Performance evaluation and tuning of GRAPE-6 — towards 40 "real" Tflops

Junichiro Makino
Department of Astronomy, School of Science, University of Tokyo, Tokyo 113-0033, Japan
makino@astron.s.u-tokyo.ac.jp

Eiichiro Kokubo
National Astronomical Observatory of Japan, Mitaka, Tokyo 181-8588, Japan

Toshiyuki Fukushige
General System Studies, College of Arts and Sciences, University of Tokyo, Tokyo 153-8902, Japan

ABSTRACT

In this paper, we describe the performance characteristics of GRAPE-6, the sixth-generation special-purpose computer for gravitational many-body problems. GRAPE-6 consists of 2048 custom pipeline chips, each of which integrates six pipeline processors specialized for the calculation of gravitational interaction between particles. The GRAPE hardware performs the evaluation of the interaction. The frontend processors perform all other operations, such as the time integration of the orbits of particles, I/O, on-the-fly analysis etc. The theoretical peak speed of GRAPE-6 is 63.4 Tflops. We present the result of benchmark runs, and discuss the performance characteristics. We also present the measured performance for a few real scientific applications. The best performance so far achieved with real applications is 35.3 Tflops.

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.

Roughly speaking, the target systems for N-body simulations can be classified into two categories: collisional systems and collisionless systems. In the case of collisional systems, the evolution of the system is driven by two-body relaxation process. In other words, by microscopic exchange of thermal energies between particles due to gravitational scattering. In this case, the simulation timescale tends to be long, since the relaxation timescale is proportional to $N/\log N$, where N is the number of particles in the system.

The calculation cost of the simulation of collisional systems increases rapidly as we increase the number of particles N, because of the following two reasons. First, as stated earlier, the relaxation timescale increases roughly linearly as we increase N. This means that the number of the timestep also increases at least linearly[11]. The second reason is that it is not easy to use fast and approximate algorithms such as Barnes-Hut tree algorithm[3] or the fast multipole method[6] to calculate the interaction between particles. This implies that the cost per timestep is $O(N^2)$, and the total cost of the simulation is $O(N^3)$.

The primary reason why it is difficult to use approximate algorithms is that the orbital timescales of particles can be wildly different, essentially because the gravity is an attractive force. Particles can go arbitrary close to each other, requiring arbitrary short timesteps.

It is clearly very wasteful to apply the same timestep to all particles in the system, and it is crucial to be able to apply individual and adaptive timestep to each particle. Such an "individual timestep" algorithm, first developed by Aarseth [1, 2], has been the core for practically any program which handles the time integration of collisional N-body systems such as star clusters and systems of planetesimals.

In principle, it is not impossible to combine individual timestep algorithm and fast algorithms such as Barnes-Hut tree algorithm or FMM. McMillan and Aarseth [16] developed such combination, where the tree structure is dynamically updated to follow the motion of particles and force on a particle is calculated using multipole expansion up to octupole. They assigned predictor polynomials to each node of the tree structure so that they can calculate the force from nodes to particles at arbitrary times. However, so far the attempts to implement such algorithms on parallel computers have not been very successful.

In order to accelerate this kind of N-body simulations, we have developed a series of special-purpose hardware, starting with GRAPE-1 [8].

The basic idea of the GRAPE (GRAvity piPE) architecture is to develop a fully pipelined processor specialized for the calculation of the gravitational interaction between particles. In this way, a single force-calculation pipeline integrates more than 30 arithmetic units, which all operate in parallel. In the cause of the Hermite time integration scheme[10], we need to calculate the first time derivative of the force, resulting in nearly 60 arithmetic operations. This means that we can integrate a large number of arithmetic unit into a single hardware with minimal amount of additional logic.

SC'03, November 15-21, 2003, Phoenix, Arizona, USA

The history of GRAPE hardware can be found in [14]. Here we concentrate on its newest incarnation, the GRAPE-6.

GRAPE-6 is the direct successor of the 1-Tflops GRAPE-4[15]. The main difference between GRAPE-4 and GRAPE-6 is in the performance. The GRAPE-6 chip integrates 6 pipelines operating at 90 MHz, offering the speed of 30.8 Gflops, and the entire GRAPE-6 system with 2048 chips offers the speed of 63.04 Tflops, nearly two orders of magnitude faster than that of GRAPE-4. In this paper, we briefly overview the architecture of GRAPE-6 and present a detailed analysis of its performance for real astrophysical problems.

The plan of this paper is as follows. In section 2, we describe the overall architecture. In section 3 we discuss the differences between GRAPE-6 and its predecessor, GRAPE-4. In section 4 we present the result of performance measurement on simple problems. In section 5, we report the performance of GRAPE-6 for real scientific problems. Section 6 is for summary.

2. THE ARCHITECTURE OF GRAPE-6

In this section, we give an overview of the architecture of GRAPE-6. It calculates the gravitational force, its time derivative, and potential, given by the equations

$$\mathbf{a}_i = \sum_j G m_j \frac{\mathbf{r}_{ij}}{(r_{ij}^2 + \epsilon^2)^{3/2}}, \tag{1}$$

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

$$\phi_i = -\sum_j G m_j \frac{1}{(r_{ij}^2 + \epsilon^2)^{1/2}}, \tag{3}$$

where $\mathbf{a}_i$, $\dot{\mathbf{a}}_i$ and ϕ_i are the gravitational acceleration, its first time derivative, and the potential of particle i, m_i, $\mathbf{x}_i$ and $\mathbf{v}_i$ are the mass, position and velocity of particle i, G is the gravitational constant and ϵ is the softening parameter. Relative position and relative velocity $\mathbf{r}_{ij}$ and $\mathbf{v}_{ij}$ are defined as

$$\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i, \tag{4}$$

$$\mathbf{v}_{ij} = \mathbf{v}_j - \mathbf{v}_i. \tag{5}$$

The position $\mathbf{x}_j$ and velocity $\mathbf{v}_j$ of particle j which exerts the force are "predicted" by the following predictor polynomial

$$\mathbf{x}_{\rm p} = \frac{\Delta t^4}{24}\mathbf{a}^{(2)}{}_0 + \frac{\Delta t^3}{6}\dot{\mathbf{a}}_0 + \frac{\Delta t^2}{2}\mathbf{a}_0 + \Delta t \mathbf{v}_0 + \mathbf{x}_0, \tag{6}$$

$$\mathbf{v}_{\rm p} = \frac{\Delta t^3}{6}\mathbf{a}^{(2)}{}_0 + \frac{\Delta t^2}{2}\dot{\mathbf{a}}_0 + \Delta t \mathbf{a}_0 + \mathbf{v}_0, \tag{7}$$

where $\mathbf{x}_{\rm p}$ and $\mathbf{v}_{\rm p}$ are the predicted position and velocity, $\mathbf{x}_0$, $\mathbf{v}_0$, $\mathbf{a}_0$ and $\dot{\mathbf{a}}_0$ are the position, velocity, acceleration and its time derivative at time t_0, and Δt is the difference between the current time t_j of particle j and the system time t, *i.e.*,

$$\Delta t = t - t_j. \tag{8}$$

The top-level architecture of GRAPE-6 is shown in figure 1. The total system consists of 4 clusters, each of which then consists of 16 GRAPE-6 processor boards and 4 host computers. Clusters are connected through Gigabit Ethernet. Within a cluster, 16 processor boards forms a 2-dimensional grid, so that board ij calculates the forces on particles on host i from particles from host j. With this structure, the bandwidth of the communication between the host computers does not limit the performance. For parallel program with multiple clusters, we still need to use the host-host communication, but the bandwidth is increased by a factor of four, since four hosts in a same cluster can send/receive different data in parallel.

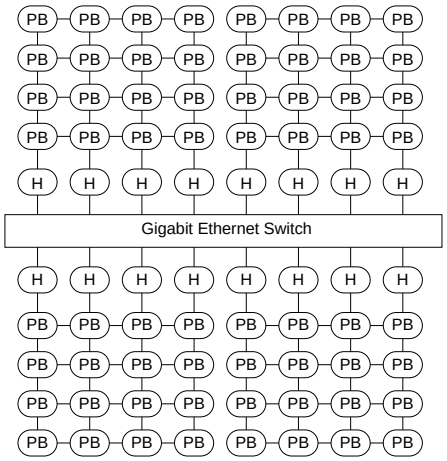

Figure 1: The top level network structure of GRAPE-6.

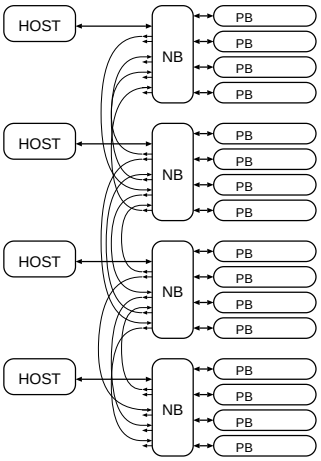

Figure 2: A GRAPE-6 cluster.

Figure 2 shows one cluster. Four processor boards are connected to a host computer through a network board. Four network boards are connected to each other, so that we can use a cluster as a single unit or as multiple units. Figure 3 shows the internal structure of the network board.

Figure 4 shows the structure of a single processor board. It houses 8 processor modules. The processor board has one broadcast network which broadcasts data from the input port to all processor modules, and one reduction network which reduce the results obtained on 32 chips and return to the host through output port.

Each processor module consists of 4 processor chips each with its memory, and one summation unit. The structure

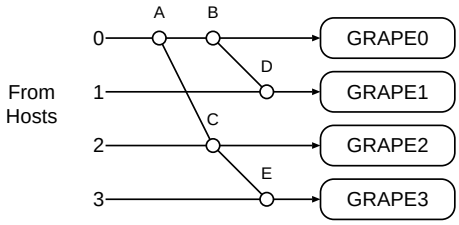

Figure 3: A 4-input example of switching network for parallel GRAPE.

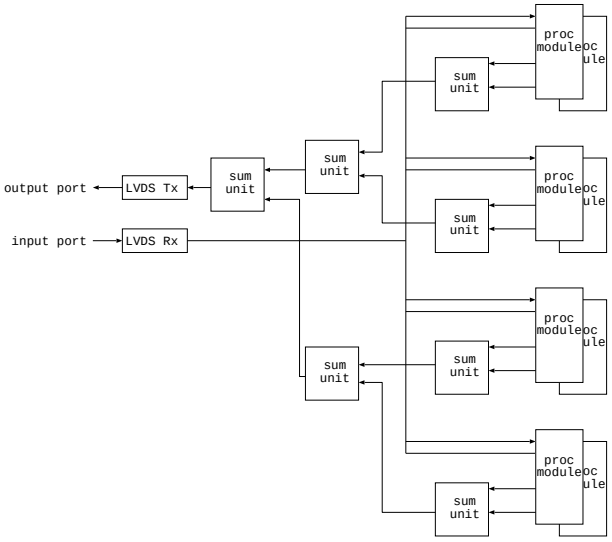

Figure 4: The structure of the processor board.

of a processor module is the same as that of the processor board, except that it has 4 processor chips instead of 8 processor modules.

Figure 6 shows the processor board with eight processor modules installed.

2.1 Processor chip

The GRAPE-6 processor chip was fabricated using Toshiba TC-240 process with the nominal design rule of 0.25μm. The physical size of the chip is roughly 10 mm by 10 mm, and it is packaged into a 480-contact BGA package. It operates on single 2.5V power supply at 90 MHz clock cycle. Heat dissipation is around 12 W when in full operation.

A processor chip consists of six force calculation pipelines, a predictor pipeline, a memory interface and I/O ports. Figure 7 shows the overview of the chip.

Figure 8 shows the block diagram of the force calculation pipeline. it evaluates equations (1)-(3).

The predictor pipeline evaluates the predictor polynomials expressed in equations (6) and (7).

2.2 Packaging and Hosts

Figure 9 shows the complete GRAPE-6 system consisting of five racks (three with two subracks and two with one subracks), with 16 host computers in front of it. Host computers are Linux-running PCs, with AMD Athlon XP 1800+ pro-

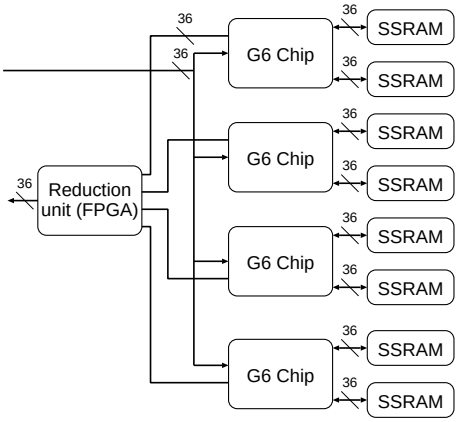

Figure 5: The structure of the processor module.

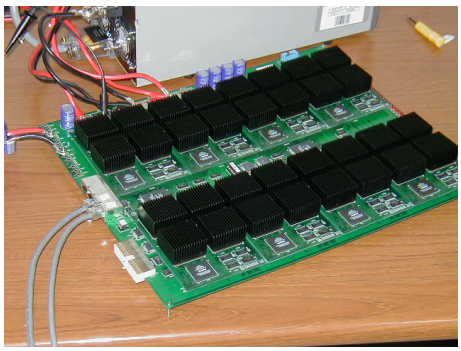

Figure 6: The processor board.

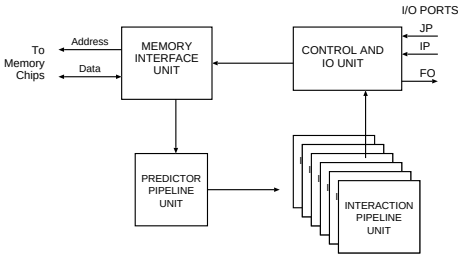

Figure 7: The block diagram of the processor chip.

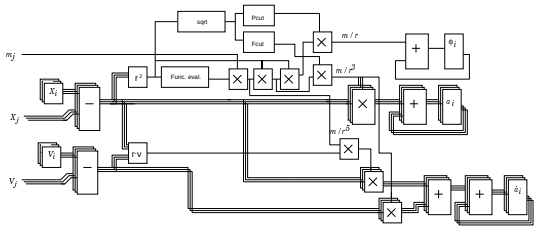

Figure 8: The block diagram of the force calculation pipeline.

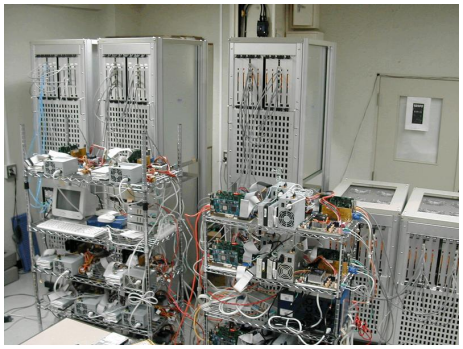

Figure 9: The 64-board, 4-cluster GRAPE-6 with 16 host computers.

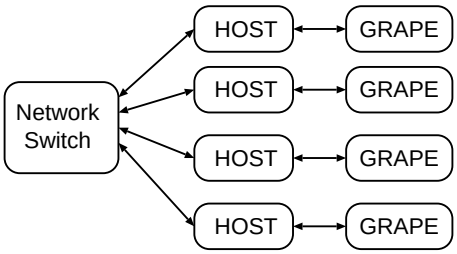

Figure 10: A simple parallel-host, parallel-GRAPE system.

cessors and ECS K7S6A motherboards. They are connected with Gigabit Ethernets. The total power consumption of the system is around 40 KW, when in full operation.

3. DIFFERENCES BETWEEN GRAPE-4 AND GRAPE-6

As described in the previous sections, the architecture of GRAPE-6 is quite different from that of GRAPE-4, even though it is the direct successor of GRAPE-4 for essentially the same goal. In this section, we describe what design changes are made why.

3.1 Differences in the semiconductor technology

The primary difference is that for GRAPE-6 processor chip we used 0.25μm design rule, while with GRAPE-4 we used 1μm design rule. This difference with additional advance in wiring enables us to integrate roughly 20 times larger number of transistors, with 3-4 times faster clock speed. Thus, roughly speaking, a single GRAPE-6 chip offers the speed two orders of magnitude higher than that of GRAPE-4.

This large advance, however, implies almost every design decision had to be changed. In the following, we summarize the changes made.

3.2 The host computer and overall architecture

In GRAPE-4, 4 clusters are connected to a single host, sharing one I/O bus. For the peak speed of 1 Tflops, the single host was still okay for simulations with large number of particles (10^5 and larger), and communication through a single I/O bus was also okay.

With GRAPE-6, however, the peak speed is increased roughly by a factor of 100. On the other hand, the speed of a single host would be improved only by a factor of 10 or so, if we assume the standard Moore's law (performance doubling time of 18 months). Thus, if we want to achieve a reasonable speed for similar number of particles as that for GRAPE-4, we need to use around 10 host computers and the communication channel must be 10-20 times faster than that used for GRAPE-4.

Around the time of the design, it was clear that a shared-memory multiprocessor system with 8-16 processors and sufficient I/O bandwidth would be prohibitingly expensive, with the price tag of the order of 1 M USD. On the other hand, a cluster of 8-16 single-processor workstations or PCs would be much less expensive. As far as the cost is concerned, clearly a cluster of single-processor machines was better than a shared-memory multiprocessor system.

One problem with the cluster is that the simplest configuration (see figure 10) does not work. The reason is the following.

With this configuration, there are two different ways to distribute particle data over processors[9]. One is that each processor has the complete copy of the system (the "copy" algorithm). In this case, parallelization is performed as follows. At each blockstep, each processor determines which particles it updates. After all processors update their share of particles, they exchange the updated particles so that all processors have the updated copy of the system. This algorithm has been used to implement the individual timestep algorithm on distributed-memory parallel computers [19]

In this algorithm, at the end of one block timestep each processor receives the particles updated on all other processors. This means the amount of communication is independent of (or, strictly speaking, is a slowly increasing function of) the number of processors, and the overall performance of the system is limited by the speed of communication.

The other possibility is to let each processor to have a non-overlapping subset of the system, so that one particle resides only in one processor. In this case, with the blockstep algorithm we need to pass around the particles in the current blockstep, so that each processor can calculate the forces from its own particles to particles on other processors(the "ring" algorithm). The amount of communication (host-host and host-GRAPE) per blockstep is again independent of the number of processors. This algorithm is also implemented on distributed-memory parallel computers with direct summation [5] and even with the tree algorithm [18].

For general-purpose parallel computers, this simple algorithm actually works rather well, simply because the calculation speed of single node is so slow. Even a cluster with several hundred nodes is still slower than a single GRAPE-4. So the communication speed of 10-100 MB/s is sufficient. However, with GRAPE-6 we do need a faster speed.

Now we understand that it is possible to use a hybrid of the above two algorithm to solve the bottleneck [9]. In this hybrid algorithm, we organize processors into two-dimensional grid, and distribute the particles so that each row (and each column) has the complete copy of the system.

In the standard realization, this algorithm requires that

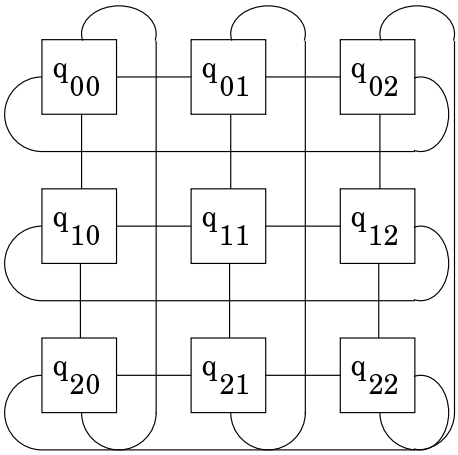

Figure 11: An example of 2D processor array.

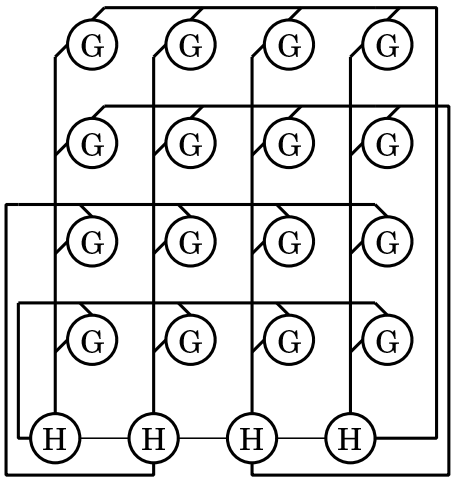

Figure 12: Two-dimensional network of GRAPE hardwares connected to one-dimensional host network.

total number of processors is r^2, where r is a positive integer number. Figure 11 shows an example of a 3^2 array. We divide N particles into r subsets, each with N/r particles. If we number processors from p_{11} to p_{rr}, processor p_{ij} has the copy of both i-th and j-th subsets.

At the beginning of the each blockstep, each processor selects the particles to be updated from subset i. Then all of them calculate the force on them from subset j. After that, the total forces can be calculated by taking summation over columns. Here, we assume that the summed results are obtained on diagonal processors p_{ii}.

After particles in the current block are updated on p_{ii}, they are broadcasted to all other processors in the same row (p_{xi}) and also in the same column (p_{ix}) so that both subsets i and j are updated on each processor.

In this algorithm, the amount of communication for one node is $O(N/r)$. In other words, the effective communication bandwidth (both host-host and host-GRAPE) is increased by a factor r. Thus, the communication speed is improved by a factor proportional to the square root of the number of processors.

At present, this solution looks fine, since the price of the fastest single-processor frontend is now rather low. The cost of the communication network is also rather low, with Gigabit Ethernet adapters available for less than 100 USD per unit.

When we started the design of GRAPE-6 in 1996, we did not expect such a drastic change in the price of fast frontend processors. At that time, RISC microprocessors were still several times faster than PCs with IA-32 architecture, and 100Mbit Ethernet adapters were still expensive. Thus, we had to come up with a design that did not need r^2 processors or fast host-host communication.

It was not really difficult to come up with such a design, since the only thing non-diagonal processors does is the force calculation. Instead of two-dimensional grid of host processors, we can construct a two-dimensional grid of GRAPE hardwares with orthogonal broadcast networks (figure 12). The GRAPE hardwares in the same row store the same data to their particle memories. When they calculate the forces, GRAPEs in the same column receive the same particles and calculate forces on them from particles in the memory. The calculated results on boards in the same column are then summed and returned to the host.

One practical problem with this network architecture is that we cannot divide the system to smaller configurations so that we can run multiple programs. In the case of r^2 hosts, we can divide the system to any sub-squares, down to r^2 single host-GRAPE pairs. In the case of 2D hardware network, we do not have any such division. This problem can be partly circumvented by attaching a simple switching network before memory interface, so that they can select input. So we adopted the network structure shown in figure 3.

In the final design of GRAPE-6, we actually adopted a hybrid of host-grid approach and GRAPE-network approach, to make a reasonable compromise between the flexibility and absolute performance. Of course, this shift from the pure hardware network to hybrid one is made partly because we took into account the evolution of the host computers during the development period of GRAPE-6. It has become more cost effective to use large number of inexpensive (yet fast) computers as host than to have an elaborate hardware network to connect GRAPEs to small number of hosts.

3.3 board-board connection

GRAPE-4 consisted of 36 processor boards, organized in a two-stage simple tree network. Nine boards are housed in one rack, with one backplane bus. These boards are all connected to a control board, which broadcasts the data from the host to all processor boards and take the summation of the calculated data on nine processor boards. Since all boards are connected through a shared backplane bus, the control board has to access processor boards serially. In order to improve the data transfer rate, we used a wide data bus with the width of 96 bits.

The connection between the control board and the host was a 32-bit parallel connection through a coaxial flat cable. This connection is robust and reliable, but had three drawbacks: it was physically large, it was difficult to use long wires, and it was pretty expensive. Because a common

clock signal is used on the both side of the connection, the wire length is limited by the allowable signal skew, which means it is difficult to use fast clock (GRAPE-4 used 16 MHz clock).

A more practical problem is that board-board wiring would become too bulky and cumbersome, with hundreds of flat cables and nearly 10,000 contact points, if we use the same connection for GRAPE-6. In particular, it would be difficult to design the network board, since it needs to have more than 10 connectors. Also, it would be impractical to use a backplane to connect the network board and processor boards, since the number of pins on the network board would be too large.

An obvious solution for this problem is to use a fast serial signal, such as the physical layer of the Gigabit Ethernet. At the time of our design decision, however, Gigabit Ethernet was unpractical because copper wire connection was not available in 1998. Optical connection would be too expensive and would dissipate too much heat.

We adopted what is called "LVDS Link" or "Flat Panel Display (FPD) link", which uses four twisted-pair differential signal lines (three for signals and one for clock). The reason we chose this interface was that inexpensive serializer/deserializer chips were commercially available and that we could use standard category 5 shielded 4-pair cables for 100Mbit Ethernet cable and its connectors for reliable data transmission, for the cable length up to about 5 meters.

Additional advantage of this choice is that we can use backplane connection (with custom-designed signal pattern) for connection between the network board and processor boards. Because the number of signals is small (8 for one port), we can pack many ports into a standard backplane connector (we adopted Compact PCI connector).

3.4 Pipeline chip and memory interface

The processor chip for GRAPE-4 had a single pipeline, which calculates forces on two particles in every six clock cycles (2-way VMP). During force calculation, one chip receives the data of one particle (position, velocity and mass) in every three external clock cycles, and the width of the input data bus was 107 bits.

One GRAPE-4 board housed 48 pipeline chips, all of which receive the same particle data from the memory and calculate the force on two particles. This means that a single board calculates forces on 96 particles in parallel.

This shared-memory architecture is simple to implement. However, we could not use this architecture for GRAPE-6, since the hardware parallelism would become excessively large. The pipeline chip for GRAPE-6 would be roughly 50 times faster than that for GRAPE-4. Thus, even if we somehow increase the data transfer rate by a factor of 5, the number of particles on which the forces are calculated in parallel would increase by a factor of 10, from 100 to 1,000. This number is too large, if we want to obtain a reasonable performance for simulations of star clusters with small, high-density cores. Note that with multiple-board configurations, this number would become even larger. On an $r \times r$ two-dimensional system, the degree of parallelism becomes larger by a factor of r.

The data transfer rate of GRAPE-4 chip was about 200 MB/s. To keep the degree of parallelism to be around 100 or less, the GRAPE-6 chip would have to have the data transfer rate of 5 GB/s, which was well beyond our capability of designing and manufacturing. At 100 MHz clock, the speed of 5 GB/s requires 400 input pins. It is quite difficult to have 400 signal lines, all with 100 MHz data rate, to connect more than a few chips.

Clearly, a different design was necessary. Too large degrees of parallelism arose from our decision to let a large number of chips to share one memory unit. If we reduce the number of chips to share the memory, we can solve the problem. The extreme solution is to attach one memory unit to each pipeline chip, and let multiple pipelines to calculate the force on the same set of chips, but from different set of particles.

This extreme solution has one important practical advantage. The connection between the processor chip and its memory is point-to-point, and physically short (since we can put a processor chip and its memory next to each other). This means a high clock frequency, such as 100 MHz, is relatively easy to achieve.

To attach memory chips directly to the processor chips, we need to integrate the predictor pipeline and the memory controller unit (generation of address and other control signals) to the processor chip. These do not consume much transistors. Therefore it does not have any effect to the performance of the chip.

With GRAPE-6, we adopted a 72-bit (with ECC) data width for transfer between memory and the processor chip. A GRAPE-6 chip integrates six 8-way VMP pipelines. Therefore it calculates the forces on 48 particles in parallel. All pipelines on board calculate the forces on the same set of particles. Thus, even with the largest configuration we considered (an 8×8 system), the degree of the parallelism is still less than 400, not much different from that of full-size GRAPE-4 (which was also 400).

This change from the shared memory design to the local memory design implied we had to take summation of large number of partial forces obtained on chips on one board. With GRAPE-4, we had to take summation of forces obtained on different boards, and we used commercially available single-chip floating-point arithmetic units for this summation. With GRAPE-6, we could not apply this solution simply because such chips no longer existed. Thus, we have to either integrate this summation function into the processor chip, or develop another chip to take summation.

We adopted the latter approach, but used FPGA (Field-programmable gate array) chips to implement adders. It was not impossible to integrate floating-point adders into FPGAs, but such a design would require rather large, expensive FPGA chips and a complex design. In order to simplify the design, we chose to use a block floating point format for the force and other calculated result. In this format, we specify the exponent of the result before we start calculation. The actual value of exponent can be different for forces on different particles, so that we can calculate the forces with wildly different magnitudes in parallel.

With this block floating point method, we can greatly simplify the design of the hardware to take the summation. Of course, we have to supply the value of exponent, but the value of the exponent at the previous timestep is almost always okay. For the initial calculation, we sometimes need to repeat the force calculation a few times until we have a good guess for the exponent.

A rather important advantage of the block floating point format is that the calculated result is independent of the

number of processor chips used to calculate one force. Since the actual summations, both within the chip and outside the chip, are done in fixed-point format, no round-off error is generated during summation. Of course, round-off error is generated when we shift the calculated force to meet the block floating point format, but this error is independent of the order in which the summation is performed. In the case of the usual floating-point format used in GRAPE-4, the round-off error generated in the summation depends on the order in which the forces from different particles are accumulated, and therefore the calculated force is not exactly the same, if the number of boards in the system is different.

Of course, this difference does not have any effect on the accuracy of the simulation itself, since the word length itself is chosen as such. However, it is quite useful to be able to obtain exactly the same results on machines with different sizes, since it makes the validation of the result much simpler.

4. PERFORMANCE

Here we discuss the performance of GRAPE-6 with individual timestep algorithm. As the benchmark run, we integrated the Plummer model with equal-mass particles for 1 time unit (we use the "Heggie" unit [7]). We used standard Hermite integrator [10]. For the softening parameter, we tried three different choices. The first one is a constant softening, $\epsilon = 1/64$. We also tried $\epsilon = 1/[8(2N)^{1/3}]$ and $\epsilon = 4/N$, to investigate the effect of the softening size. Note that for $N = 256$, all three choices of the softening give the same value.

4.1 single-node performance

Figure 13 shows the actual calculation speed achieved as a function of the number of particles N. Here, we define the calculation speed as

$$S = 57 N n_{steps}, \tag{9}$$

where n_{steps} is the average number of individual steps performed per second. The factor 57 means we count one pairwise force calculation as 57 floating-point operations. We took this number from recent entries for Gordon Bell Prize. We assign 38 floating-point operations for the calculation of the pairwize gravitational force, following [20]. The calculation of the time derivative requires additional 19 operations, resulting in 57 operations per pairwize interaction. From this figure, we can see that the achieved speed is practically independent of the choice of the softening.

Roughly speaking, we can model the calculation time per one particle step as follows:

$$T_{single} = T_{host} + T_{comm} + T_{GRAPE}, \tag{10}$$

where T_{host} is the time needed by the host computer to perform computations to integrate one particle, T_{comm} is the time needed for the communication, and T_{GRAPE} is the time to calculate force on GRAPE.

In figure 14, the solid curve shows the measured result for the CPU time per step. The dashed curve is a fit, with constant T_{host}. The dotted curve is an empirical model which takes into account the effect of the cache-hit rate of the host. For small N, the cache-hit rate is higher and therefore the calculation on the host is faster. This model is purely empirical, but apparently gives a reasonable description for the performance.

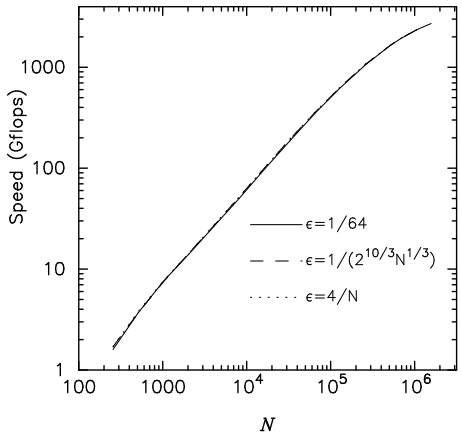

Figure 13: The calculation speed of 1-host, 4-board system in Gflops, plotted as a function of the number of particles in the system.

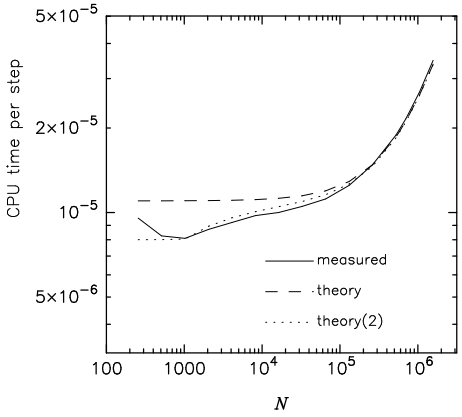

Figure 14: CPU time per one particle step plotted as a function of the number of particles N. Solid curve is the measured result. Dashed and dotted curves denote two different theoretical estimates.

For $N < 1000$, the experimental value is larger than the prediction of the refined theory. This is because the number of particles in one block is too small. The overhead to invoke DMA operations becomes visible.

4.2 multi-node performance

Figure 15 shows the calculation speed for multi-host systems with up to 4 hosts. For up to 4 hosts, the network boards are used to distribute the data, and the communication network between the host computers are used only for synchronization. The parallel program itself is written using MPI, and we used MPICH/p4 over TCP/IP as the MPI library. The network interface is Planex GN-1000TC, which uses NS 83820 chip. We found the performance of MPICH/p4 on this network interface to be quite unsatisfactory, and used UNIX TCP/IP socket system calls for actual communication. The parallel algorithm used here is an individual-timestep variant of 2-dimensional algorithm described in [9].

We can see that a multi-host system require rather large

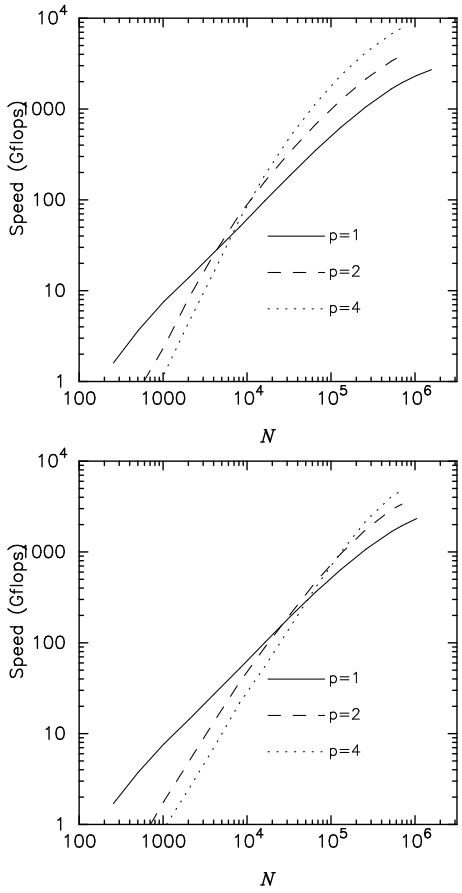

Figure 15: The calculation speed in Gflops plotted as a function of N. Solid, dashed and dotted curves show the results for 1, 2 and 4-node systems, respectively. The left panel shows the result for constant softening, and the right panel $\epsilon = 4/N$.

number of particles to achieve the speed faster than that of the single-host system. Even with the constant softening, the two-host system becomes faster than the single-host system only at $N \sim 3000$, and for $\epsilon = 4/N$, this crossover point moves to around $N \sim 3 \times 10^4$.

Figure 16 shows the calculation time per one particle step for 4-node parallel calculation. This figure clearly shows why the value of N for the crossover is rather large. For "small" N ($N < 10^4$), the calculation time is inversely proportional to the number of particles N. This is because the communication between hosts, which takes constant time per one blockstep, dominates the total cost in this regime. Note that the number of particles integrated in one blockstep (the number of particles which share the same current time) is roughly proportional to N. An extension of the performance model which includes the synchronization overhead reproduces the measured result quite accurately.

4.3 multi-cluster performance

Parallelization over multiple clusters is achieved by the so-called "copy" algorithm, where each cluster maintains the complete copy of the entire system, but integrate only its share of particles. After one step is finished, all clusters

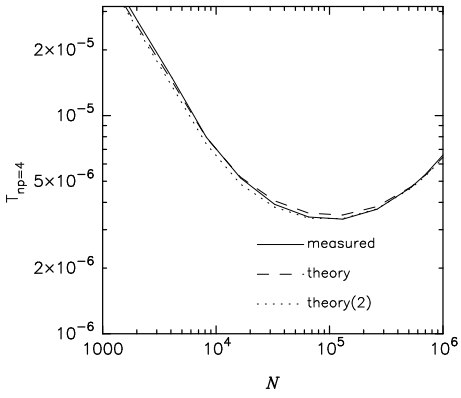

Figure 16: Same as figure 14 but for the case of 4-node parallel calculation.

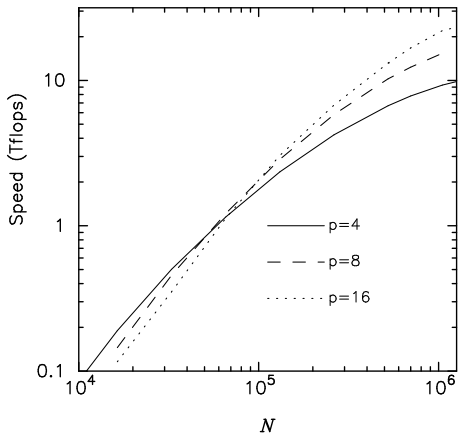

Figure 17: The calculation speed in Tflops plotted as a function of N. Solid, dashed and dotted curves show the results for 4, 8 and 16-node (1, 2, and 4-cluster)systems, respectively. Constant softening is used for all runs.

exchange the updated particles.

Figure 17 shows the calculation speed for multiple-cluster systems, as a function of the number of particles in the system N. The crossover point at which multi-cluster systems becomes faster than single-cluster system is rather high ($N \sim 10^5$), and even for $N = 10^6$, the speedup factors achieved by multi-cluster systems are significantly smaller than the ideal speedup.

Figure 18 shows the calculation time per one particle step for full-cluster calculation (16 nodes, 4 clusters). Theoretical estimate took into account the fact that hosts on different cluster need to exchange the data of particles. Here, again, the calculation time per one particle step is inversely proportional to N, for $N < 10^5$. This means that the main bottleneck is again the synchronization time.

4.4 Performance tuning

From the benchmark results, we can see that the performance of a single-node system is pretty good with better than 1 Tflops at $N = 2 \times 10^5$. The performance of the par-

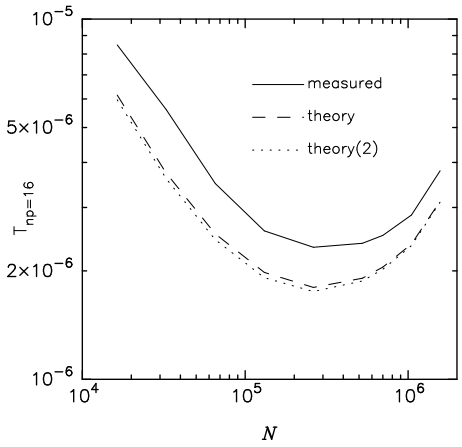

Figure 18: Same as figure 14 but for the case of 16-node parallel calculation.

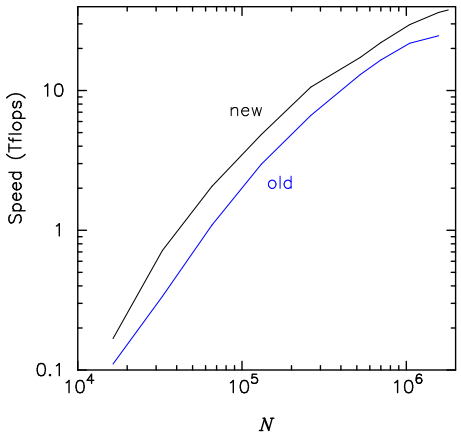

Figure 19: Comparison of the calculation speed with Intel 82540EM (upper curve) and NS 83820 (lower curve).

allel calculation with up to 4 nodes, for which the inter-node communication is via the hardware network on the side of the GRAPE (single-cluster code), the performance is acceptable, with the crossover point (compared to a single-node code) less than 10^4 particles for relatively large gravitational softening. However, for the parallel calculation with 8 and 16 nodes (multi-cluster code), the crossover point (compared to the 4-node code) goes up beyond 10^5 particles.

With both single-cluster and multi-cluster parallel codes the communication latency limits the performance. The communication bandwidth is relatively unimportant, even for the case of multi-cluster calculation. The reason why we can conclude that latency is more important is that in figures 16 and 18, calculation time per particle increases for smaller N, roughly in proportional to $1/N$. If the bandwidth limits the performance, calculation time would be constant in the limit of small N. If the latency limits the performance, the calculation time is proportional to $1/N$, since calculation time is determined by the number of synchronization, which is necessary at every timestep. The number of particles integrated in one timestep is roughly proportional to N. Therefore the time per particle behaves as $1/N$ for small N. For the multi-cluster code, this synchronization overhead is far more severe, because (a) the calculation speed itself becomes faster, (b) overhead of one synchronization operation becomes larger, and (c) the number of synchronization operation itself is larger with the multi-cluster code, since it requires data transfer between host computers.

In order to improve the performance of the parallel code, thus, it is most important to reduce the synchronization overhead. With our current code, synchronization is done through butterfly message exchange using TCP/IP, which is about two times faster than the use of MPI_bbarrier provided by MPICH/p4 over TCP/IP. There are several possibilities to further reduce the overhead. Clearly, the most obvious solution is to move to a faster network hardware such as Myrinet. Myrinet would provide the latency 5-10 times shorter than usual TCP/IP over Ethernet.

Unfortunately, we could not try this option simply because we did not have much funding support this year. So we investigated several other options. One possibility would be to use some communication software which bypasses the TCP/IP protocol layer, such as GAMMA or VIA. Another possibility would be to try different NIC/driver software, while using TCP/IP and socket interface. In the following, we report the result of pursuing the last option. Originally, we used an AMD box and Gigabit NIC based on NS 83820 controller chip. With this combination, round-trip latency was around 200μs, and the peak bandwidth was 60 MB/s. We tried two other NICs, one is Netgear GA621T with Tigon 2 chipset, and the other is the Intel 82540EM chip on board on Iwill P4GB mainboard. Since P4GB is a mainboard for Intel P4, for this one we used Intel P4 2.53GHz processor, overclocked to 2.85GHz. Tigon 2 shows somewhat better throughput (85MB/s), but not much improvement in the latency. However, Intel 82540EM gave us a surprisingly good result. The round-trip latency was cut down to 67μs, and the throughput is increased to 105MB/s.

Figure 19 shows the performance of our original system with NS83820+AMD Athlon and that of the new system with Intel 82540EM+Intel P4. We can see that the performance is improved by 50-100% for the entire range of N. The improvement is larger for smaller N, since the communication overhead is more serious with smaller N. For 1.8M particles, the measured speed reached 36.0 Tflops.

5. PERFORMANCE FOR REAL APPLICATIONS

In the previous section, we reported that the sustained speed of GRAPE-6 reached 36 Tflops. In this section, we present two real applications, where similar speed has been achieved. The first one is the evolution of early Kuiper belt region. The detail of the scientific rational and problem setup is given in [12]. We used 1.8M particles.

We performed a simulation for 21120 dynamical time units, for which the number of individual steps was 1.911×10^{10}. The whole simulation, including file operations, took 16.30 hours. The total number of floating point operations is $1.911 \times 10^{10} \times 1799999 \times 57 = 1.961 \times 10^{18}$, since one particle-particle interaction amounts to 57 floating point operations as we described in section 4. The resulting average computing speed is 33.4 Tflops.

The second example is the simulation of a binary black hole. This calculation is similar to the one we reported in [13]. Recently, several groups tried to study this problem, using parallel code or hybrid method. Maximum number of particles used in these recent works is 4×10^5, significantly smaller than what GRAPE-4 could handle in 1995, even when a highly-approximate method was used [4]. The largest number of particles used for this type of calculation with direct summation code without using GRAPE hardware is currently 32,768 [17].

With GRAPE-6, we used 2M particles. The initial model is a standard Plummer model. We placed two "black hole" particles, which are just massive point-mass particles, with mass 0.5% of the total mass of the system. We integrated the system for 36 time units. for which the number of individual steps was 4.143×10^{10}. The whole simulation, including file operations, took 37.19 hours. The total number of floating point operations is $4.143 \times 10^{10} \times 1999999 \times 57 = 4.723 \times 10^{18}$. The resulting average computing speed is 35.3 Tflops.

It might be useful for illustrative purpose what kind of performance one can achieve with Barnes-Hut treecode on a PC-cluster or massively-parallel general-purpose computer for these problems. Since treecodes scales as $O(N \log N)$, it is appropriate to compare the speed not in terms of the raw flops but in terms of number of particle times steps per second. In above examples, the speed achieved with GRAPE-6 is around 3.3×10^5 particle steps per second.

The only implementation of BH treecode with individual timestep on distributed-memory parallel computer with detailed performance measurement available is Gadget[18]. Unfortunately, the performance of this code does not scale well for more than 16 processors in the case of Cray T3E. The reason lies in the parallelization strategies they used, as we discussed in introduction and section 3.2. In their parallel algorithm, even in the ideal case, amount of communication per processor is constant, independent of the number of processors. In practice, communication cost actually increases, since even though the amount of the data transfer is constant, the number of communication transaction is proportional to the number of processors. On the other hand, the calculation cost per processor reduces as we increase the number of processors. Thus, the performance does not scale even with very fast communication available with T3E. For 16 nodes, the measured speed was around 10^4 steps/sec, or around 3% of the speed achieved with our calculations described above. This measurement, however, is for force accuracy much lower than required in our calculation. For acceptable force accuracy, we provably need at least five times more CPU time, which would result in the speed less than 1% of what we obtained.

If individual timestep is not used, treecode on PC clusters or MPPs has shown very good scalability. For example, Warren *et al.* [20] ran the treecode on 6800-processor ASCI-red, and achieved the speed of 2.55×10^6 particle-steps per second, around 7 times faster than GRAPE-6. However, this is for shared timestep. If we use shared timestep, we need at least 100 times more particle steps, since the ratio between the smallest timestep and (harmonic) mean timestep is larger than 100 for both test calculations. In addition, the accuracy of the force was also rather low for the calculation by Warren *et al.*, so if we assume we need a factor of 5 more calculation cost (this is probably underestimate), shared timestep treecode on 6800-processor ASCI-Red would run approximately 1/70 of the speed of GRAPE-6 for our applications described above.

Even though they gives some idea on the relative performance of GRAPE-6 and BH treecode, these comparisons remain rather speculative, since there is no good implementation of high-accuracy BH treecode with individual timestep on distributed-memory parallel computer. In fact, we do not even know whether it is possible to develop a reasonably scalable parallel algorithm for the combination of individual timestep and BH tree.

6. SUMMARY

In this paper, we presented the overview of the performance characteristic of GRAPE-6, a special-purpose computer for astrophysical N-body problems. The theoretical peak speed of GRAPE-6 is 63 Tflops. We have shown that a reasonable fraction of this theoretical peak speed can be realized with real scientific applications, for modest number of particles. The best measured application performance was 35.3 Tflops.

The main bottleneck of the performance is currently the synchronization overhead of host computers. We are currently investigating ways to further reduce this overhead.

Acknowledgments

We would like to thank all of those who involved in the GRAPE project. In particular, We thank Daiichiro Sugimoto for his continuous support to the project, Masaki Koga and Atsushi Kawai for helping the hardware design, Yoko Funato, Simon Portegies Zwart, Piet Hut, Steve McMillan, Sverre Aarseth and many others for discussions on the experience with GRAPE-4. We thank Ken Namura of IBM, who did the gate-level design of GRAPE-6 chip, and many others from IBM, Toshiba, Ebrain, Kyoden, Uber, Hamamatsu Metrix and other companies who involved in the manufacturing of the hardware. This work is supported by the Research for the Future Program of Japan Society for the Promotion of Science (JSPS-RFTF97P01102).

7. ADDITIONAL AUTHORS

Additional authors: Hiroshi Daisaka (Department of Astronomy, School of Science, University of Tokyo, Tokyo 113-0033, Japan)

8. REFERENCES

[1] J. Aarseth, Sverre. Dynamical evolution of clusters of galaxies, i. *Monthly Notices of Royal Astronomical Society*, 126:223–255, 1963.

[2] S. J. Aarseth. Star Cluster Simulations: the State of the Art. *Celestial Mechanics and Dynamical Astronomy*, 73:127–137, 1999.

[3] J. Barnes and P. Hut. A hiearchical o(nlogn) force calculation algorithm. *Nature*, 324:446–449, 1986.

[4] P. Chatterjee, L. Hernquist, and A. Loeb. Effects of wandering on the coalescence of black hole binaries in galactic centers, astro-ph/0302573, 2003.

[5] E. N. Dorband, M. Hemsendorf, and D. Merrit. Systolic and hyper-systolic algorithms for the gravitational n-body problem, with an application to brownian motion. *J. Comput. Phys*, 185:485–511, 2003.

[6] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73:325–348, December 1987.

[7] D. C. Heggie and R. D. Mathieu. Standardised units and time scales. In P. Hut and S. McMillan, editors, *The Use of Supercomputers in Stellar Dynamics*, pages 233–236, New York, 1986. Springer.

[8] T. Ito, J. Makino, T. Ebisuzaki, and D. Sugimoto. A special-purpose n-body machine grape-1. *Computer Physics Communications*, 60:187–194, 1990.

[9] J. Makino. An efficient parallel algorithm for $O(N^2)$ direct summation method and its variations on distributed-memory parallel machines. *New Astronomy*, 7:373–384, Oct. 2002.

[10] J. Makino and S. J. Aarseth. On a hermite integrator with ahmad-cohen scheme for gravitational many-body problems. *Publications of the Astronomical Society of Japan* , 44:141–151, 1992.

[11] J. Makino and P. Hut. Performance analysis of direct n-body calculations. *The Astrophysical Journal Supplement Series* , 68:833–856, 1988.

[12] J. Makino, E. Kokubo, T. Fukushige, and H. Daisaka. A 29.5 tflops simulation of planetesimals in uranus-neptune region on grape-6. In *Proceedings of SC2002*, pages CD–ROM, Los Alamitos, 2003. IEEE Comp. Soc.

[13] J. Makino and M. Taiji. Astrophysical n-body simulations on grape-4 special-purpose computer. In *Supercomputing '95*, pages CD–ROM, Los Alamitos, 1995. IEEE Comp. Soc.

[14] J. Makino and M. Taiji. *Scientific Simulations with Special-Purpose Computers — The GRAPE Systems*. John Wiley and Sons, Chichester, 1998.

[15] J. Makino, M. Taiji, T. Ebisuzaki, and D. Sugimoto. Grape-4: A massively parallel special-purpose computer for collisional n-body simulations. *The Astrophysical Journal* , 480:432–446, 1997.

[16] S. L. W. McMillan and S. J. Aarseth. An o(n log n) integration scheme for collisional stellar systems. *The Astrophysical Journal* , 414:200–212, 1993.

[17] M. Milosavljević and D. Merritt. Formation of Galactic Nuclei. *The Astrophysical Journal* , 563:34–62, Dec. 2001.

[18] V. Springel, N. Yoshida, and S. D. White. Gadget: A code for collisionless and gasdynamical cosmological simulations. *New Astronomy*, 6:79–117, 2001.

[19] R. Spurzem and H. Baumgardt. A parallel implementation of an aarseth n-body integrator on general and special purpose supercomputers. submitted to Monthly Notices of Royal Astronomical Society, 1999.

[20] M. S. Warren, J. K. Salmon, D. J. Becker, M. P. Goda, T. Sterling, and G. S. Winckelmans. Pentium pro inside: I. a treecode at 430 gigaflops on asci red, ii. price/performance of $50/mflop on loki and hyglac. In *Proceedings of SC97*, pages (CD–ROM). ACM, 1997.