190 TFlops Astrophysical N-body Simulation on cluster of GPUs

Tsuyoshi Hamada
Nagasaki Advanced Computing Center
Nagasaki University
1–14 Bunkyo–machi, Nagasaki–city
Nagasaki, 852–8521, Japan
Email: hamada@nacc.nagasaki-u.ac.jp

Keigo Nitadori
High-Performance Molecular Simulation Team
RIKEN Advanced Science Institute
61–1 Onocho, Tsurumi, Yokohama–city
Kanagawa, 230–0046, Japan
Email: keigo@riken.jp

***Abstract*—We present the results of a hierarchical N-body simulation on DEGIMA, a cluster of PCs with 576 graphic processing units (GPUs) and using an InfiniBand interconnect. DEGIMA stands for DEstination for GPU Intensive MAchine, and is located at Nagasaki Advanced Computing Center (NACC), Nagasaki University. In this work, we have upgraded DEGIMA's interconnect using InfiniBand.**

DEGIMA is composed by 144 nodes with 576 GT200 GPUs. An astrophysical N-body simulation with 3,278,982,596 particles using a treecode algorithm shows a sustained performance of 190.5 Tflops on DEGIMA. The overall cost of the hardware was $411,921 dollars. The maximum corrected performance is 104.8 Tflops for the simulation, resulting in a cost performance of 254.4 MFlops/$. This corrections is performed by counting the FLOPS based on the most efficient CPU algorithm. Any extra FLOPS that arise from the GPU implementation and parameter differences are not included in the 254.4 MFLOPS/$.

I. Introduction

Ground-breaking N-body simulations have won the Gordon Bell prize in 1992 [24],1995-2003 [18], [4], [25], [23], [13], [15], [17], [16], 2006 [19], and 2009 [9] in many cases with the aid of special-purpose computers, *i.e.* GRAPE. However, in the field of N-body simulations during the past few years, graphics processing units (GPUs) have been preferred, rather than the expensive special-purpose computers.

In astrophysical N-body simulations, we solve for the force due to gravitational interactions between N bodies, which is given by:

$$\boldsymbol{f}_i = m_i \, \mathrm{G} \sum_j \frac{m_j \boldsymbol{r}_{ij}}{(\boldsymbol{r}_{ij}^2 + \epsilon^2)^{3/2}}, \tag{1}$$

where $\boldsymbol{r}_i$ and m_i are the position and mass of i-th particle. $\boldsymbol{r}_{ij} = \boldsymbol{r}_j - \boldsymbol{r}_i$, ϵ is a softening parameter, and G is the gravitational constant.

The direct $O(N^2)$ N-body simulations of equation (1) using NVIDIA's CUDA (Compute Unified Device Architecture) programming environment have achieved a performance of over 200 GFlops on a single GPU: Hamada *et al.* reported a performance of 256 GFlops for a 131,072-body simulation on an NVIDIA GeForce 8800 GTX [8]. Nyland *et al.* reported a performance of 342 GFlops for a 16,384-body simulation on an NVIDIA GeForce 8800 GTX [20]. Belleman *et al.* also reported a GPU performance of 340 GFlops for a 131,072-body simulation on an NVIDIA GeForce 8800 GTX with their code `Kirin` [3]. More recently, Hamada reported a performance of 653 GFlops for a 131,072-body simulation on an NVIDIA GeForce 8800 GTS/512 [10][1]. Gaburov *et al.* report a performance of 800 GFlops for a 10^6 particle simulation on two GeForce 9800GX2s with their code `Sapporo` [5].

The use of such data-parallel processors are a necessary but not sufficient condition for executing large scale N-body simulations within a reasonable amount of time. Hierarchical algorithms such as the tree algorithm [1] and fast multipole method (FMM) [6] are also indispensable requisites, because they bring the order of the operation count from $\mathcal{O}(N^2)$ down to $\mathcal{O}(N \log N)$ or even $\mathcal{O}(N)$. Stock & Gharakhani [22] implemented the treecode on the GPU to accelerate their vortex method calculation. Similarly, Gumerov & Duraiswami [7] calculated the Coulomb interaction using the FMM on GPUs.

One of the most common problems in previous GPU implementations of hierarchical algorithms was the inefficient use of parallel pipelines when the number of target particles per cell was small. In order to solve this problem, we developed a method that could pack "multiple walks" that are evaluated by the GPU simultaneously[9]. As a result, a drastic gain in performance was achieved as compared to previous implementations of these hierarchical algorithms on GPUs.

We performed an astrophysical N-body simulation of a sphere of radius 1,152 Mpc (mega parsec) with 3,278,982,596 particles for 1,000 time-steps using our GPU-tree algorithm.

II. GPU Treecode

A. The Treecode

The treecode by Barnes and Hut [1] represents a system of N particles in a hierarchical manner by the use of a spatial tree data structure, as shown in Figure 1. Whenever the force on a particle is required, the tree is traversed, starting at the root. At each level, a gravity center of particles in a cell is added to an "interaction list" if it is distant enough for the truncated series approximation to maintain sufficient accuracy

[1] `http://prograpе.jp/cs/`

SC10 November 2010, New Orleans, Louisiana, USA 978-1-4244-7558-2/10/$26.00

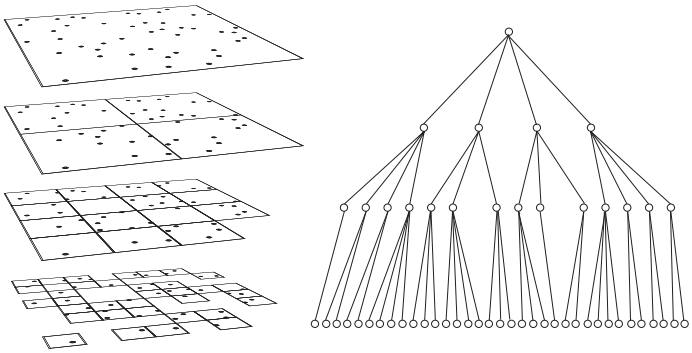


Fig. 1. Hierarchical division of the calculation domain and the corresponding tree structure.

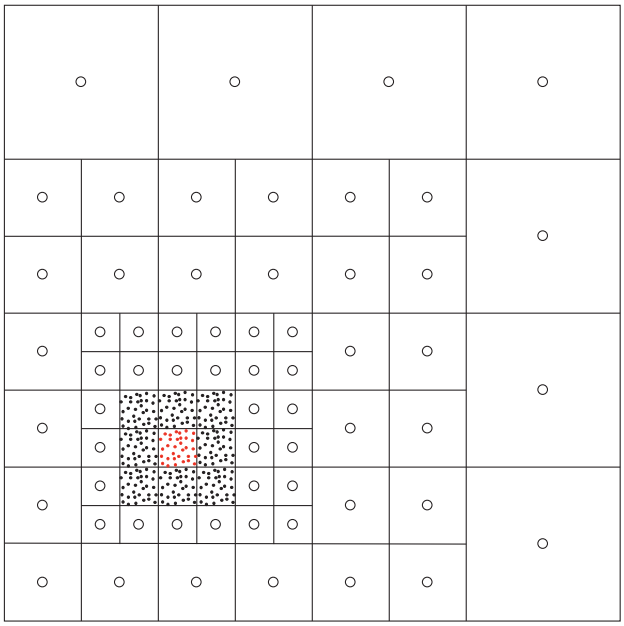


Fig. 2. Interaction list in Barnes' modified tree algorithm. The small black dots represent the interacting particles. The white circles represent the truncated series expansions for each cell. Both the dots and circles are included in an interaction list.

in the force. If the cell is too close, the sub-cells are used for the force evaluation or opened further. This tree traversal procedure is called a "walk". The "interaction list" contains particles themselves or gravity centers of cells.

The original BH algorithm performs the walk for each particle separately. We have modified a parallel treecode [14] for GPU which is based on Barnes' modified tree algorithm [2], where the neighboring particles are grouped together to share an "interaction list". Figure 2 is an illustration of the shared "interaction list" for the particles shown in red. The modified tree algorithm terminates the tree traversal when the cell contains less than N_{crit} particles. This results in an average number of particles per cell N_g in between $N_{crit}/8$ and N_{crit}, depending on the non-uniformity. The modified tree algorithm reduces the calculation cost of the host computer by roughly a factor of N_g. On the other hand, the amount

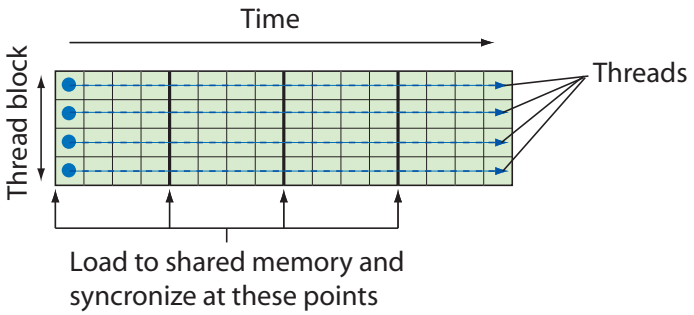


Fig. 3. Definition of the conceptual diagram used in the following three figures.

of work on the force pipelines increases as we increase N_g, since the interaction list becomes longer. There is, therefore, an optimal N_g at which the total computation time is minimum. On CPUs, the optimal value is typically $N_g \approx 32$ [2]. On GPUs, N_{crit} is around 2000. Note that the performance (Flops) we are reporting is the corrected value after the difference in the N_g between the CPU and GPU is taken into account. This correction is performed by counting the Flops based on the most efficient CPU algorithm. Any extra Flops that arise from the difference in N_{crit} are not counted.

Load balancing in our parallel code was achieved by space decomposition using an orthogonal recursive bisection (ORB) [24]. In an ORB, the box of particles is partitioned into two boxes with an equal number of particles using a separating hyper-plane oriented to lie along the smallest dimension. This partitioning process is repeated recursively until the number of divided boxes becomes the same as the number of processors. When the number of processors is not a power of two, it is a trivial matter to adjust the division at each step accordingly. It is expensive to recompute the ORB at each time step, but the cost of incremental load-balancing is negligible.

The interaction list diagram and a conceptual diagram of the computations, shown in Figure 2 and Figure 3 respectively, are used to graphically describe different treecode implementations, and to clearly depict its parallel and sequential computations. For brevity we shall call the target particles "i-particles" and the source particles "j-particles". N_i is the number of i-particles in the terminal cell, and N_j is the number of j-particles that interact with it.

A GPU implementation used in [20] and [3] is depicted in Figure 4. In these works, each thread calculates the force on a different i-particle– we call this the "i-parallel" approach. In this method, when the number of i-particles in a walk N_g (which shares the same interaction list as the j-particles) is smaller than the number of physical processor units, the rest of the processors remain idle. Since the optimal N_g for GPUs is typically of the order of several hundreds, the efficiency of processor usage tends to be low in this implementation. Belleman *et al.* [3] use a large N_g –up to 32,768– to cover for this inefficiency.

An alternative approach proposed by Hamada *et al.* [8] is

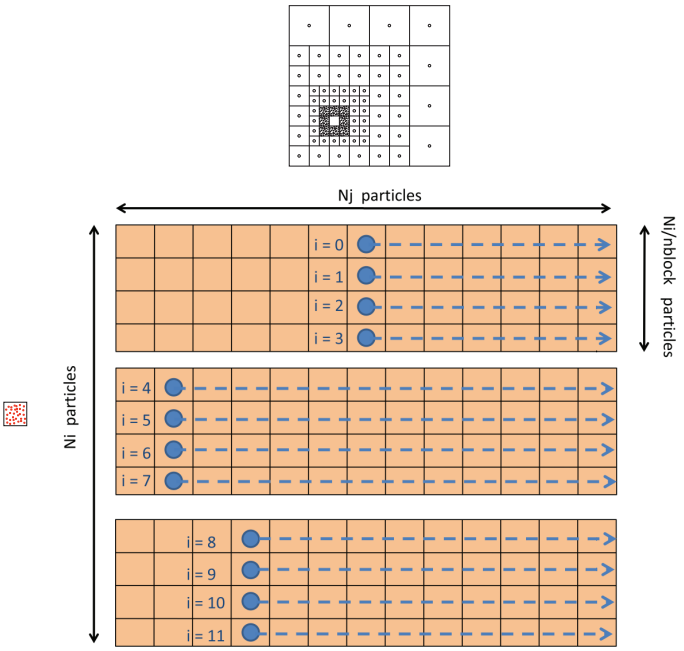


Fig. 4. Implementation of the GPU code reported by Nyland *et al.*[20] and Belleman *et al.*[3].

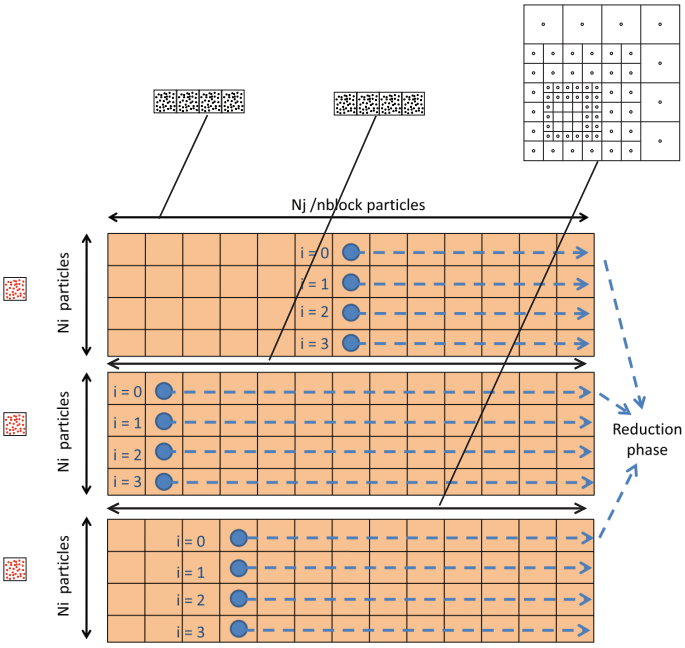


Fig. 5. Implementation of GPU code reported by Hamada et al.[8].

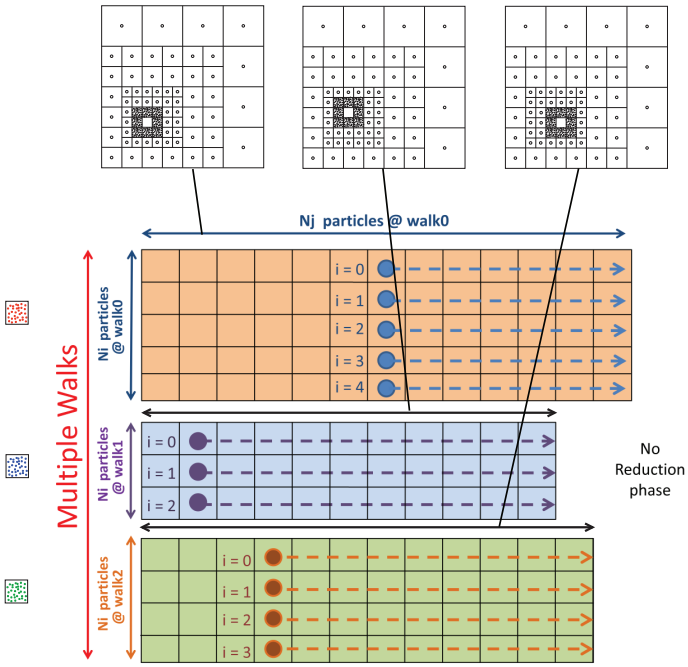


Fig. 6. Implementation of the novel GPU code reported by this paper –the multiple walks method.

shown in Figure 5. In this case, the so-called "j-parallel" approach is used in addition to the i-parallel approach. The j-particles are divided into several groups, and the partial forces on the i-particles are calculated by different blocks. Multiple threads on each block evaluate different i-particles. The number of parallel i-particles is equal to the number of physical processors divided by the number of thread-blocks (N_{blocks}), which is usually smaller than N_g. Therefore, the performance is better than that of the first approach. However, the results of the kernel benchmarks ([9]) show only a marginal increase in performance over the Phantom-GRAPE. This is due to the overhead of the reduction operations required for the partial forces. Indeed, in this approach, partial forces on an i-particle are calculated by different blocks. Consequently, they need to be summed. Unfortunately, the reduction operations on a GPU are slow due to the large overhead in the thread synchronization. There is a report available on the reduction operations that run on GPUs [21]. However, it is only valid for reductions of large arrays with thousands of elements. Therefore, the fastest method for the reduction of small arrays is to use the host CPU. Hence, the amount of communication between the GPU and the host CPU increases by N_{blocks}.

B. Details of proposed code

In this section, we explain the details of our novel approach and present its advantages over previous approaches. They key idea of our novel approach comes from viewing each thread block as one GRAPE, and mapping the previous tree-GRAPE algorithm accordingly. Figure 6 presents a conceptual diagram of our method: at the thread-block level, multiple *walks* (tree traversals) are evaluated by different blocks, where only a

complete *interaction lists* is evaluated within a single block; and at the thread level, threads within a block evaluate different i-particles that belong to the same walk. The sequence of the algorithm is as follows:

1. First, the data of the multiple walks are prepared on the host CPU, *i.e.* the lists of i-particles and j-particles of each walk are prepared. The number of walks in each calculation is determined by the size of the GPU global memory.
2. Multiple walks are then sent to the GPU.
3. Calculations are performed. The GPU is partitioned so that each GPU block evaluates a single walk.
4. The forces calculated by the different blocks in the GPU are received in a bundle.
5. The orbital integration and other calculations are performed on the host CPU.
6. The process is repeated.

A pseudo C++ code for one GPU call from the host is as follows:

```
void force_nwalk(
    float4 xi[],
    float4 xj[],
    float4 accp[],
    int ioff[],
    int joff[],
    int nwalk)
{
  // block level parallelism
  for(int iw=0; iw<nwalk; iw++){
    int ni = ioff[iw+1] - ioff[iw];
    int nj = joff[iw+1] - joff[iw];
    // thread level parallelism
    for(int i=0; i<ni; i++){
      int ii = ioff[iw] + i;
      float x = xi[ii].x;
      float y = xi[ii].y;
      float z = xi[ii].z;
      float eps2 = xi[ii].w;
      float ax = 0;
      float ay = 0;
      float az = 0;
      float pot = 0;
      for(int j=0; j<nj; j++){
        int jj = joff[iw] + j;
        float dx = xj[jj].x - x;
        float dy = xj[jj].y - y;
        float dz = xj[jj].z - z;
        float r2 = eps2 + dx*dx + dy*dy + dz*dz;
        float r2inv = 1.f / r2;
        float rinv = xj[jj].w * sqrtf(r2inv);
        pot += rinv;
        float r3inv = rinv * r2inv;
        ax += dx * r3inv;
        ay += dy * r3inv;
        az += dz * r3inv;
      }
      accp[ii].x = ax;
      accp[ii].y = ay;
      accp[ii].z = az;
      accp[ii].w = -pot;
    }
  }
}
```

In all the previous approaches, the outer most loop (walk-loop) has been performed serially, and either the next i-loop or the inner most j-loop has been mapped to the multiple blocks of GPU. In our new approach, the outer most walk-loop is mapped to the multiple blocks, hence we can fully exploit the parallel nature of the GPU. Our approach has several advantages. First, the reduction of partial forces is no longer necessary since each walk is calculated in a block and no j-parallelization is used. This solves the problem in the previous approach proposed by Hamada *et al.* Second, multiple walks are sent to a GPU simultaneously, and the forces of the multiple walks are also retrieved from the GPU at the same time. This enables a more efficient communication between the CPU and GPU, since the number of direct memory access (DMA) requests decrease, and the length of each DMA transfer becomes longer. Third, it makes parallelization in the host computation easier. Our approach involves the calculation of multiple walks at the same time. Therefore, each thread in the GPU can process a group of different walks in parallel.

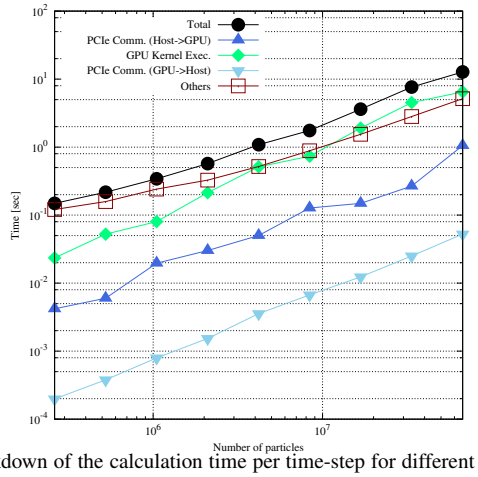
In summary, we have successfully developed a novel tree algorithm that can be implemented efficiently on GPUs. Implementation results [9] show that our new algorithm has a speed performance that is 2.4 times that of the previous GPU implementations.

III. Performance

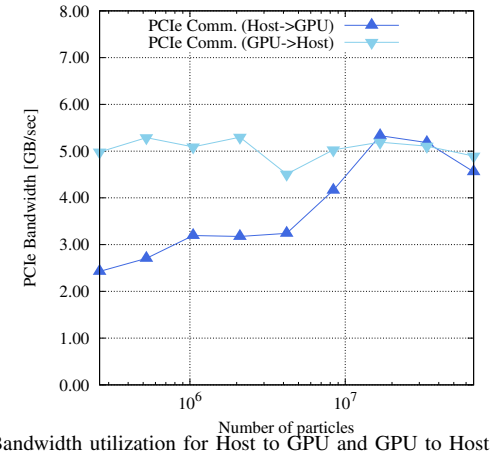
In this section, we present an astrophysical N-body simulation that uses our novel algorithm, on our DEGIMA GPU cluster. We used CentOS for the machines that used x86_64 Linux (version 5.1) as the operating system, and OpenMPI (version 1.2.8) for the implementation of message passing interface and OFED 1.4 for InfiniBand. The CPU code was written in C++ and compiled with the GNU compiler collection (version 4.1.2). The GPU code was also written in CUDA/C++ and complied with the NVIDIA CUDA compilation tool (version 2.3), respectively.

A. Performance on a single node

As a reference for the parallel results, we present the single node performance of our tree algorithm that make use of all four GPU chips. The test problem is based on an uniform distribution of points for which we present the total computational time for a single time-step, this includes: spatial division, tree construction, sorting of particles, list construction, Host-GPU data transfers, force evaluation on the GPU, and time integration. The single node test runs 4 MPI processes, where each process exclusively handles one of the four GPUs. Figure 7 presents the results for a single node performance: Figure 7(a) shows the total computational time for a single time-step for different problem sizes (from 262,144 bodies to 67,108,864 bodies), and a breakdown for the each stage of the tree algorithm (Host to GPU data transfer, GPU kernel execution, GPU to Host data transfer, and all other calculations on the host); Figure 7(b) shows the effective bandwidth utilization of the Host-GPU data transfers through PCI-express gen 2.0 link (with a theoretical peak 8 [GB/sec])



(a) Breakdown of the calculation time per time-step for different problem sizes using single node.



(b) Bandwidth utilization for Host to GPU and GPU to Host data-transfers.

Fig. 7. Single node benchmark with an uniform distribution of points. The opening parameter(θ) and Barnes' modification parameter(N_{crit}) are fixed to $\theta = 0.4$ and $N_{crit} = 2000$, respectively.

for one of the MPI processes. In Figure 7(b), we can see the performance drop for the case in 33,554,432 bodies and 67,108,864 bodies, this is because of "swapping". If our tree algorithm consumes all of the random access memory (RAM) of the host for storing particles, it can employ the swap filesystem that makes use of the hard disk drive (HDD) to store particles overflowing from the RAM.

B. Performance on DEGIMA system

We performed an astrophysical N-body simulation of a sphere of radius 1,152 Mpc (mega parsec) with 3,278,982,596 particles for 1,000 timesteps. A particle represents 1.7×10^{10} solar masses. We assigned initial positions and velocities to particles of a spherical region selected from a discrete realization of a density contrast field based on a standard cold dark matter scenario. We performed the simulation from $z = 23.8$, where z is red-shifted, to $z = 0$. Figure 10 shows images of the simulation with 64M particles.

The change in the performance with the evolution of the system is shown in Figure 8, along with its breakdown in Figure 9. In Figure 9, we can see a constant oscillation of the CPU time that is caused by our algorithm, as sorting of particles takes place every 5 time-steps. The sorting algorithm uses a Morton-ordering scheme.

Here, we use an operation count of 38 operations per interaction. The floating point operation count used by previous Gordon Bell winners for hierarchical N-body simulations has always been 38, both on general and special purpose computers [25], [23], [13], [12], [9]. The 38 comes from the method of calculating gravity using the Newton-Raphson method with an initial guess by Chebyshev interpolation [11]. It is close to 38 when a division and a square root are counted as ten floating point operations, respectively. Although this operation count strictly holds for CPU only, as GPUs can perform division or square root operations in less than 10 multiplications or additions. We have adapted this operation count once again in our calculations for the sake of comparability as an "equivalent CPU Flops" rather than a "GPU Flops".

In total, the number of particle-particle interactions was 3.1×10^{16}. The whole 1,000 time-step simulation took 6,180 seconds and the resulting average computing speed was 190.5 TFlops.

It should be noted that our modified tree algorithm performs a larger number of operations than the conventional tree algorithm that runs on a general-purpose microprocessor. In order to rectify this, we estimated the operation count of the original tree algorithm for the same simulation based on the data of a simulation image (at $z = 0$) and the same accuracy parameter. The number of interaction counts was then found to be 1.7×10^{16}, consequently a correction factor 0.55 is used. Using the correction factor, we obtained an effective sustained speed of 104.8 TFlops, which results on a price/performance ratio of **254.4 MFlops/$** ($3.93/GFlops).

Finally, it should also be noted that our modified tree algorithm is more accurate than the original tree algorithm for the same accuracy parameter, as shown in [2][13].

IV. Hardware Configuration

We have built a 144-node cluster of PCs in which each node has two GeForce GTX295 graphics cards (Figure 13(b)), and each card has two GT200 GPU chips (Figure 11). As a whole, DEGIMA is composed by 576 GT200 GPU chips. One compute node consists of 2.66 GHz Intel Core i7 920 processor, MSI X58 pro-E mother board, 12 GB DDR3-1333 memory and Mellanox MHES14-XTC SDR InfiniBand host adaptor.

Figure 12 shows a block diagram of DEGIMA. Compute nodes are physically organized in groups of 24, the nodes of each group are connected to a 36-port QDR switch through 10 Gbps SDR links. The six switches are interconnected with trunked QDR links (80 Gbps), in a bar topology arrangement. For all connections, we used passive copper cables. A photograph of the GPU cluster is shown in Figure 13(a).

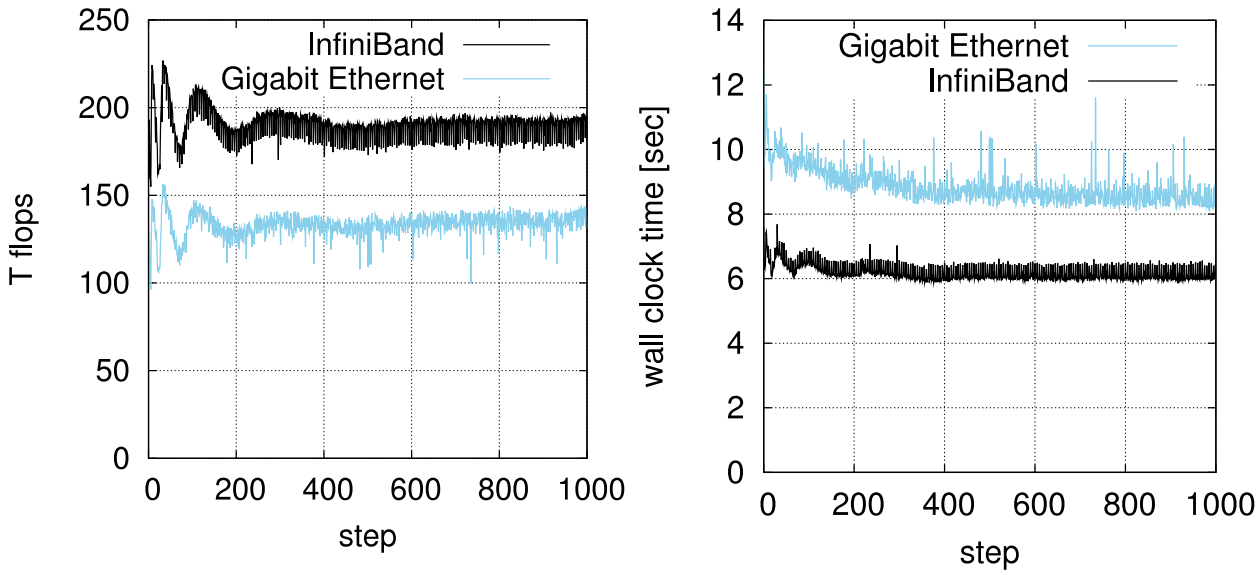


Fig. 8. Performance comparison between two interconnects on DEGIMA: gigabit and InfiniBand.

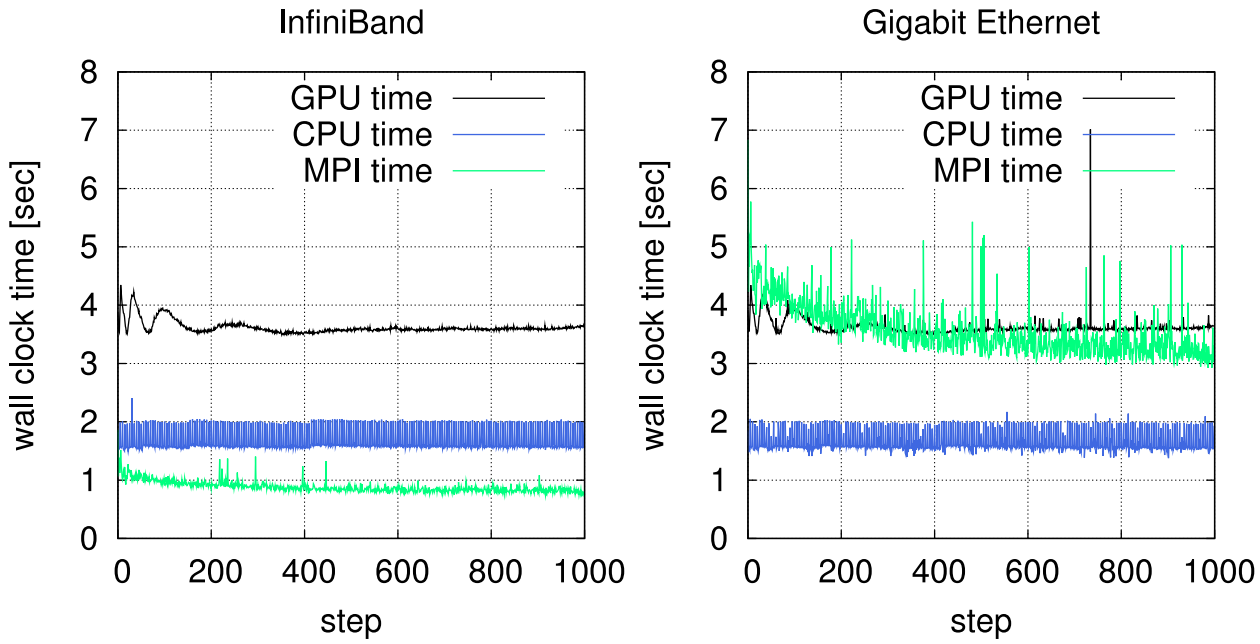


Fig. 9. Breakdown of the calculation time of each time-step.

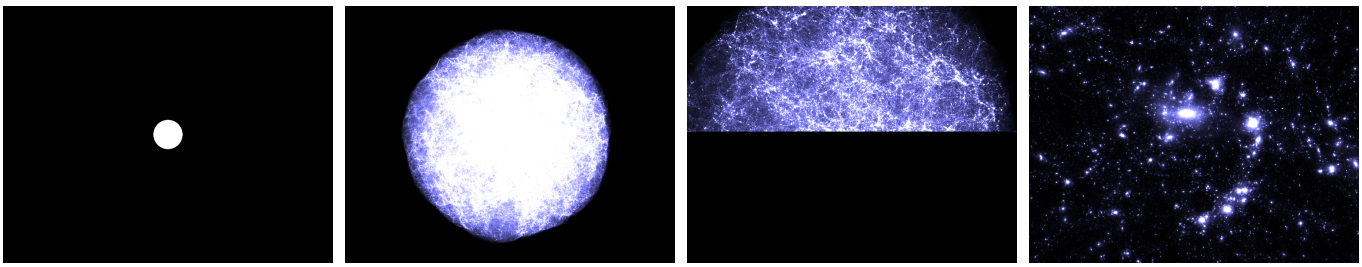


Fig. 10. Snapshots of the simulation with 64M particles at z =18.8 (left), z = 4.5 (middle left), z = 2.6 (middle right), and z = 0 (right).

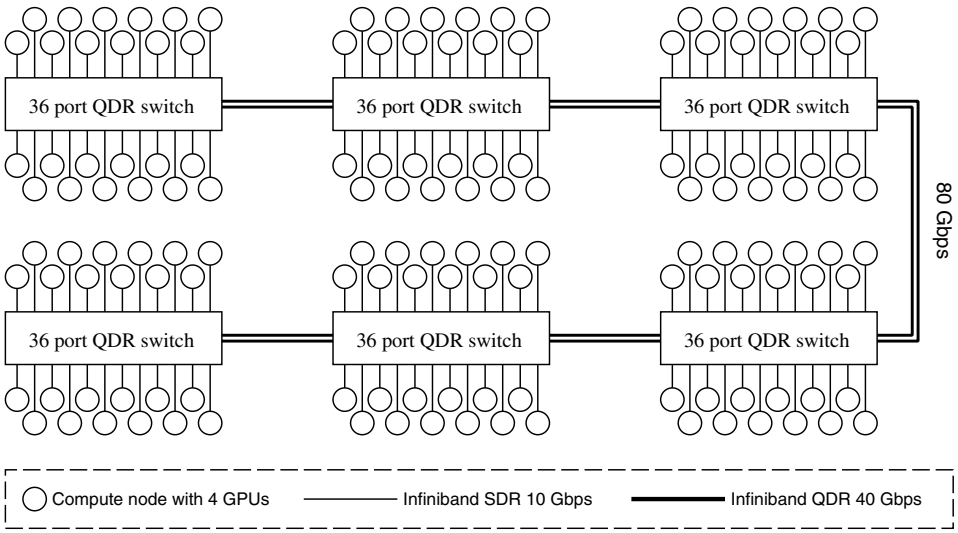
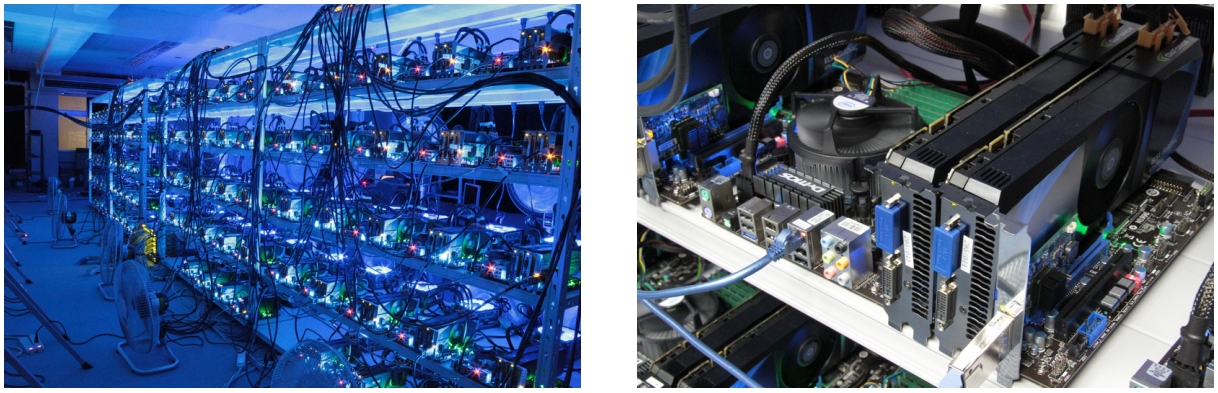


Fig. 12. Block diagram of the GPU cluster (DEGIMA). 24 compute node is connected to a 36-port switch with InfiniBand x4 SDR (10 Gbps) and 6 switches are connected with 2 InfiniBand x4 QDR (80 Gbps) cables in a simple bar topology.



(a) Picture of one half of DEGIMA

(b) Picture of a single node of DEGIMA

Fig. 13. Photographs of DEGIMA cluster.

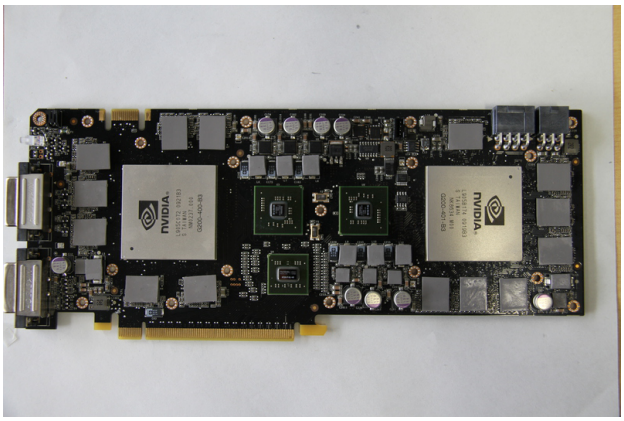


Fig. 11. A photograph of GeForce GTX295 board.

The costs of the constituent elements of DEGIMA are summarized in Table I. The total price of 576 GPUs, 144 PCs, and InfiniBand networks was $411,921. All prices are inclusive of a sales tax of 5%. The sustained performance was 190.5 TFlops. The corrected performance of the astrophysical N-body simulation in section III is 104.8 TFlops, which results in a cost performance of 254.4 MFlops/$ and $3.93/GFlops.

The cumulative power consumption for the whole simulation time of 6,180 seconds results 172 kWh, it includes power for air-conditioners and electric fans, and it was directly measured from the power meter. Therefore, this simulation achieved an averaged power consumption of 100 kW and a performance per watt value of 1.05 GFlops/W.

A. Application mapping

To compensate the relatively narrow inter-switch bandwidth which comes from the low-cost configuration, the application program was optimized to map its spatial locality to the network topology. The 576 MPI processes map to the same number of sub-domains that have been created as follows: First, the root domain is divided into 12 slabs in x-direction.

TABLE I
PRICE OF THE GPU CLUSTER

Elements	Price ($)
GPUs	$ 152,675
Host PCs	$ 155,979
InfiniBand SDR cards	$ 20,646
InfiniBand cables	$ 30,905
InfiniBand switches	$ 51,717
Total	**$ 411,921**

Then, each slab is divided into 6 columns in y-direction, and each column is divided into 8 boxes in z-direction. In this way, 2 slabs fit in 1 switch and 4 boxes fit in 1 compute node. All y- or z-directional communications are confined within a switch, and x-directional communications only occur between neighbor switches.

This kind of approach might be beneficial for many other 'short-range interaction' applications, however, rarely employed unless the network configuration is designed from the application side.

V. CONCLUSION

A hierarchical N-body simulation has been performed on a cluster of 576 graphics processing units (GPUs). Using this fast N-body solver, an astrophysical N-body simulation using 3,278,982,596 particles was performed as a standard benchmark.

With our approach, the tree algorithm shows a significant performance gain when executed on GPUs as compared to the performance of the hierarchical algorithms running on CPUs. The previous implementations of the hierarchical algorithms made it difficult to achieve an effective GPU performance, especially compared to their performances on conventional PC clusters. Using our approach, however, a GPU cluster can outperform a PC cluster from the viewpoints of cost/performance, power/performance, and hardware footprint/performance. The astrophysical N-body simulation showed a sustained performance of 190.5 TFlops. The overall cost of the hardware was $411,921 dollars. The corrected performance is 104.8 TFlops, which results in a cost performance of **254.4 MFlops/$** and $3.93/GFlops. The correction is performed by counting the Flops based on the most efficient CPU algorithm. Any extra Flops that arise from the GPU implementation and parameter differences are not counted.

Compared to the Gordon Bell winner of 2009 [9], the number of particles used in the present simulation is 2.04 times larger. Furthermore, the performance is 4.52 times higher and the price/performance is 2.51 times better as shown in Table II.

TABLE II
COMPARISON WITH SC09 GB WINNER.

	'09 winner	This work	Ratio
Number of particles	1,608,044,129	3,278,982,596	2.04
Price	$ 228,912	$ 411,921	1.80
TFlops (uncorrected)	42.15	190.5	4.52
MFlops/ $	101.4	254.4	2.51

ACKNOWLEDGMENTS

This study was supported by a grant called "Tokubetsu Kyouiku Kenkyu Keihi (Kenkyu Sokushin), Developing the Next Generation GPU-based Supercomputing Environment" from the Special Coordination Funds for Promoting Science and Technology, Ministry of Education, Culture, Sports, Science and Technology, Japan (MEXT).

REFERENCES

[1] J. Barnes and P. Hut. O(nlogn) force-calculation algorithm. *Nature*, 324:446–449, 1986.
[2] J. E. Barnes. A modified tree code: Don't laugh; it runs. *Jouranl of Computational Physics*, 87:161–170, 1990.
[3] R. G. Belleman, J. Bedorf, and S. F. Portegies Zwart. High performance direct gravitational n-body simulations on graphics processing units ii: An implementation in cuda. *New Astronomy*, 13:103–112, 2008.
[4] T. Fukushige and J. Makino. N-body simulation of galaxy formation on grape-4 special-purpose computer. In *Proceedings of the 1996 ACM/IEEE conference on Supercomputing*, pages 1–9, 1996.
[5] E. Gaburov, S. Harfst, and S. Portegies Zwart. Sapporo: A way to turn your graphics cards into a grape-6. *New Astronomy*, 14:630–637, 2009.
[6] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *Journal of Computational Physics*, 73(2):325–348, 1987.
[7] N. A. Gumerov and R. Duraiswami. Fast multipole methods on graphics processors. *Journal of Computational Physics*, 227:8290–8313, 2008.
[8] T. Hamada and T. Iitaka. The chamomile scheme: An optimized algorithm for n-body simulations on programmable graphics processing units. *arXiv:astro-ph/0703100v1*, 2007.
[9] T. Hamada, T. Narumi, R. Yokota, K. Yasuoka, K. Nitadori, and M. Taiji. 42 tflops hierarchical n-body simulations on gpus with applications in both astrophysics and turbulence. In *SC '09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis*, pages 1–12, New York, NY, USA, 2009. ACM.
[10] T. Hamada, K. Nitadori, K. Benkrid, Y. Ohno, G. Morimoto, T. Masada, Y. Shibata, K. Oguri, and M. Taiji. A novel multiple-walk parallel algorithm for the barnes-hut treecode on gpus- towards cost effective, high performence n-body simulation. *Computer Science - Research and Development*, Special Issue Paper:1–11, 2009.
[11] A. H. Karp. Speeding up n-body calculations on machines without hardware square root. *Scientific Programming*, 1:133–141, 1992.
[12] A. Kawai and T. Fukushige. $158/gflops astrophysical n-body simulation with reconfigurable add-in card and hierarchical tree algorithm. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, pages 1–8, 2006.
[13] A. Kawai, T. Fukushige, and J. Makino. $ 7.0/mflops astrophysical n-body simulation with treecode on grape-5. In *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, pages 1–6, 1999.
[14] J. Makino. A fast parallel treecode with grape. *Publications of the Astronomical Society of Japan*, 56:521–531, 2004.
[15] J. Makino, T. Fukushige, and M. Koga. A 1.349 tflops simulation of black holes in a galactic center on grape-6. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, pages 1–18, 2000.
[16] J. Makino, E. Kokubo, and T. Fukushige. Performance evaluation and tuning of grape-6 –towards 40 "real" tflops. In *Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, pages 1–11, 2003.
[17] 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 the 2002 ACM/IEEE conference on Supercomputing*, pages 1–14, 2002.
[18] J. Makino and M. Taiji. Astrophysical n-body simulations on grape-4 special-purpose computer. In *Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, pages 1–10, 1995.
[19] T. Narumi, Y. Ohno, N. Okimoto, T. Koishi, A. Suenaga, N. Futatsugi, R. Yanai, R. Himeno, S. Fujikawa, M. Taiji, and M. Ikei. A 55 tflops simulation of amyloid-forming peptides from yeast prion sup35 with the special-purpose computer system mdgrape-3. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, pages 1–16, Tampa, Florida, 2006.
[20] L. Nyland, M. Harris, and J. Prins. Fast n-body simulation with cuda. *GPU Gems*, 3:677–695, 2007.

[21] S. Sengupta, M. Harris, Y. Zhang, and J. D. Owens. Scan primitives for gpu computing. In *Proceedings of the 22nd ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware*, pages 1–11, 2007.

[22] M. J. Stock and A. Gharakhani. Toward efficient gpu-accelerated n-body simulations. *AIAA Paper*, 2008-608, 46th AIAA Aerospace Sciences Meeting and Exhibit, Reno, Nevada, Jan. 7 - 10:1–13, 2008.

[23] M. S. Warren, T. C. Germann, P. S. Lomdahl, D. M. Beazley, and J. K. Salmon. Avalon: An alpha/linux cluster achieves 10 gflops for $150k. In *Proceedings of the 1998 ACM/IEEE conference on Supercomputing*, pages 1–10, 1998.

[24] M. S. Warren and J. K. Salmon. Astrophysical n-body simulation using hierarchical tree data structures. In *Proceedings of the 1992 ACM/IEEE Conference on Supercomputing*, pages 570–576, 1992.

[25] M. S. Warren, J. K. Salmon, D. J. Becker, M. P. Goda, and T. Sterling. 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 the 1997 ACM/IEEE conference on Supercomputing*, pages 1–16, 1997.