algorithm with a shared (and in most cases constant) time step is the simplest application of these hardware systems. They can also be used to accelerate the Barnes-Hut tree algorithm (Makino 1991a) and the P^3M algorithm (Brieu, Summers, & Ostriker 1995).

The main reason for building a special-purpose computer is that it can achieve a better price-performance ratio better than that of general-purpose computers. The production

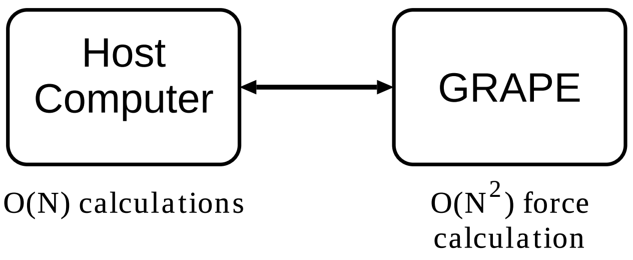

FIG. 1.—Basic structure of the GRAPE system

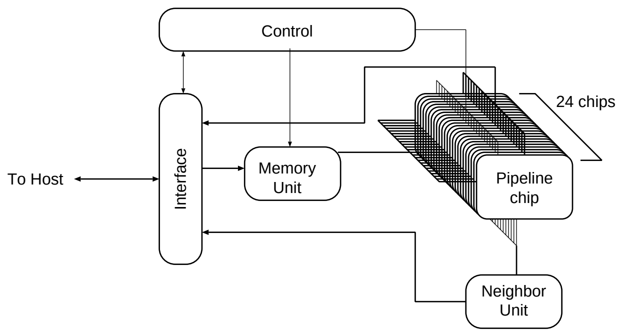

FIG. 2.—Block diagram of a GRAPE-3 board

cost of the GRAPE-1A system was about $7000. Its theoretical peak speed was 240 MFLOPS, and half of the peak speed was achieved on direct summation for a few thousand particles. The development cost of GRAPE-3 was about $100,000. Most of the cost was spent for the design and manufacture of the pipeline chip. The reproduction cost of GRAPE-3 system would be around $10,000. Thus, even though the total cost of GRAPE-3 was 10 times higher than the reproduction cost, it still offered raw performance that was at least 100 times faster than the performance of a general-purpose computer of a similar price.

Once a custom chip is designed, the manufacturer can produce a number of identical chips for the cost of about $100 per chip. On the other hand, designing of a custom chip is a rather costly venture. Therefore, it is crucial to use a reasonably large number of custom chips in parallel to achieve a good price performance.

With the GRAPE architecture, it is easy to use a number of pipelines in parallel, since all chips can share one memory unit. Thus, if the total amount of budget is larger than the initial design cost of the custom LSI, GRAPE can almost always achieve very high price-performance.

A number of research institutes both within and outside Japan acquired copies of the GRAPE-3A board. These institutes include Princeton University, University of California at Berkeley, Marseille Observatory, Max Planck Institute for Astrophysics, Edinburgh University, Tokyo University, Kyoto University, and Tohoku University. As of 1996 August, more than 20 laboratories have acquired more than 40 GRAPE-3A boards.

2.2. *Individual Time Step Algorithm*

As stated earlier, the individual time step algorithm (Aarseth 1963) is the essential part of any simulation program for gravitational collisional systems.

In the individual time step algorithm, each particle has its own time step Δt_i and maintains its own time t_i. To integrate the system, one first selects the particle for which the next time $(t_i + \Delta t_i)$ is the minimum. Then, one predicts its position at this new time $t = t_i + \Delta t_i$. Positions of all other particles at time t must be predicted also. Then the force on that particle from other particles is calculated following Newton's law of gravity. The position and velocity of the particle are then corrected, and the new time step is calculated. The integration scheme is a variant of Krogh's scheme (Krogh 1974) modified for second-order equations. It is essentially a classical Adams-Bashforth-Moulton (ABM) linear multistep predictor-corrector method, modified to allow variable time steps. The order of the integrator is four. For a wide range of the required accuracy and the number of particles, the fourth-order scheme is close to optimal (Makino 1990), though there had been some claims that higher order scheme would be more efficient (Press & Spergel 1988).

A modification of this individual time step algorithm is now used to achieve higher efficiency on vector/parallel machines (McMillan 1986) and special-purpose computers (Makino 1991b). This scheme is called the hierarchical time step scheme. In this scheme, the time steps of particles are forced to integer powers of 2. In addition, the time step of a particle is chosen so that the current time of that particle is an integer multiple of the time step. These two criteria make it possible to force many particles to share exactly the same time.

In the original individual time step algorithm, only one particle can be integrated at one time. Thus, the parallelism is rather limited. One still can use multiple processors to calculate the force from other $N-1$ particles in parallel, but the speed-up is rather limited.

In the hierarchical time step algorithm, a fairly large number of particles can share the same time. Theoretically, the average number of particles to share the same time is $O(N_c^{2/3})$, where N_c is the number of particles in the core of the cluster (Makino 1991b).

It should be noted that the estimates for the optimal order given in Makino (1990) or Press & Spergel (1988) are for the original individual time step algorithm. In this case, the calculation cost per one particle step increases as we increases the order of the integrator. However, for the hierarchical time step scheme, the calculation cost is almost independent of the order of the integrator. Therefore, the integrator with order higher than 4 might be preferred.

2.3. *The Hermite Integration Scheme*

As stated before, the standard time integration method for collisional N-body simulation had been the variable–step-size linear multistep method. The implementation of such a scheme is rather complicated because it requires a considerable amount of bookkeeping. Roughly speaking, one has to maintain the calculated accelerations at several previous time steps as well as the previous time steps themselves.

When the time integration is started, the accelerations at previous time steps are not available. Therefore, a special procedure to start up the integration is required. The initialization of the integrator occurs rather often, since we apply special procedures for close encounters and close binaries (see Aarseth 1985). Standard bootstrapping is not appropriate since it would significantly increase the calculation cost. Aarseth implemented the start-up procedure in which the time derivatives of the gravitational forces are analytically calculated and converted to forces at the previous time steps.

The fourth-order Hermite integrator (Makino 1991a; Makino & Aarseth 1992) is much simpler than the ABM scheme, and yet it offers similar (but somewhat better) accuracy for the same calculation cost. The Hermite scheme uses the Hermite interpolation formula to construct the predictor and the corrector. To construct a Hermite interpolation formula, both the values of the function and its derivatives are used. For example, if we know the values of the acceleration and its first time derivative at two points in time, we can construct a third-order interpolation polynomial. If the information of the time derivative is not available, we need values of accelerations at four different points in time.

The predictor of the fourth-order Hermite scheme is expressed as

$$\boldsymbol{x}_p = \frac{\Delta t^3}{6}\dot{\boldsymbol{a}}_0 + \frac{\Delta t^2}{2}\boldsymbol{a}_0 + \Delta t\boldsymbol{v}_0 + \boldsymbol{x}_0 \tag{1}$$

$$\boldsymbol{v}_p = \frac{\Delta t^2}{2}\dot{\boldsymbol{a}}_0 + \Delta t\boldsymbol{a}_0 + \boldsymbol{v}_0\,, \tag{2}$$

where $\boldsymbol{x}_p$ and $\boldsymbol{v}_p$ are the predicted position and velocity; $\boldsymbol{x}_0$, $\boldsymbol{v}_0$, $\boldsymbol{a}_0$, and $\dot{\boldsymbol{a}}_0$ are the position, velocity, acceleration, and its time derivative at time t_0; and Δt is the time step.

The acceleration $\boldsymbol{a}$ and its time derivative $\dot{\boldsymbol{a}}$ are calculated as

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

$$\dot{\boldsymbol{a}}_i = \sum_j Gm_j \left[\frac{\boldsymbol{v}_{ij}}{(r_{ij}^2+\epsilon^2)^{3/2}} - \frac{3(\boldsymbol{v}_{ij}\cdot\boldsymbol{r}_{ij})\boldsymbol{r}_{ij}}{(r_{ij}^2+\epsilon^2)^{5/2}}\right], \tag{4}$$

where

$$\boldsymbol{r}_{ij} = \boldsymbol{x}_j - \boldsymbol{x}_i\,, \tag{5}$$

$$\boldsymbol{v}_{ij} = \boldsymbol{v}_j - \boldsymbol{v}_i\,. \tag{6}$$

Here, ϵ is the softening parameter.

The corrector is given by

$$\boldsymbol{x}_c = \frac{\Delta t^5}{120}\boldsymbol{a}^{(3)} + \frac{\Delta t^4}{24}\boldsymbol{a}^{(2)} + \boldsymbol{x}_p \tag{7}$$

$$\boldsymbol{v}_c = \frac{\Delta t^4}{24}\boldsymbol{a}^{(3)} + \frac{\Delta t^3}{6}\boldsymbol{a}^{(2)} + \boldsymbol{v}_p\,. \tag{8}$$

Here, $\boldsymbol{a}^{(2)}$ and $\boldsymbol{a}^{(3)}$ are the second and third derivatives of the acceleration at time t_0, which are constructed from the third-order Hermite interpolation by

$$\boldsymbol{a}^{(2)} = \frac{-6(\boldsymbol{a}_0 - \boldsymbol{a}_1) - 2\,\Delta t(2\dot{\boldsymbol{a}}_0 + \dot{\boldsymbol{a}}_1)}{\Delta t^2} \tag{9}$$

$$\boldsymbol{a}^{(3)} = \frac{12(\boldsymbol{a}_0 - \boldsymbol{a}_1) + 6\,\Delta t(\dot{\boldsymbol{a}}_0 + \dot{\boldsymbol{a}}_1)}{\Delta t^3}\,, \tag{10}$$

where $\boldsymbol{a}_1$ and $\dot{\boldsymbol{a}}_1$ are the acceleration and its derivative calculated at time $t_0 + \Delta t$ using the predicted position and velocity.

If we neglect the $\boldsymbol{a}^{(3)}$ term in position, we obtain a much simpler form for the corrector (see, e.g., Lambert 1973)

$$\boldsymbol{x}_c = \boldsymbol{x}_0 + \frac{\Delta t}{2}(\boldsymbol{v}_c + \boldsymbol{v}_0) - \frac{\Delta t^2}{12}(\boldsymbol{a}_1 - \boldsymbol{a}_0)\,, \tag{11}$$

$$\boldsymbol{v}_c = \boldsymbol{v}_0 + \frac{\Delta t}{2}(\boldsymbol{a}_1 + \boldsymbol{a}_0) - \frac{\Delta t^2}{12}(\dot{\boldsymbol{a}}_1 - \dot{\boldsymbol{a}}_0)\,. \tag{12}$$

Equation (11) offers essentially the same accuracy as equation (7). The local truncation error of the position of this formula is Δt^5, while that for equation (7) is Δt^6. However, this difference does not have any effect on the global error, since the error in the velocity is Δt^5 for both schemes.

The predictor given in equation (1) uses position, velocity, acceleration, and its derivative at time t. The acceleration and its time derivative are calculated from the position and velocity. Therefore, no information concerning the previous time step is necessary. The corrector also requires information concerning the present and old time steps only. Thus, the fourth-order Hermite scheme is a one-step, self-starting scheme. It is particularly suitable for use with the individual time step scheme.

Makino (1991a) as well as Makino & Aarseth (1992) made detailed comparisons between the Aarseth-type schemes and Hermite schemes of various orders. Their conclusions are summarized as follows. First, the Hermite scheme allows a time step that is roughly twice as long as that of the Aarseth scheme of the same order for the same accuracy. Second, the fourth-order scheme is close to optimal for a wide range of required accuracy. In short, the fourth-order Hermite scheme is better than the Aarseth scheme, in terms of accuracy, efficiency, and simplicity of the algorithm.

2.4. *Hermite Scheme on GRAPE*

Neither the individual time step scheme nor the Hermite scheme can be directly used on GRAPE hardware systems in their original forms, which are designed for shared-time step algorithms. The basic GRAPE architecture must be modified in two ways to run the individual time step scheme with the Hermite integrator. The changes required are the following:

1. The force calculation pipeline must be extended so that it calculates the time derivatives as well.
2. The hardware to calculate the predicted positions and velocities of the particles must be implemented.

Figure 3 shows the hard-wired pipeline to calculate force and its time derivatives. The upper half is the original GRAPE pipeline.

Figure 4 shows the hardware to evaluate the positions of particles at the present time. Equation (1) is used to evaluate these quantities. The hardware to evaluate the velocities is essentially the same.

The complete system with one pipeline is shown in Figure 5. The complete system consists of a control unit, a memory unit, a predictor pipeline, a force calculation pipeline, and the interface to the host. We call this architecture HARP (Hermite AcceleRator Pipeline).

We developed several versions of experimental hardware systems for the individual time step algorithm (GRAPE-2/2A; Ito et al. 1991, 1993) and the Hermite scheme (HARP-1; Kokubo, Makino, & Taiji 1994). We did not implement the predictor pipeline in these systems in order to simplify the design. On these machines, the prediction was performed on the host computer.

On a complete system, the time integration proceeds in the following steps:

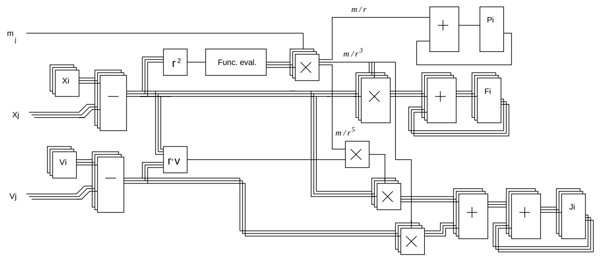

FIG. 3.—Pipeline to calculate the gravitational force and its first time derivative (HARP pipeline).

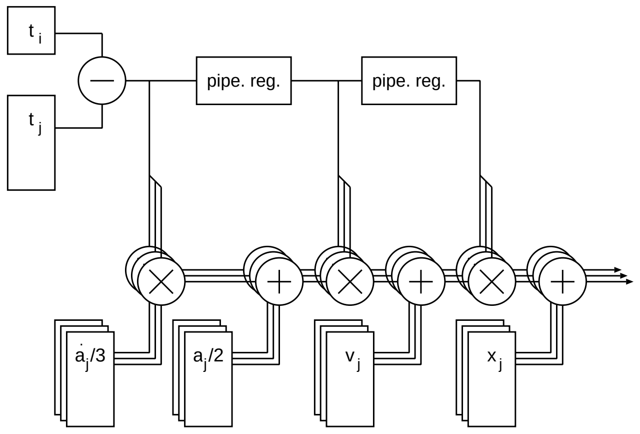

FIG. 4.—Predictor pipeline for position. The pipeline for the velocity looks similar.

1. As the initialization procedure, the host sends all data (position, velocity, acceleration, its first time derivative, mass, and time) of all particles to the HARP memory unit.
2. The host creates the list of particles to be integrated at the present time step.
3. For each particle in the list, repeat steps (4)–(7)
4. The host predicts the position and velocity of the particles and sends them to HARP. HARP stores them in the registers of the force calculation pipeline. It also sets the current time to a register in the predictor pipeline.
5. HARP calculates the force from all other particles. Positions and velocities of other particles at the current time are calculated in the predictor pipeline.
6. After the calculation is finished, the host retrieves the result.
7. The host integrates the orbits of the particles and determines new time steps.
8. Update the present system time and go back to step (2).

Using the present VLSI technology, it is not very difficult to implement the pipelines for force calculation and prediction into a single VLSI chip. In order to achieve a really high performance, however, we have to integrate a number of pipeline units into a single system. In the next section, we describe the parallel architecture of the present GRAPE-4 system.

3. THE GRAPE-4 SYSTEM

3.1. *Architecture*

Figure 6 shows the GRAPE-4 system, and Figure 7 shows its structure. The GRAPE-4 system consists of a host computer and four clusters. One cluster has one host-interface board, one control board, and nine processor boards. The total number of processor boards is thus 36. Each processor board houses 48 HARP (Hermite AcceleRator Pipeline) chips, which are custom LSI chips to cal-

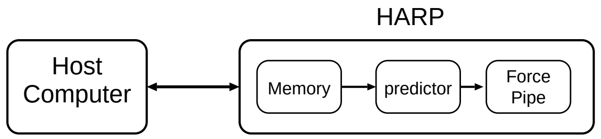

FIG. 5.—Overview of the hardware accelerator for the Hermite scheme

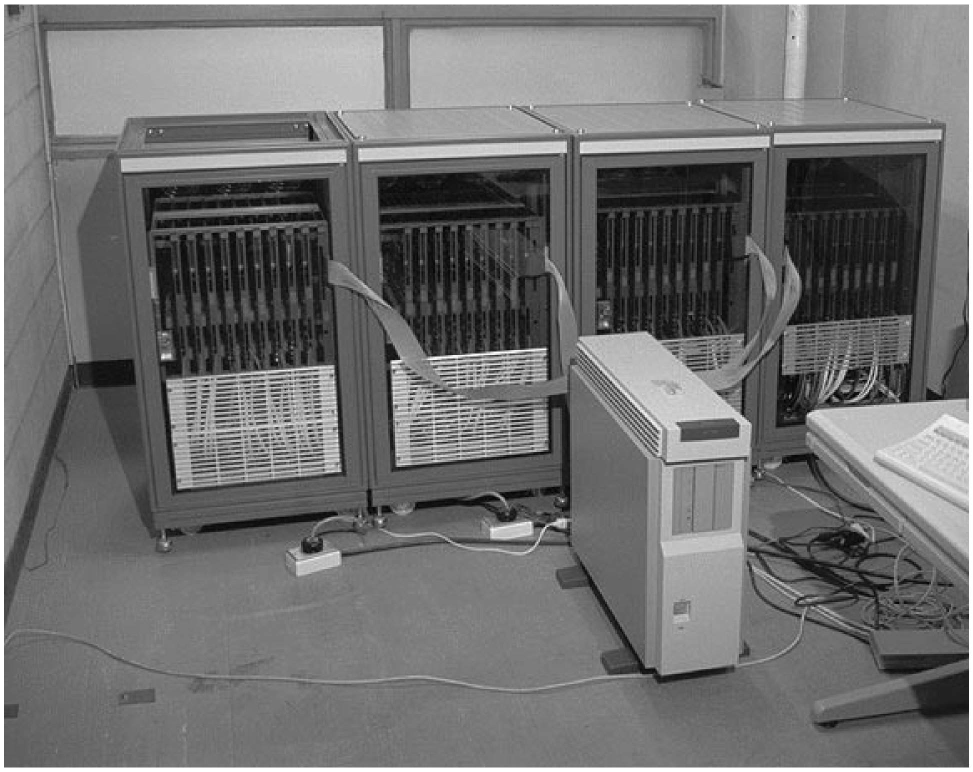

FIG. 6.—GRAPE-4 system

culate the gravitational force and its first time derivative. It also has one PROMETHEUS chip, another custom LSI to calculate the predicted positions and velocities of particles at a given time. The peak speed is 1.08 TFLOPS for the system clock of 16 MHz. In the following, we describe each components in some detail,

3.2. *The Host Computer*

We chose a DEC Alpha AXP workstation as the host computer. Currently, we are using a DEC AXP 3000/900.

We selected the host taking into account the following considerations. First, its scalar computational speed should be reasonably fast. At present, this is achieved by selecting one of the high-end RISC workstations. Second, the I/O bus must be reasonably fast, and the latency of the I/O bus must be small. Some of the midrange server machines have I/O buses with very large transfer rates. On these machines, however, I/O is controlled by a separate I/O processor. As a result, the latency of I/O operations tends to be very long. The DEC Alpha AXP systems have the TURBOchannel

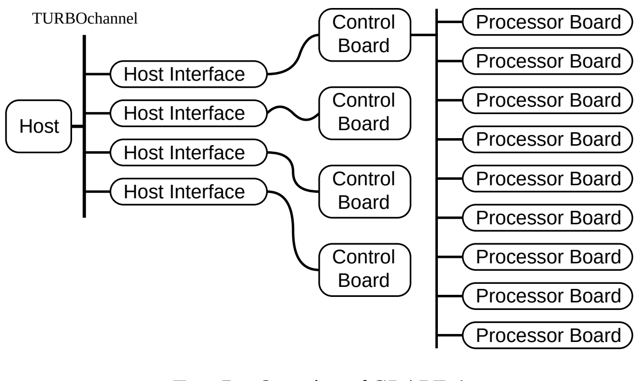

FIG. 7.—Overview of GRAPE-4

with the peak transfer rate of 100 MB s^{-1}, which was pretty good as of 1993. In addition, they do not have separate I/O processors. Therefore, the latency of I/O operations is small. Third, it is desirable that the development of the device driver software be easy, since the available manpower for software development is severely limited. The UNIX operating system from DEC is reasonably stable, and development of the device driver is not very difficult.

The selection of the TURBOchannel caused one practical problem. DEC stopped the development of the machines with TURBOchannel in 1994 and switched to the PCI bus. We foresaw such a problem and designed the total system so that it is relatively easy to connect it to I/O buses different from the TURBOchannel, as described in the next subsection.

3.3. *The Host Interface Board*

The most important function of the host interface board is to extend the I/O bus of the host. In addition, the interface board converts the data transfer protocol of the host I/O bus to the protocol used for the link between the host interface board and the control board. The protocol on the link is designed so that it does not depend on the protocol of the host I/O bus. Because of this structure, we can connect GRAPE-4 to different host computers by changing the host interface board. We are currently developing a new host interface board to the PCI bus, since at present the PCI bus seems to be the most widely available I/O bus.

In this paper, we describe the TURBOchannel interface board. The host interface board and the control board are connected with a 32 bit parallel interface. We adopted a coaxial flat cable for the physical connection between them. The length of the cable is 1 m. The characteristic impedance of the signal lines is 95 Ω. Unidirectional signal wires are terminated with 100 Ω passive terminators, while bidirectional lines are terminated with 200 Ω passive terminators at both ends. This configuration is good enough for the system clock speed up to 20 MHz.

Figure 8 shows the structure of the host interface board. It consists of the data transceivers (trcv) to exchange data with the host and the control board, the FIFO (first-in first-out) unit to buffer the data, and the control logic. The size of FIFO unit is 2048 32 bit words (8 kbyte). The data transfer rate between the host interface unit and the control unit is slower than that between the host and the host interface board. Using the FIFO buffer, the host computer can send the data at the peak transfer speed. In addition, there are several restrictions for the DMA (direct memory access) transfer on the TURBOchannel bus. For example, the number of words to be transferred in a single burst must not exceed 128 words, and the address should not go across a 2 kbyte boundary. The FIFO buffer allows the control board to transfer data without checking the status of the host bus. As a result, the design of the control board becomes independent of the host bus.

The control logic is made of three 22V10 PLD (programmable logic device) chips and one AMD MACH230 complex PLD chip. The AMD MACH chip integrates most of the control logics such as the DMA sequence controller and DMA word address counter. Other PLD chips perform timing-critical handshake operations on both the TURBOchannel and the control-board interface. We used Cypress CY7C453-14 clocked FIFO chips for the data FIFO. This clocked FIFO chip accepts separate clock signals for read and write data paths. The interface to TURBOchannel operates on the TURBOchannel clock, while the interface to the control board operates on the clock signal provided by the processor board. For other data buffer/transceivers, we used the 74FCT series logic chips from Integrated Device Technology.

3.4. *The Control Board*

The control board has two main functions. The first one is to distribute the data received from the host computer to processor boards. The second one is to sum up the force and potential calculated on processor boards. The summed result is transferred to the host computer through the host interface board. This summation is necessary to reduce the bandwidth of the communication with the host computer (Makino, Kokubo, & Taiji 1993).

Each processor board of GRAPE-4 has multiple pipelines that share one memory unit. These pipelines calculate the forces on different particles from the same set of particles, as in the case of GRAPE-3.

There are two different ways to use multiple boards. One is to let all chips calculate the force on different particles from the same set of particles. In this case, the content of the memory of all processor boards would be identical. The other way is to let each processor board calculate the force on the same set of particles, but from different set of particles. In this case, each processor board calculates the partial forces that need to be added with results obtained on other boards.

The former approach is simpler. However, it is not practical in the case of GRAPE-4 with more than 1000 pipelines, since the average number of particles that share the same time is less than 1000 for many cases. Thus, we adopted the latter approach. The main problem with the latter approach is that the amount of communication is proportional to the number of processor boards. If we connect all the processor boards directly to the host computer, the communication would take too much time. In order to solve this problem, we designed the control board so that it can add the results calculated on the processor boards under its control.

The internal structure of the cluster is not visible to the application program. To the host computer, a cluster looks like a board with single memory unit and multiple pipeline chips.

In order to distribute the calculation over different clusters, we use the same algorithm as that used for different processor boards in one cluster. If we have four clusters, the host sends $N/4$ particles to each cluster, where N is the total number of particles that exert the force. In this case, each processor board takes care of the contributions from $N/36$ particles. The host computer adds the partial forces calculated on clusters.

The control board and processor boards are connected by a backplane bus (the HARP bus) with 96 bit data width. A synchronous, pipelined protocol with a fixed latency is used on this HARP bus. For the backplane board of the HARP bus, we used the backplane board of the VME bus. Three VME J1 backplane boards are used to construct a HARP bus. This choice eliminated the need to design the backplane board and card racks. The physical size of the control board (and the processor board) is 366.7 mm by 450 mm.

Figure 9 shows the structure of the control board. It consists of the control logic unit, three accumulator/buffer units, and several transceivers and buffers. The control unit generates all necessary signals to control the HARP bus. The 96 bit data bus is divided into three 32 bit subbuses, each of which is connected to different accumulator/buffer unit. The accumulator/buffer unit contains a 64 bit floating point ALU, which accumulates the result calculated on processor boards.

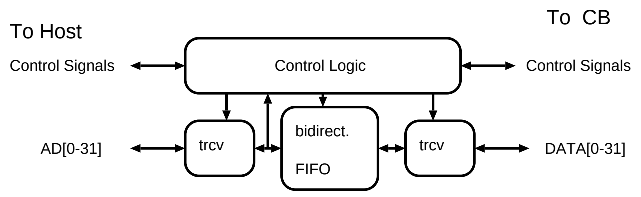

FIG. 8.—Host interface board

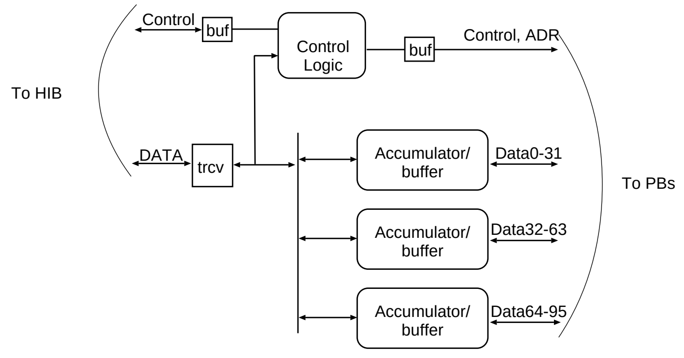

FIG. 9.—Control board

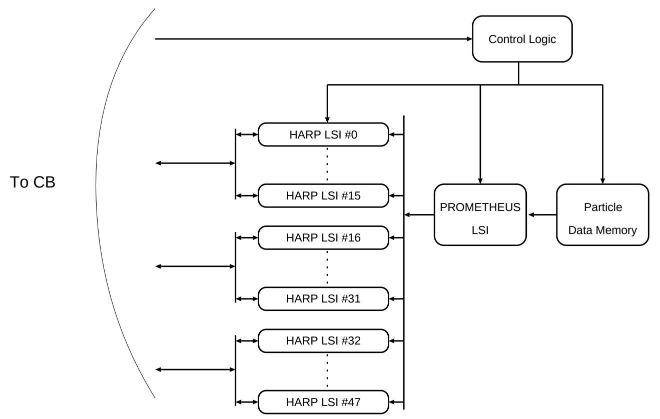

FIG. 10.—Processor board

The control logic unit consists of three AMD MACH230 complex PLDs. The accumulator/buffer unit consists of a TI 74ACT8847 64 bit floating point LSI chip and transceivers, buffers, and FIFO chips.

3.5. *Processor Board*

Figure 10 shows the structure of a processor board. The particle data memory stores the data of particles that exert the force. The PROMETHEUS LSI is used to predict the position (and velocity) of particles at a specified time. The HARP LSI chips calculate the gravitational accelerations and their first time derivatives for particles. One board has 48 HARP chips.

The main control logic is implemented using Altera EP7256 FPGA (field-programmable gate array) chip. For most of interface logics, TI 74ALS series IC chips are used. As will be described in § 3.6, the clock frequency of the HARP chips is twice that for the rest of the system. The base clock signal is generated in the control board and is distributed to all processor boards. Within each board, the clock signals are distributed using PLL (phase-locked loop) clock replicator chips (Cypress 7B991). This chip can multiply the clock frequency and is also used to generate the clock signals for the HARP chip.

3.6. *HARP Chip*

Figure 11 shows the architecture of the HARP chip. It is a fully pipelined hardware implementation of equations (3) and (4). We adopted an architecture in which the x, y, and z components of all vector quantities are processed sequentially, in order to reduce the gate count. Thus, it takes three clock periods to calculate one interaction.

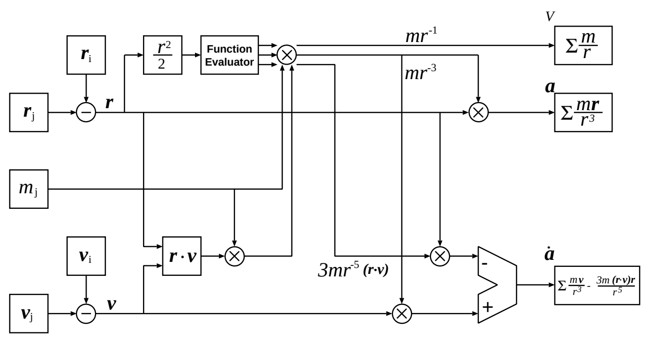

FIG. 11.—HARP chip

Each chip calculates the forces on two particles, using the "virtual multiple pipeline" (VMP). With VMP, the clock period of the pipeline LSI is 2 times that of the system clock, and it calculates the forces on two different particles at alternate clock cycles. From the outside, one force calculation chip looks as if it has two pipelines. The advantage of this architecture is that we can increase the performance of the pipeline chip without increasing the system clock cycle.

This approach is conceptually similar to what is used in "superpipeline" chips such as MIPS R4000 and "clock-doubled" chips such as Intel i486DX2. In our VMP architecture, however, the chip actually has two separate sets of registers, and two virtual pipelines operate independently. On the other hand, in the case of the chips such as i486DX2, the CPU still looks like one CPU. Only the cycle time of the external memory bus is reduced to a fraction of the internal clock cycle. The clock-doubling in the i486DX2 causes some performance penalty, since the relative speed of the main memory becomes slower. The penalty of the cache miss-hit ratio increases significantly. Thus, it is not practical to increase further the ratio between the internal and external clock, unless the width of the memory bus is increased.

It is also possible to compare our VMP with multithreaded architectures such as the Denelcor HEP (Heterogeneous Element Processor; Kowalik 1985) or Tera Computer MTA (multithreaded architecture). The processors of these machines have multiple register sets so that a fast processor looks like a collection of slow processors. Thus, the idea is quite similar to that of VMP. With a multithreaded architecture, however, the data transfer rate of the main memory must still be fast enough to supply the data to a large number of "slow" processors. It is still not very easy to design the memory system.

In the case of VMP, neither of the above problems limits the performance, since all virtual pipelines share the same input data. As a result, it is not very difficult to increase the number of virtual pipelines. In our architecture, it is also easy to have physical multiple pipelines in one chip, when a larger number of gates becomes available. In the present HARP chip, the number of VMPs is two because the external clock speed of 16 MHz is already sufficiently low to make the design of the processor board easy. In addition, a further decrease in the clock cycle of the processor board would cause a decrease in the data transfer rate between the control board and the processor boards.

The number format used in the HARP chip is essentially the same as that used in HARP-1. HARP-1 (Kokubo et al. 1994) is a machine made of off-the-shelf floating point LSI chips. The subtraction of the position vectors and the accumulation of the calculated accelerations are performed in 64 bit floating point format. Other calculations to obtain the acceleration are performed in 32 bit format. The subtraction of velocity vectors and the accumulation of the time derivatives are performed in 38 bit format. Other calculations to obtain the time derivative are performed in 29 bit format, in which the length of mantissa is 20 bits.

The HARP chip is designed using the cell-based method. It is fabricated by the LSI Logic Corporation. The design rule is 1 μm, and the gate count is about 100K. The total number of transistors is about 400K. The die size is 14.2 by 14.2 mm. The worst case clock cycle of the chip is 30 MHz. We started the design of the chip in 1992 summer and completed the design in 1993 spring. Sample chips were delivered in 1993 August, and they had no design failure.

3.7. *MCM Package for HARP Chip*

Eight HARP chips are packaged in an MCM (multichip module) package manufactured by Kyocera. As shown in Figure 10, 16 HARP chips share the same data bus. Therefore, all eight HARP chips in a MCM share the same input data bus and I/O bus. This design is well suited for an MCM package because several difficult problems with MCM packages do not arise.

The use of MCM makes it possible to integrate a large number of HARP chips on a single processor board. As stated earlier, one processor board houses 48 HARP chips (six MCMs). If we use a usual single-chip package, it would be very difficult to integrate more than 16–24 chips to a single board. The total number of processor boards would be around 100, and the manufacturing cost would be about 40% higher.

In general, MCMs are difficult to use because of the following two problems: testability and yield. Each LSI must be tested before actual use. However, the test for an MCM is difficult to design because some of the signals cannot be set/sensed from outside the module. In the case of HARP MCM, there is no such difficulty, since all the I/O pins of HARP chips can be directly accessed from outside. Therefore, the test procedure for an MCM is the same as that for a single chip.

The yield of an MCM tends to be low, since the probability of getting a working MCM is the product of probabilities to get working chips for all chips in the module. For example, if the yield of a single chip is 90%, the yield of an MCM with eight chips would be $(0.9)^8 = 43\%$. We expected the yield of the single chip after DC test to be around 98%. In this case, the yield of the MCM is still higher than 80%. In other words, the yield is not a severe problem. We designed the processor board so that it can accommodate MCMs with defective chips by adding the translation table between the logical chip number and physical chip number. Thus, if each board has (at the maximum) two defective chips, we can use all other 46 chips even if the physical locations of the defects are different on different boards.

The actual yield of MCMs with no defect was around 70%. This is slightly lower than our expectation but good enough to assemble required number of working modules.

3.8. *PROMETHEUS Chip*

Figure 12 gives the architecture of the PROMETHEUS chip. It is a straightforward hard-wired implementation of the predictor equations (1) and (2). The subtraction of time and the addition of x_j and the higher order term are performed in 64 bit format. All other calculations are performed in 32 bit format. The PROMETHEUS chip handles x, y, and z components sequentially in the same way as the HARP chip. Its clock is, however, the same as that of the system clock of the processor board.

The PROMETHEUS chip is a 1 μm CMOS gate array. It is fabricated by LSI Logic Corp. The raw gate count is about 182K. The actual used gates is about 60K. It is packaged into a 391 pin CPGA (ceramic pin-grid array) package.

4. GRAPE-4 SOFTWARE

4.1. *Library Interface*

Table 1 gives the list of subroutines (FORTRAN callable). The routines h3open and h3close acquire/release the right to use the GRAPE-4 hardware. GRAPE-4 is designed as a single-user system, like most of hardware accelerators and attached processors. A simple access control mechanism is required since the operating system of the host computer is multitasking. We used the file-locking mechanism to control the access to GRAPE-4. The interface board is accessed through a device file associated with the board. The function h3open locks this device file when called. If someone else is using GRAPE-4, the process that called h3open is put into the sleep state until the other process releases the GRAPE-4. Requests from multiple processes are automatically queued by the file-locking mechanism of the operating system.

The function h3npipe returns the number of pipelines available on the hardware, and h3setti sets the present time (t_i) for the predictor unit.

The function h3mjpdma_indirect sends the particle data to the memory unit of GRAPE-4. This function updates the memory data of particles specified by the index array. It accesses both the memory of the host computer and that of GRAPE-4 indirectly in order to minimize the number of copy operations within the host main memory.

The function h3calc actually lets GRAPE-4 calculate the force. It receives the positions, velocities, softenings, and the neighbor sphere radii of the particles in order to calculate the accelerations and their time derivatives and returns them and potentials as well.

The function h3calc lets the GRAPE-4 calculate the force and waits until GRAPE-4 finishes the calculation. Thus, the host computer cannot perform useful work while GRAPE-4 is working. We implemented an additional set of functions that allows the concurrent computation of the host and GRAPE-4. The function h3calc_firsthalf initiates the calculation on GRAPE-4 and returns immediately, and h3calc_lasthalf retrieves the calculated result. Thus, the host

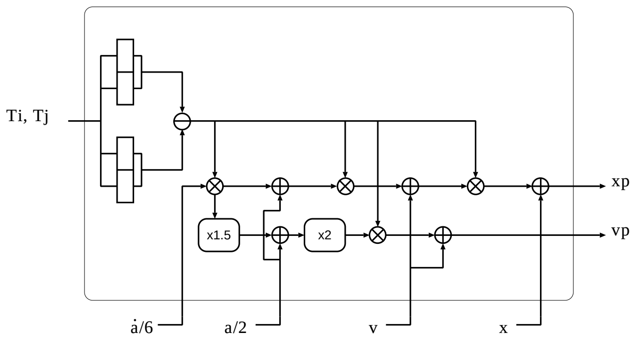

FIG. 12.—PROMETHEUS chip

TABLE 1
GRAPE-4 INTERFACE ROUTINES

Name	Function
h3open....................	Acquire GRAPE-4
h3close....................	Release GRAPE-4
h3npipe...................	Return the number of pipelines
h3setti....................	Store the current time
h3mjpdma_indirect......	Send particle data to memory
h3calc.....................	Let GRAPE-4 calculate force
h3calc_firsthalf...........	Let GRAPE-4 calculate force and return
h3calc_lasthalf...........	Retrieve calculated force from GRAPE-4

computer can do useful work while GRAPE-4 calculates the force.

There are several other functions that retrieve the neighbor list for each particle.

Beside the bookkeeping functions, there are only two functions to be called. One is h3mjpdma_indirect that stores the data to the memory, and the other is h3calc that lets the GRAPE-4 calculate the force and retrieve the result.

4.2. *Implementation of the Library Functions*

As stated in § 3.3, the communication between the present host interface board and the host computer relies on the DMA transfer, which makes the implementation of the communication software somewhat more complicated than the simple direct access described in Makino & Funato (1993). All GRAPE hardware systems other than GRAPE-1 and GRAPE-4 operate as a VME-bus slave. This means that it looks like a memory card on the I/O bus to the host computer. The virtual memory system of the UNIX operating system allows the application program to read/write directly the address space assigned to GRAPE through memory mapping. Thus, once the mapping is accomplished, the application program accesses the GRAPE hardware without any intervention from the operating system.

When DMA is used, the communication becomes more complicated. The traditional way to implement the data transfer using DMA in a UNIX operating system is the following. In the case of a "read" operation, the user process requests the operating system kernel to transfer data from a device. Then the operating system issues the command to the device to write the data to the buffer within the kernel address space. After the data transfer by the device is completed, the kernel copies the data in its buffer to the address space of the user process and returns the control to the user process. This process has two performance problems. First, the overhead of transferring the control between the user process and the kernel process is very large. Second, the time spent to copy the data between the kernel buffer and user memory is considerable. In fact, on the DEC Alpha workstations that we used, the data transfer through DMA can achieve a speed of more than 90 MB s^{-1} quite easily, while the throughput of the copy within the main memory is around 50 MB s^{-1}. Thus, a single copy operation by the host CPU effectively reduces the data transfer speed by a factor of 3.

In order to avoid these overheads, we designed the interface library so that the application process can invoke the DMA transfer to/from its memory space directly. In order to implement this, the user process needs to know the physical address of the data area in its virtual address space. A special system call was developed for this purpose. This solution, however, is not perfectly reliable, since the operating system might change the physical address without notifying the user process when a page fault occurs.

5. PERFORMANCE

In this section, we present the measured speed of the GRAPE-4 system.

To measure the performance, we performed test runs on various configurations. As the initial conditions, we used the King profiles with nondimensional central potential W_c of 3, 7, and 10. We changed the number of particles from 128 to 524,288, and we measured the speed for the configurations with one, two, and three clusters. The fourth cluster was unavailable at the time of experiments.

The system of units is chosen so that the total mass of the system M and the gravitational constant G are both unity. The total energy of the system E is $-\frac{1}{4}$ (Heggie & Mathieu 1986). The softening parameter is $1/N$, where N is the number of particles. The mass of all particles is $m = 1/N$. We integrated the system for one time unit and measured the CPU time on the host. The host computer was a DEC Alpha AXP 3000/900 with a 448 MB memory.

Figure 13 shows the calculation speed of GRAPE-4 in GFLOPS and the actual CPU time per unit time for the runs from King models with $W_c = 3$. The achieved speed is roughly proportional to the number of particles for $10^3 < N < 10^5$ and is almost independent of the number of clusters. In this range, the use of more than one cluster actually decreases the overall speed, since the total per-

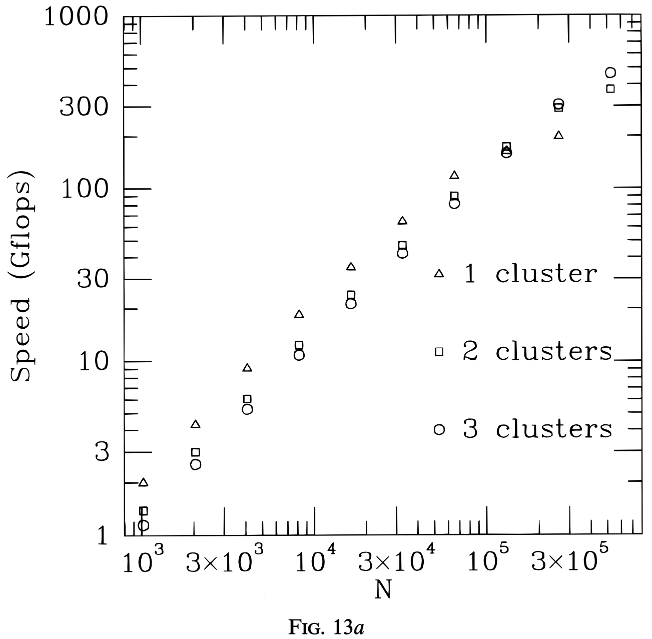

FIG. 13*a*

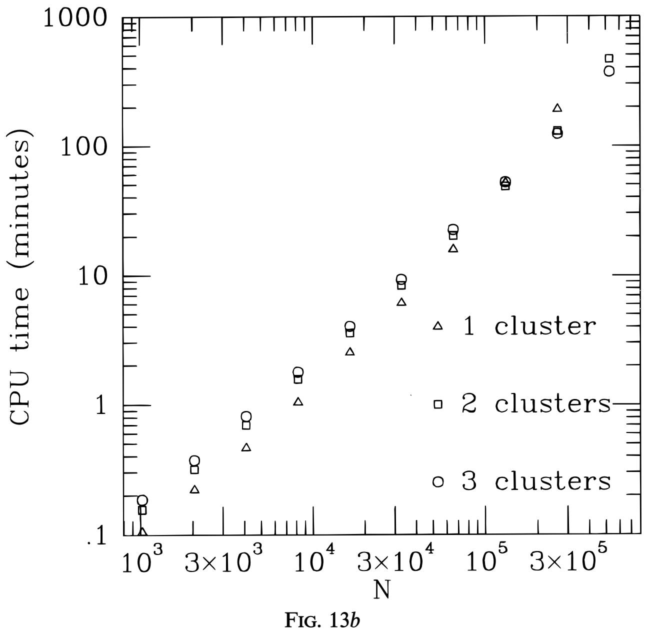

FIG. 13*b*

FIG. 13.—(*a*) Calculation speed of GRAPE-4 system in GFLOPS and (*b*) CPU time per time unit plotted as a function of the number of particles into the system. Initial particle distribution is a King model with $W_c = 3$.

formance of the system is limited by the speed of the host computer. As described in § 3.4, the amount of the communication increases as the number of clusters increases. For 256K particle runs, two- and three-cluster configurations are actually faster than the one-cluster configuration. For 512K particle runs, the three-cluster configuration is the fastest. The CPU time per time unit is about 8 hr for a 512K particle run on the three-cluster system. Thus, such a simulation (Fukushige & Makino 1996) is feasible at least for the crossing time scale. For the relaxation time scale, simulations with a particle number less than 10^5 have been performed (Makino 1996a).

Figure 14 shows the speed of the three-cluster configuration for various initial models. For small-N runs, the speed is lower for a higher central concentration. For large-N runs, the difference is very small. This difference is due to the difference in the average number of particles, n_s, sharing the same time that is shown in Figure 15. If this n_s is smaller than the number of force calculation pipelines, the performance of GRAPE-4 becomes lower since some of the pipelines are not used. The number of pipelines is 94 in GRAPE-4.

Figure 15 shows that n_s is quite well approximated by $(N)^{1/2}$, though it depends somewhat on the structure of the cluster. This dependence is different from the theoretical model (Makino 1991b), which predicts

$$n_s \propto N^{2/3} . \tag{13}$$

The disagreement between the theoretical prediction and the experimental result for n_s is caused by the difference in the average number of time steps $s_{\rm avg}$. The theoretical model is derived using the assumption that the average number of the time steps per particle per time unit, $s_{\rm avg}$, should satisfy

$$s_{\rm avg} \propto N^{1/3} , \tag{14}$$

and the number of the system time steps per time unit (the time average of the inverse of the minimum time step), s_b, is $O(N^{2/3})$. Figure 16 shows $s_{\rm avg}$ as the function of the number of particles N. The behavior of $s_{\rm avg}$ is systematically different

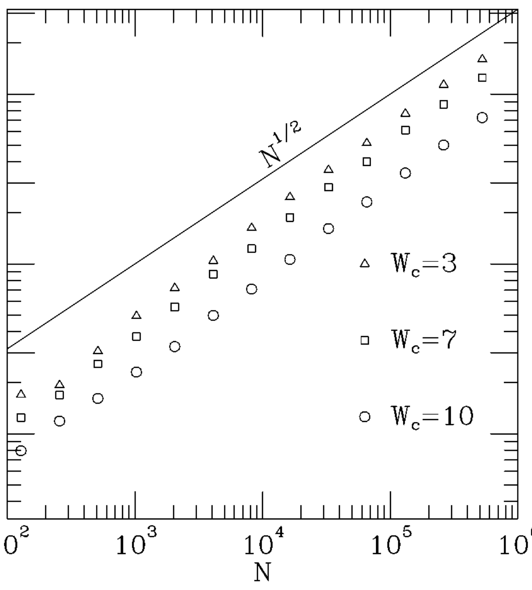

FIG. 15.—Number of system time updates per unit time, n_s, plotted as a function of the number of particles N.

from the theoretical prediction. The formula

$$s_{\rm avg} \propto N^{1/6} \tag{15}$$

seems to be a good fit. The experimental result of equation (15) is a direct consequence of the fact that we used the time step criterion based on the time derivatives of the acceleration. Any criterion that is expressed as a nondimensional function of the acceleration and its time derivatives shows a dependence of the form of equation (15) (Makino & Hut 1988). On the other hand, equation (14) is derived from the assumption that the time step should be proportional to the average interparticle distance. In other words, the time step criterion we have been using might fail to resolve close encounters for very large N. For the range of N we tested, the difference between $N^{1/3}$ and $N^{1/6}$ is only a factor of 4. Therefore, it is rather unlikely that the close encounters are not resolved properly.

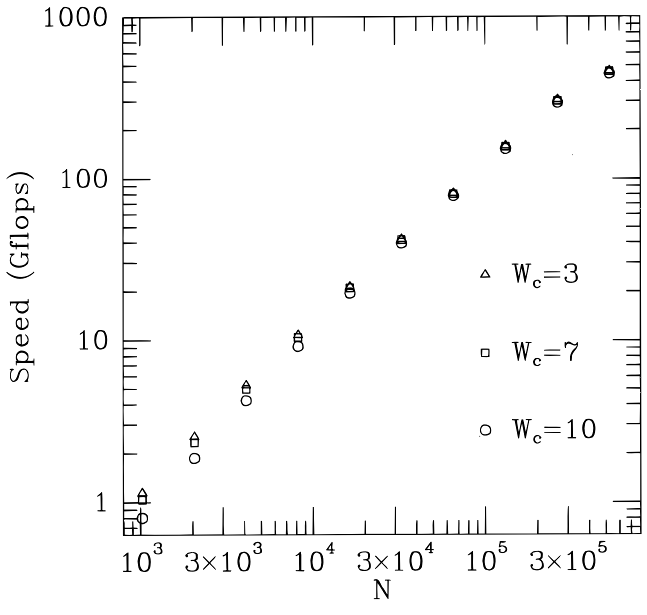

FIG. 14.—Same as Fig. 13 but for different initial models

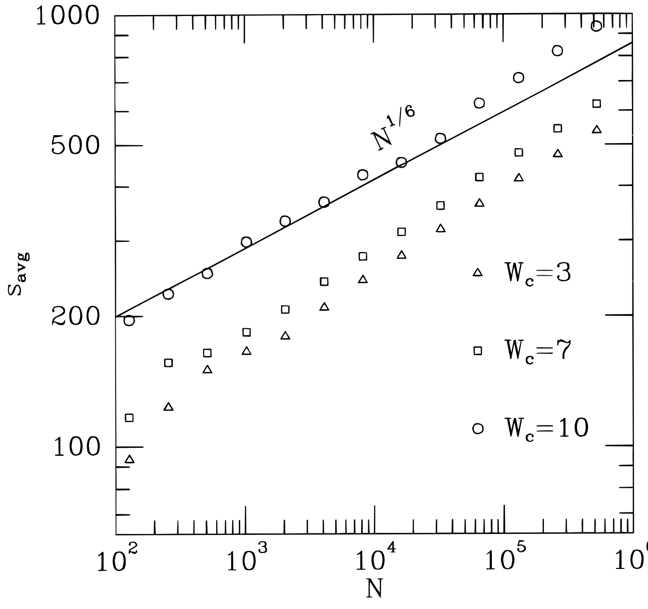

FIG. 16.—Average number of time steps per particle per unit time plotted as a function of the total number of particles N.

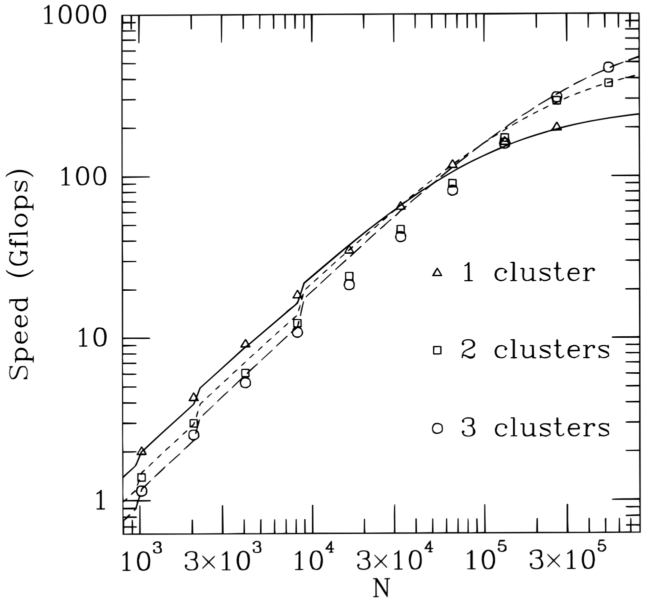

FIG. 17.—Actual and estimated performance of GRAPE-4 system plotted as a function of total number of particles N.

Conceptually, the time to integrate one particle on GRAPE-4 system is given as follows:

$$T_{\rm step} = T_{\rm host} + T_{\rm grape} + T_{\rm comm} , \qquad (16)$$

where $T_{\rm host}$, $T_{\rm grape}$, and $T_{\rm comm}$ are the time to integrate the orbit of one particle on host, the time to calculate the force on one particle on GRAPE-4, and the time to transfer data between the host and GRAPE-4, respectively. In reality, the actual time can be made shorter than this estimate because the calculations on GRAPE and that on the host partially overlap.

The three terms in the right-hand side of equation (16) are expressed as

$$T_{\rm host} = 250 t_{\rm host} , \qquad (17)$$

$$T_{\rm grape} = \gamma \frac{3N t_{\rm pipe}}{47 n_{\rm cl} n_{\rm pr}} , \qquad (18)$$

$$T_{\rm comm} = (34 + 20\gamma n_{\rm cl}) t_{\rm comm1} . \qquad (19)$$

Here, $t_{\rm host}$ is the average time for the host computer to perform one floating point operation. For the Hermite scheme, the total number of floating point operations per particle per time step is 200–300. The actual speed of the host computer for the part of the code executed on the host is rather low, 15–20 MFLOPS. We thus estimate that $t_{\rm host} = 5 \times 10^{-8}$ s. In equation (18), N is the number of particles in the system, $t_{\rm pipe}$ is the cycle time of the HARP chip, and $n_{\rm cl}$ and $n_{\rm pr}$ denote the number of clusters and the number of processor boards per cluster. The parameter γ is the loss of parallel efficiency of multiple pipelines, which is estimated as

$$\gamma = \left[\frac{n_s + n_{\rm vp} - 1}{n_s} \right] . \qquad (20)$$

Here, $[x]$ denotes the maximum integer that does not exceed x, and $n_{\rm vp}$ is the number of virtual pipelines. In real GRAPE-4, $n_{\rm vp} = 94$. For present GRAPE-4, the clock cycle of HARP chips is 32 MHz, and the number of processor boards per cluster is nine, i.e., $t_{\rm pipe} = 3.125 \times 10^{-8}$ s and $n_{\rm pr} = 9$. In equation (19), $t_{\rm comm1}$ is the time to transfer one word (4 bytes) between the host and a control board. The number of words to be transferred between the host and a control board is $34 + 20n_{\rm cl}$, in the present implementation of the control board.

For the communication time, we used the value $t_{\rm comm1} = 1.8 \times 10^{-7}$ s. The raw speed of the DMA on the TURBO-channel interface is close to 100 MB s^{-1}, which is translated to $t_{\rm comm1} = 4 \times 10^{-8}$ s. The actual time spent for the communication is significantly larger simply because the data copying within the main memory takes a rather long time as is discussed in § 4.2. The DMA function of HIB can directly access the user address space. Even so, the user process need to pack/unpack the data because a single DMA operation can transfer only a block of memory.

We can see from Figure 17 that the theoretical performance model agrees well with the experimental result for both small- and large-N but tends to overestimate the speed for intermediate values of N. Most likely, this is because our estimate for γ is too crude.

The actual performance is quite notably lower than our original assessment (Makino et al. 1994). This is mainly because we underestimated $t_{\rm comm1}$. In our original assessment, we neglected the time to move data within the main memory.

6. FUTURE PROSPECTS

6.1. *Planned Improvements*

The GRAPE-4 hardware achieved the planned peak speed of 1 TFLOPS. However, the sustained performance for small-N simulations is somewhat less than expected. This is simply because the speed of the host computer we are currently using is lower than our expectation of early 1994. In 1994, we hoped to have the host computer about 3 times faster than what we had then.

Since the performance of the general-purpose workstation doubles every 18 months, our expectation was reasonable. The actual reason the expected performance is not achieved is that most recent workstations cannot be connected to the present GRAPE-4 because the interface is different.

As mentioned in § 3.3, we are currently developing a new host interface board for the PCI bus (Kawai, Fukushige, & Makino 1997). This new interface board will allow us to connect GRAPE-4 to a wide range of computers, from Intel-based PCs to big servers from DEC, SGI, or HP.

As is discussed in the previous section, the bottleneck of the total performance of the current GRAPE-4 system is the bandwidth of the main memory of the host computer, not really the calculation speed of the CPU of the host computer. Thus, in order to improve the performance, we should chose a machine with high-memory bandwidth. Unfortunately, present UNIX workstations provide rather poor main memory performance, if we compare it with the speed of the CPU (McCalpin 1995). Traditional vector processors seem to have a better balance between the main memory bandwidth and the speed of the processor. Unfortunately, vector processors are quite difficult to use as the host computer of GRAPE systems because they almost always have rather complicated systems for handling I/O. Thus, it is difficult to meet the requirement of I/O with very low latency.

For the near future, we plan to use an SMP (shared-memory multiprocessor) box with multiple PCI interfaces as the host. This should give us at least a factor of 4 improvement in the speed of I/O, which means that 50% of the peak speed will be achieved for less than 10^5 particles. In 2 or 3 years, we will probably use a cluster of personal computers such as the Beowulf (Becker et al. 1995) or PAPERS as the host computer. An Intel-based personal computer currently offers a main memory bandwidth comparable to that of high-end RISC workstations for a fraction of the price. A cluster of such machines would be the natural choice, if the communication between processors is reasonably fast.

It might be rather difficult to develop a parallelized simulation program. In practice, we can achieve good throughput even if a single simulation program is not well parallelized, because we can simply use multiple hosts (or multiple CPUs on a single host) to run separate programs.

6.2. *Advantage of GRAPE Architecture*

The GRAPE-4 hardware achieved the speed of 1 TFLOPS for a total cost less than $2,000,000. Compared to the speed of general-purpose computers of a similar price, this is faster by at least 1 order of magnitude. The actual difference is even larger, since GRAPE-4 can actually achieve a sizable fraction of the theoretical peak performance for real applications. On the other hand, the actual performance of many general-purpose computers on real applications is much lower than the theoretical peak.

The essential reason that GRAPE-4 has a price-performance ratio so much better than that of general-purpose computers is that the general-purpose computers do not make full use of the available resources. GRAPE-4 integrates about 35,000 floating point arithmetic units. On the other hand, the largest number of floating point units ever integrated on a single general-purpose machine is around 4000. With GRAPE-4, it was relatively easy to integrate a number of arithmetic units for the following reasons. First, because of the pipelined architecture, a number of floating point units can be integrated into a single chip with a relatively small number of I/O pins. Second, because of the nature of the problem, a number of these pipelines can be connected to a single memory unit. Third, since the problem is computation intensive, the amount of memory required for the machine is quite small.

These three characteristics allow us to use almost all transistors available on a VLSI chip for floating point units. In both the GRAPE and HARP chips, practically all the transistors are used for arithmetic units. The HARP chip integrates about 20 arithmetic units on the chip with 4×10^5 transistors. A 64 bit floating point multiplier needs about 10^5 transistors. The HARP chip could integrate 20 arithmetic units because we adopted a 32 bit (and in some parts a 29 bit) format for multipliers. This choice does not affect the accuracy of the final result.

A very small fraction of the transistors is used for the arithmetic units in present microprocessors. A typical microprocessor of 1996 integrates 5–10 million transistors, which is more than a factor of 10 larger than that of the HARP chip. Even so, most of these chips have just one multiplier and one adder. Some chips have two adders and two multipliers. No chip has more than four arithmetic units. Thus, the arithmetic units consume less than 10% of the total silicon available on the chip. This is because none of the three characteristics of GRAPE architecture is shared by general-purpose microprocessors. In a programmable computer, if we have multiple floating point units, the bandwidth of the memory must be increased accordingly. This is true for both the multiple units within a chip and multiple processor chips. Thus, a large fraction of the total cost of a general-purpose parallel computer is spent for the interconnection between the memory and processors.

The silicon VLSI technology is expected to advance continuously for at least next 15 years. This advancement implies that the available number of transistors would quadruple every 3 years, and the clock speed would double in the same period (the latter might be a bit too optimistic, though).

GRAPE architecture can take full advantage of the increase in the number of transistors and the reduction in the clock cycle. This means that we can improve the performance of GRAPE by a factor of 8 every 3 years or by a factor of 1000 in a decade. On the other hand, the performance of general-purpose computers has been improved at a rate of a factor of 100 per decade, and this rate is expected to remain constant (Sterling, Messina, & Smith 1995) because of diminishing efficiency. Thus, GRAPE will be more cost-effective in the future.

Recently, the idea of using field-programmable gate arrays (FPGAs) as custom computing engines has become popular (see, e.g., Buell, Arnold, & Kleinfelder 1996). A FPGA chip consists of logic elements and interconnections that can be programmed electrically. The FPGA has the important advantage over custom LSI chips that the development cost of the hardware is small. The development cost of a custom VLSI chip is somewhere between $40,000 and $400,000, depending on various factors such as company, technology, size of the chip, and so on. For example, we paid $250,000 for the design and samples of HARP chips. Using public domain design software and the service offered by MOSIS, one could develop a similar chip for a total cost of about $50,000. On the other hand, the investment necessary to acquire the programming tools for a FPGA is less than $10,000, and one can develop a number of different chips with this tool.

For small experimental projects, FPGAs are handy and quite useful because the initial investment is small. In addition, the flexibility offered by FPGAs might be useful.

If the total budget is large enough to allow the design and development of a custom chip, it is almost always more cost-effective than the implementation using FPGA. There is a big difference between the amount of the circuit that can be integrated in an FPGA and that can be integrated in a custom LSI. As of 1996 summer, it is not impossible to integrate about 5×10^6 gates on a single custom chip; on the other hand, the number of gates on the most advanced FPGAs is around 5×10^4. Thus, there is a difference of about 2 orders of magnitude. In addition, the clock speed of a custom chip would be significantly faster than that of an FPGA chip. Therefore, a single custom chip can provide a performance level several hundred times greater than that of an FPGA.

6.3. *Next-Generation Hardware*

To illustrate the relative advantage of GRAPE systems, let us outline what a next-generation machine would look like. In the following, we assume using the technology that will be available in 1998–1999, which will be 0.25 μm technology. With this level of technology, one can pack 2×10^7 transistors, or about 5 M gates, into a single chip. The clock speed of 150 MHz will not be very difficult. Thus, the number of transistors is 50 times larger than that of a HARP chip, and the clock frequency is 5 time larger. A single chip should provide about 150 GFLOPS.

If we construct a machine with 1400 chips, it will have a peak speed of 200 TFLOPS. The total cost and the overall architecture will not be much different from that of current GRAPE-4, since the total number of pipeline chips is similar. Such a machine can be completed by year 2000 for a total cost of about $2,000,000. For a total cost of $10,000, 000 the speed of the machine can be increased to 1 PFLOPS.

A machine with a speed in the range of 100 TFLOPS to 1 PFLOPS will allow us to study, for example, the evolution of globular clusters with up to 10^6 particles and the dynamics of galactic cores containing more than 10^7 particles. The former would allow us to study various aspects of the evolution of real globular clusters. The latter would make it possible to follow the evolution of, for example, the central black holes with necessary accuracy.

In practice, the most important outcome of the next-generation project is not a few gigantic calculations but rather the possibility of producing a number of copies of a multi-TFLOPS system. A single next-generation chip will deliver 150 GFLOPS. Thus, it will be relatively easy to construct a board with 5–15 chips, which will have a peak speed of 1–2 TFLOPS. Such a board will cost $10,000–$20, 000 and can be hooked to a workstations to instantly change it to a supercomputer.

6.4. *Other Applications*

The present GRAPE-4 is specialized for gravitational N-body simulations with $1/r$ potential. However, the basic idea of the GRAPE system can be applied to any particle-based simulations. In fact, a few machines for molecular dynamics simulations had been developed before we started GRAPE project (Bakker & Bruin 1988; Fine, Dimmler, & Levinthal 1991). In these machines, the force law was programmable using a lookup table and polynomial interpolation.

We also have developed GRAPE-2A (Ito et al. 1993) and MD-GRAPE (Taiji et al. 1994). Using the programmable force table, they can run P^3M and the Ewald method, which are used for cosmological simulations with periodic boundary conditions (Fukushige et al. 1996).

Another promising application is SPH. In fact, currently, a large number of GRAPE-3A hardware systems are used for SPH simulations (Umemura et al. 1993; Steinmetz 1996). In SPH simulations, the GRAPE-3A hardware is used to calculate the gravitational interactions and construct the list of neighbor particles. The host computer uses this list to evaluate the SPH interactions. The number of neighbors for which the SPH interaction is calculated is typically 30–50. Therefore, the cost of the SPH interaction is essentially $O(N)$. Even so, the calculation cost is relatively large, in particular when GRAPE is used.

Since the calculation of the SPH interaction is rather similar to that performed in GRAPE, there have been a few attempts to build a hard-wired pipeline for SPH calculations using a design similar to that of GRAPE. No hardware has been completed yet, but one system is supposed to be completed soon (Yokono 1996).

There are several technical problems with a hardware specialized for SPH. The actual gain that could achieved is largely unknown, since there is no detailed analysis of the behavior of the calculation cost of SPH simulations yet. If one particle actually interacts with only 30–50 neighbors, the gain one can achieve with an SPH hardware would be rather limited.

In practice, however, there are two different factors that seem to suggest that the gain is larger. First, most of the present SPH implementations for cosmological simulations pose a lower limit to the kernel size h. This is necessary to keep the minimum time step from becoming too small. However, if such a limit is used, the calculation cost of SPH interaction is likely to be dominated by the particles with this minimum h, since they have the largest local densities (and therefore largest number of neighbors) and smallest time steps. Therefore, it is quite likely that the calculation cost of SPH interaction is no longer $O(N)$.

Second, with the present implementation of the individual (hierarchical) time step, most existing calculation codes perform $O(N)$ calculation (prediction of physical quantities of all the particles) at each system step, no matter how small the number of particles to be updated at that system step. In this case, if the prediction is performed on the specialized hardware, the gain in the speed can be very large. It should be noted, however, that it is in theory possible to avoid this prediction altogether (Makino & Funato 1993).

To summarize, the gain that can be achieved by SPH hardware seems to be larger than the naive estimate, and it needs to be studied more carefully. This is an ideal project for an experimental hardware using FPGA.

We thank Sverre Aarseth, who made the NBODY1 program available, and Toshiyuki Fukushige for help in developing part of the GRAPE-4 hardware and comments on the manuscript. We thank F. Summers, C. Frenk, and D. C. Heggie for discussions on SPH. This work was supported by Grant-in-Aid for Specially Promoted Research (04102002) of the Ministry of Education, Science, Sports and Culture, Japan.

REFERENCES

Aarseth, S. J. 1963, MNRAS, 126, 223
———. 1985, in Multiple Time Scales, ed. J. U. Brackhill & B. I. Cohen (New York: Academic), 377
———. 1996, in IAU Symp. 174, Dynamical Evolution of Star Clusters, ed. P. Hut & J. Makino (Dordrecht: Kluwer), 161
Bakker, A. F., & Bruin, C. 1988, in Special Purpose Computers, ed. B. J. Alder (San Diego: Academic), 183
Barnes, J. E. 1986, in The Use of Supercomputers in Stellar Dynamics, ed. S. McMillan & P. Hut (New York: Springer), 175
———. 1990, J. Comp. Phys., 87, 161
Barnes, J. E., & Hut, P. 1986, Nature, 324, 446
Brieu, P. P., Summers, F. J., & Ostriker, J. P. 1995, ApJ, 453, 566
Buell, D. A., Arnold, J. M., & Kleinfelder, W. J. 1996, Splash 2: FPGAs in a Custom Computing Machine (Los Alamitos: IEEE Computer Society Press)
Chikada, Y., et al. 1987, Proc. IEEE, 75, 1203
Couchman, H. M. P., Thomas, P. A., & Pearce, F. R. 1995, ApJ, 452, 797
Fine, R., Dimmler, G., & Levinthal, C. 1991, in Proteins: Structure, Function, and Genetics, 11, 242
Fukushige, T., Ito, T., Makino, J., Ebisuzaki, T., Sugimoto, D., & Umemura, M. 1991, PASJ, 43, 841
Fukushige, T., & Makino, J. 1997, ApJ, in press
Fukushige, T., Taiji, M., Makino, J., Ebisuzaki, T., & Sugimoto, D. 1996, ApJ, 468, 51
Heggie, D. C., & Mathieu, R. D. 1986, in The Use of Supercomputers in Stellar Dynamics, ed. S. McMillan & P. Hut (New York: Springer), 233
Hernquist, L. 1990, J. Comp. Phys., 87, 137
Hut, P., Makino, J., & McMillan, S. L. W. 1988, Nature 336, 31
Hut, P., et al. 1992, PASP, 104, 681
Ito, T., Ebisuzaki, T., Makino, J., & Sugimoto, D. 1991, PASJ, 43, 547

Ito, T., Makino, J., Ebisuzaki, T., & Sugimoto, D. 1990, Comput. Phys. Commun., 60, 187
Ito, T., Makino, J., Fukushige, T., Ebisuzaki, T., Okumura, S. K., & Sugimoto, D. 1993, PASJ, 45, 339
Kawai, A., Fukushige, T., & Makino, J. 1997, in preparation
Kokubo, E., & Ida, S. 1996a, Icarus, 123, 180
———. 1996b, Icarus, submitted
Kokubo, E., Makino, J., & Taiji, M. 1994, in Proc. of the 27th Annual Hawaii International Conference on System Sciences, ed. T. N. Mudge & B. D. Shriver (Los Alamitos: IEEE Computer Society Press), 1, 292
Kowalik, J. S., ed. 1985, Parallel MIMD Computation: The HEP Supercomputer and Its Applications (Boston: MIT Press)
Krogh, F. T. 1974, in Proc. of the Conf. on the Numerical Solution of Ordinary Differential Equations, ed. D. G. Bettis (New York: Springer), 22
Lambert, J. D. 1973, Computational Methods in Ordinary Differential Equations (New York: John Wiley and Sons)
Makino, J. 1990, J. Comp. Phys., 87, 148
———. 1991a, ApJ, 369, 200
———. 1991b, PASJ, 43, 841
———. 1996, ApJ 471, 796
———. 1997, ApJ, in press
Makino, J., & Aarseth, S. J. 1992, PASJ, 44, 141
Makino, J., & Ebisuzaki, T. 1996, ApJ, 465, 527
Makino, J., & Funato, F. 1993, PASJ, 1993, 45, 279
Makino, J., & Hut, P. 1988, ApJS, 68, 833
Makino, J., Kokubo, E., & Taiji, M. 1993, PASJ, 45, 349.
Makino, J., Taiji, M., Ebisuzaki, T., & Sugimoto, D. 1994, in Proc. of Supercomputing '94 (Los Alamitos: IEEE Computer Press), 429
McCalpin, J. D. 1995, Newsletter of Technical Committee on Computer Architecture, IEEE Computer Society, December, 19
McMillan, S. L. W. 1986, in The Use of Supercomputers in Stellar Dynamics, ed. S. McMillan & P. Hut (New York: Springer), 156
McMillan, S. L. W., & Aarseth, S. J. 1993, ApJ, 414, 200
McMillan, S. L. W., Hut, P., & Makino, J. 1990, ApJ, 362, 522
Okumura, S. K., Makino, J., Ebisuzaki, T., Fukushige, T., Ito, T., & Sugimoto, D. 1993, PASJ, 45, 329
Press, W. H., & Spergel, D. N. 1988, ApJ, 325, 715
Robinson, C., Lyne, A. G., Manchester, R. N., Bailes, M., D'Amico, N., & Johnston, S. 1995, MNRAS, 274, 547
Salmon, J., & Warren, M. 1994, J. Comp. Phys., 111, 136
Steinmetz, M. 1996, MNRAS, 278, 1005
Sterling, T., Becker, D. J., Savarese, D., Fryxell, B., & Olson, K. 1995, in Proc. of the 1995 International Conference on Parallel Processing, ed. T.-Y. Feng (Boca Raton: CRC), 1, 11
Sterling, T., Messina, P., & Smith, P. H. 1995, Enabling Technologies for Petaflops Computing (Cambridge, MA: MIT Press)
Sugimoto, D., & Bettwieser, E. 1983, MNRAS, 204, 19P
Taiji, M., Fukushige, T., Makino, J., Ebisuzaki, T., & Sugimoto, D. 1994, in Proc. of the 6th Joint EPS-APS Int. Conf. on Physics Computing, ed. R. Gruber & M. Tomassini (Geneva: European Physical Society), 67
Umemura, M., Fukushige, T., Makino, J., Ebisuzaki, T., Sugimoto, D., Turner, E. L., & Loeb, A. 1993, PASJ, 45, 311
Warren, M. S., Quinn, P. J., Salmon, J. K., & Zurek, W. H. 1992, ApJ, 399, 405
Weinberg, M. D. 1994a, AJ, 108, 1398
———. 1994b, AJ, 108, 1403
———. 1994c, AJ, 108, 1414
Yokono, Y. 1996, in Proc. of Workshop on Numerical Astrophysics Using Supercomputers, ed. K. Tomisaka (Tokyo: National Astronomical Observatory of Japan), 36