Fast N-body Simulations on GPUs

Algorithms designed to efficiently solve this classical problem of physics fit very well on GPU hardware, and exhibit excellent scalability on many GPUs. Their computational intensity makes them a promising approach for many other applications amenable to an N-body formulation. Adding features such as auto-tuning makes multipole-type algorithms ideal for heterogeneous computing environments.

Rio Yokota, Lorena A. Barba
Mechanical Engineering Dept., Boston University, Boston MA 02215

The classic N-body problem of mechanics solves for the motion of N bodies interacting via the force of gravitation. Beyond gravitational masses, a variety of physical systems can be modeled by the interaction of N particles, *e.g.*, atoms or ions under electrostatics and van der Waals forces lead to molecular dynamics. Also, the integral formulation of problems modeled by elliptic partial differential equations leads to numerical integration having the same form, computationally, as an N-body interaction. In this way, N-body algorithms are applicable to acoustics, electromagnetics, and fluid dynamics. Adding to this diversity of applications, radiosity algorithms for global illumination problems in computer graphics also benefit from N-body methods.

In the absence of a closed-form solution beyond 3 bodies, N-body problems require a numerical approach. The direct simulation of the *all-pairs* interaction results in a computational complexity of order N^2, which becomes too expensive to compute for large N. Nevertheless, the simplicity of direct integration admits ease of use of hardware accelerators leading to a prominent branch of research in this area. Beyond this approach, many notable algorithmic inventions bear on fast computation of N-body interactions. Among them, multipole-based methods are gaining traction as ideally placed for the heterogeneous, many-core hardware environment emerging beyond the petascale computing era. As we show below, fast algorithms such as treecodes and the fast multipole method are characterized by a high computational intensity, as measured using the roofline model. This reflects on their excellent performance on GPU hardware.

History of direct N-body simulation

Numerical simulation of many-body dynamics on digital computers began in the early 60's in two distinct fields of physics: astrophysics and molecular dynamics. The first astrophysical N-body simulation was conducted by von Hoerner[1] for $N = 16$ particles. A subsequent milestone was the calculation by Aarseth[2], which introduced the concept of smoothing and obtained the interaction of up to $N = 100$ bodies. The first molecular N-body simulation[3] was reported for hard-sphere liquids with $N = 500$, a notable extension of this being a simulation of liquid Argon for soft-spheres (with Leonard-Jones potentials)[4]. Since then, the N-body community has done so much more than simply rely on half a century's worth of Moore's law-governed performance improvements. Considerable effort has been dedicated to algorithmic innovations, special-purpose hardware, and performance optimizations.

Special-purpose machines

The N-body problem of astrophysics was such a strong motivator in computational science, that it drove the creation of a special-purpose supercomputer consisting of dedicated pipelines for N-body simulations. GRAPE-1 was the first of its kind and achieved a performance comparable to the CRAY-XMP/1 at 1/10,000 the cost[5]. The 4th generation of this machine, GRAPE-4, was the first computer to reach the 1 teraflop/s milestone in 1995[6] (but without being able to perform the LINPACK benchmark, it could not be so recognized in the Top500 list). The fastest general-purpose supercomputer at the time was the Numerical Wind Tunnel with 236 Gflop/s peak performance. The GRAPE machines were dedicated to gravitational N-body simulations, but were later extended to perform molecular dynamics computations and given the name MD-GRAPE[7]. However, as the cost of fabricating a micro-processor became exceedingly high, it has become difficult for a single research group to design and produce a new processor. This situation was exacerbated by the arrival of GPUs, which by providing the same capability at much lower cost have emerged as a disruptive technology for N-body simulations.

Gordon Bell prize record

N-body simulations have persistently been at the forefront of high-performance computing in terms of the flop/s that they deliver. Groundbreaking N-body simulations have won the coveted Gordon Bell prize of computing fourteen times from 1992 to 2010[6;8–20]. Some may argue that it is not the achieved flop/s that counts, but the science that they deliver. This is a valid assertion, in the same sense that reaching exaflop/s in itself should not be the purpose of high-performance computing. The maximum flop/s for N-body simulations is obtained with the embarrassingly parallel $\mathcal{O}(N^2)$ all-pairs summation, whereas the maximum amount of science will often be delivered by a fast algorithm. Many of the award-winning N-body simulations mentioned above used hierarchical N-body algorithms, and not the all-pairs summation. We discuss these fast N-body algorithms in what follows.

Fast *N*-body algorithms

Introduction to treecodes & FMMs

The amount of computation required by a direct N-body simulation is $\mathcal{O}(N^2)$, which quickly becomes prohibitive for large N, even with the help of special-purpose machines. The treecode algorithm[21] brings the complexity down to $\mathcal{O}(N \log N)$ by clustering the remote particles into progressively larger groups and using multipole expansions to approximate their influence on each target particle. Fast multipole methods (FMMs)[22] can further reduce the complexity to $\mathcal{O}(N)$ by clustering not only the remote particles, but also the nearby particles using local expansions. A common feature in treecodes and FMMs is that they both use tree data structures to cluster particles into a hierarchy of cells. Historically, however, the two methods have followed separate paths of evolution, and have adopted several different practices to meet the demands of their communities of users and their applications.

Treecodes have been popular predominantly in the astrophysics community, where the particle distribution is always highly non-uniform. Thus, the treecode evolved as an inherently adaptive algorithm; on the other hand, the FMM community viewed adaptivity as an additional feature. FMMs have not necessarily focused on a specific application, and practitioners were often aiming at higher accuracy than in the case of treecodes. As accuracy is expressed by a higher truncation level for the series, a variety of fast translation methods have been developed for FMMs (based on spherical and plane wave expansions) that can achieve higher accuracy at reduced cost. Most treecodes, in contrast, use simple Taylor series of low order in Cartesian coordinates. Treecodes also use the ratio between the size of cells and the distance between them to construct the interaction list. This is known as the multipole acceptance criterion (MAC). On the other hand, FMMs use parent, child, and neighbor relationships to construct the interaction list. These differences between treecodes and FMMs are mostly due to historical reasons, rather than mathematical or algorithmic ones. Hence, cross-fertilization of these two fields is surely possible, and may produce an algorithm that takes advantage of the best features of both methods.

Hybrid treecode/FMM algorithm

Some efforts to hybridize treecodes and FMMs have been made from both sides. As mentioned above, the main difference between treecodes and FMMs is the fact that treecodes calculate cell-particle interactions, while FMMs calculate cell-cell interactions for the far field. Warren & Salmon[23] suggested a technique to calculate cell-cell interactions in treecodes, but very little emphasis was placed on this technique in their paper. That work was refined and explained with clarity in a more recent publication by Dehnen[24], where other techniques such as generic tree traversals, mutual cell-cell interactions, and error-controlled multipole acceptance criteria were introduced.

A representative effort at hybridization from the FMM side is that of Cheng *et al.*[25], in which a mechanism to select between cell-particle and cell-cell interactions is discussed that, according to the authors, is always faster than a pure treecode or pure FMM. They also developed a more adaptive cell-cell interaction stencil that considers the interaction of cells at different levels in the tree; this is similar to what could be obtained from the use of the MAC in treecodes, since the MAC would naturally allow interaction between different levels because it relies on the ratio between the cells' size and distance.

The two hybrid algorithms mentioned above—the $\mathcal{O}(N)$ treecode by Dehnen, and the adaptive FMM by Cheng *et al.*—were directly compared on the same machine by Dehnen[24]. The relative performance depends on the required accuracy, where the $\mathcal{O}(N)$ treecode outperforms the adaptive FMM by an order of magnitude for 4 significant digits of accuracy, while the FMM performs better if the required accuracy is over 6 digits. Dehnen attributes the inefficiency of his code at higher accuracies to the fact that the order of expansions is kept constant while accuracy control is effected using the MAC. This suggests that adding the capability to handle variable order of expansions in Dehnen's framework should produce a very fast treecode-FMM hybrid method.

We have recently developed a hybrid treecode-FMM that has similar structure to Dehnen's method but has control over both the order of expansion and MAC. Our kernels are based on spherical harmonic expansions, but have the capability to switch to Cartesian expansions if the required accuracy is lower than a certain threshold; this is the key to achieving high performance for low-accuracy calculations. In addition, we have incorporated a key feature for achieving high performance in today's hardware: the capability to auto-tune the kernels on heterogenous architectures; we will explain this feature in detail below.

Use of GPUs for *N*-body simulation

Early application of GPUs

When CUDA 1.0 was released in 2006 and graphics cards became programmable in C, there were very few scientific applications that could take advantage of this new programming paradigm. N-body simulations were one of the first scientific applications to extract the full compute capability of GPUs. Hamada & Iitaka's initial effort[26] involved parallelizing the *source* particles among thread blocks, which required a large reduction to be performed at the end. Their code was released in early 2007 as CUNBODY (pronounced C-U-N-body). Soon after that, Nyland *et al.* published an article[27] in the *GPU Gems 3* book, using a different approach. In their implementation, the *target* particles were parallelized among thread blocks and no reduction was necessary. In 2008, Belleman *et al.* released a code named *Kirin* (named after a Japanese beer)[28], which relies on the same technique as used by Nyland *et al.* In 2009, Gaburov *et al.* released a GPU N-body code that emulates the GRAPE-6 machine and called it *Sapporo* (after another Japanese beer)[29]. Gaburov *et al.* were able to do this because of the similarities between the GRAPE and GPU architectures. The fact that special-purpose GRAPEs were so similar to GPUs may have given the N-body community an advantage, since many techniques that were developed a decade ago to tune the codes for the GRAPE machine could be used directly on GPUs.

Advantage of *N*-body algorithms on GPUs

Perhaps it is not a coincidence that current GPUs turn out to have similarities to the GRAPE hardware. Computation did not suddenly become cheap, and communication did not suddenly become comparatively more expensive. The trend has always been there, and data-parallel architectures like GRAPE had been performing much better than serial processors all along. It was only a matter of time until mass-produced data-parallel processors appeared and took over certain application areas. In retrospect, the late 90's and early 2000's were peculiar times in the history of high-performance computing, when even the most data-parallel algorithms were being computed on serial processors. Note that the data parallelism mentioned here refers to fine-grained parallelism, and not the coarse-gained parallelism that *was* properly handled in parallel during this era with MPI.

We will quantify the advantage of N-body algorithms on GPUs using the roofline model[30]. This model offers a useful metric for predicting the performance of algorithms on multicore architectures, in which the number of floating-point operations per byte of data transferred is used to determine whether the algorithm will be limited by the floating-point performance of the processor, or by the memory bandwidth.

The roofline model for any algorithm is contingent on the hardware used. For the FMM, we show the roofline on an NVIDIA Tesla C2050

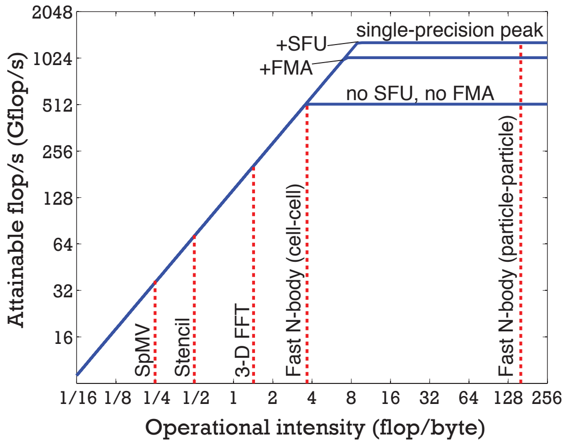

Figure 1: Roofline model of FMM kernels on an NVIDIA C2050 GPU. The 'SFU' label is used to indicate the use of special function units and 'FMA' indicates the use of fused multiply-add instructions. The order of multipole expansions was set to $p = 15$.

GPU in Figure 1, including both the cell-cell and particle-particle interactions. The C2050 has a memory bandwidth of 144 GB/s and a single-precision peak performance of 1288 Gflop/s when special function units (SFU) and fused multiply-add (FMA) operations are fully utilized. Without the use of SFUs, the peak performance decreases to 1030.4 Gflop/s, and further down to 515.2 Gflop/s if the FMA is not applicable. Fortunately, N-body kernels have many adjacent multiply-add operations that can be fused.

As can be seen on Figure 1, the particle-particle interaction is an embarrassingly parallel computation, with the operational intensity reaching 160 and well under the flat part of the roofline. The cell-cell interaction of the FMM is not as compute intensive, but nevertheless has an operational intensity that is still much higher than most algorithms. To show this, we have plotted in the figure the operational intensity of a sparse matrix-vector multiplication (labelled 'SpMV'), a multigrid method with a seven-point stencil ('Stencil'), and a 3D fast Fourier transform ('3D FFT') under the same roofline[30]. In summary, the roofline model distinctly quantifies the high operational intensity of fast N-body algorithms, and reveals their unmistakable advantage on many-core architectures.

Domain decomposition for fast N-body methods

Multi-GPU calculations are indispensable for solving large-scale N-body problems, and traditional MPI-based parallelization must be combined with the GPU kernels in order to achieve this. The key to successfully parallelizing fast N-body algorithms on distributed memory architectures is the partitioning and communication of the tree structure among individual processes. Salmon & Warren[31;32] made a significant contribution to this area by introducing techniques such as orthogonal recursive bisection (ORB), the local essential tree (LET), and N-D hypercube communication. Dubinski[33] summarizes their efforts in a concise and clear manner. A recent improvement in this area is the balancing of linear octrees by Sundar *et al.*[34], which is a technique to repartition the domain so that the per-processor partitions are better aligned with cell boundaries at coarser levels of the octree. This was a key technology behind the 2010 Gordon Bell prize-winning paper[35].

Many-GPU calculations with FMM

Biomolecular electrostatics

We have demonstrated the application of the FMM algorithm in a multi-GPU system for biological applications in Yokota *et al.*[?] There, the FMM is used to accelerate a boundary element method solution of the continuum electrostatic model for biomolecules, a popular model for calculating electrostatic interactions between biological molecules in solution. Through guest access to the Degima cluster at the Nagasaki Advanced Computing Center (a system that currently holds the #3 spot in the Green500 list), we were able to test the parallel FMM on hundreds of GPUs. The largest calculation solved a system of over a billion boundary unknowns for more than 20 million atoms, requiring one minute of run time on 512 GPUs. This work demonstrates that the FMM on GPU could enable routine calculations that were unfeasible before, for example, for analyses of protein-protein interactions in vital biological processes.

Fluid turbulence simulations

Recently, we applied our periodic FMM algorithm to the simulation of homogeneous turbulent flow in a cube, demonstrating scalable computations with many GPUs. These calculations

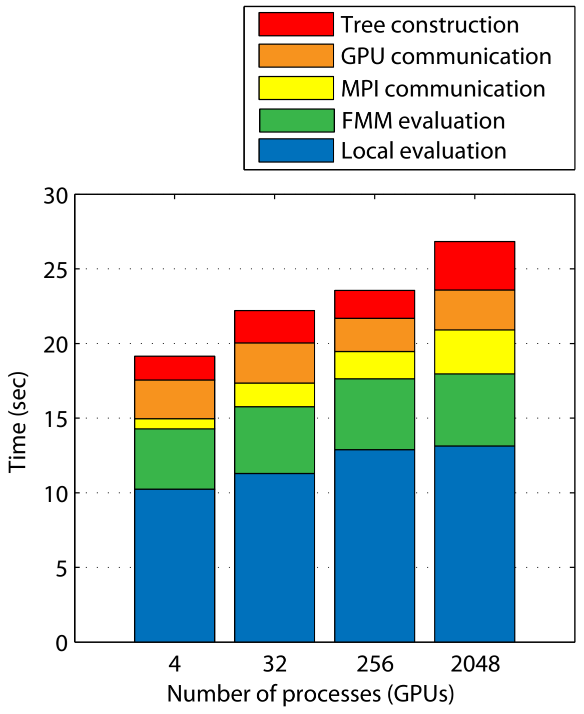

Figure 2: Weak scaling test with our FMM algorithm on many GPUs. The parallel efficiency with 2048 GPUs is 72%.

were carried out in the TSUBAME 2.0 system at the Tokyo Institute of Technology, thanks to guest access provided by the Grand Challenge Program of TSUBAME. A total of 2048 NVIDIA M2050 GPUs were used, corresponding to half of the complete system, to achieve a sustained performance of 0.5 petaflop/s. The peak performance of the complete TSUBAME system is 2.4 petaflop/s, thus we achieved excellent usage of the computational resource in an application.

The preferred method for the simulation of homogeneous isotropic turbulence in a periodic cube has always been the pseudo-spectral method. We compared an FMM-based vortex method with an FFT-based pseudo-spectral method for turbulence at $Re_\lambda \approx 500$ using 2048^3 grid points, confirming that relevant statistics quantitatively match. Moreover, the parallel scalability of the FMM algorithm is excellent, obtaining 72% parallel efficiency in a weak scaling test up to 2048 GPUs. With this recent large-scale calculation using the FMM, we show that the scalability of this algorithm starts to become an advantage over FFT-based methods beyond 2000 parallel MPI processes.

The results of a weak scaling test with 4 million particles per process is shown in Figure 2. The label 'Local evaluation' on the bar plot corresponds to the particle-particle kernel evaluation, while the 'FMM evaluation' label corresponds to the sum of all the other kernel evaluations. The MPI communication is overlapped with the kernel evaluations, so in the bar plot we show the actual time for communications, and the excess time required for the FMM evaluation obtained by subtracting the MPI communications time to the total evaluation time. In this way, the total height of the bar correctly represents the total wall-clock time of the full overlapped calculation.

A new hybrid treecode-FMM with auto-tuning

The purpose of auto-tuning

Like most algorithms, fast N-body methods permit a wide variety of mathematical formulations and computational implementations, some of which are better suited for a particular architecture and not for others. Some of the available choices are: use of Cartesian vs. spherical expansions, rotation-based vs. plane wave-based translations, cell-cell vs. cell-particle interactions for the far field, and choice of order of expansion vs. MAC-based error optimization. For each of these choices, the option that will be most efficient depends on (i) the required accuracy, (ii) the hardware and, unfortunately, (iii) the implementation/tuning of kernels. The hardware dependence is particularly problematic when heterogeneous architectures come into the equation. The reason is that the various algorithmic kernels (*e.g.*, cell-cell and cell-particle interaction) achieve different levels of performance measured in flop/s on different hardware. Thus, a well-balanced FMM calculation on one type of hardware might be unbalanced with the same parameters when moving to a different hardware. A simple solution to this problem would be to time all kernels on each hardware, and use this information to select the optimal combination during runtime. For example, if the GPU can perform cell-particle interactions faster than cell-cell interactions for a certain number of particles per cell, our hybrid treecode-FMM will shift more towards treecodes automatically.

Kernel pre-calculation

A critical ingredient for the auto-tuning of hybrid fast N-body methods is the pre-calculation of the kernels. All kernels are evaluated using artificial coordinates, mass/charges, multipole coefficients, and their execution time is measured. This information is then used to select the optimum kernel during the dual tree traversal (described below). This allows the fast N-body code to select on the fly between cell-cell, cell-particle, particle-particle interactions, and also rotation-based vs. plane wave-based translations, etc. On GPUs, certain kernels will achieve higher flop/s than others, but since the initial timings of the kernels will reflect this, the selection of kernels will be optimized for the GPU architecture automatically. Therefore, all the manual parameter tuning associated with hybridizing treecodes and FMMs is rendered unnecessary by our auto-tuning approach.

Dual tree traversal

Auto-tuning of a treecode-FMM hybrid method by dynamically selecting kernels during runtime requires a generic and flexible $\mathcal{O}(N)$ algorithm for traversing the tree. We now describe such a generic tree traversal algorithm, illustrated in Figure 3. This algorithm can be viewed as a dual tree traversal for the target tree and source tree, which results in $\mathcal{O}(N)$ complexity. In most cases the targets and sources are identical, but the present method can also handle cases where they are not.

The dual tree traversal uses a typical "last in, first out" stack data structure that holds pairs of cells. As shown on the left panel of Figure 3, once the tree is constructed, the pair of root cells is pushed into an empty stack. After this initial step, the following iterative procedure is applied—as illustrated on the right side of Figure 3. First, a pair of cells is popped from the stack, and the larger of the two cells is subdivided. Always splitting the larger of the two cells guarantees that the pairs in the stack consist of cells of somewhat similar size, which is a necessary condition for achieving $\mathcal{O}(N)$. Next, the smaller cells created by the subdivision of a member of the pair are matched with the other member of the pair. If a matching set is composed of leaf cells (at the terminal level of the tree), a direct summation is performed between all particles in the cells. If not, the multipole acceptance criterion is used to examine each of the newly created pairs of cells. If the cells in a new pair are far/small enough, the interaction between the two cells is immediately calculated. The type of interaction—cell-cell vs. cell-particle, rotation-based vs. plane wave-based translation—is then chosen to optimize the performance. If the cells are too close/large, then the pair of cells is pushed to the top of the stack. This procedure then starts again and is repeated until the stack is empty.

The procedure described is a simple but highly adaptive and flexible way of performing an $\mathcal{O}(N)$ tree traversal. Interestingly, traditional FMMs do not rely on such algorithms; they construct instead a rigid interaction list for every target cell using parent, child, and neighbor relationships. This in turn requires the kinship in the tree to be directly associated to the geometrical proximity of the cells, *i.e.*, all cells must be perfect cubes. In contrast, the dual tree traversal can be applied to adaptive k-d trees with rectangular cells, since the proximity of cells is handled by the MAC, and is unrelated to the tree structure. In other words, the dual tree traversal provides a simple bookkeeping strategy for constructing a MAC-based interaction stencil that is mutually exclusive at each level, by letting the target cells inherit a unique stack of source cells from their parents.

Results with the auto-tuning hybrid treecode/FMM method

To demonstrate the behavior of our hybrid treecode-FMM, we compare the performance of a pure treecode, a pure FMM, and a hybrid treecode-FMM. The calculation is performed on a single core of an Intel Xeon X5650 2.67GHz processor, and an NVIDIA Tesla C2050. Particles are randomly distributed in a cube of size $[-1, 1]^3$ and the number of particles was varied from $N = 10^3$ to $N = 10^6$ for the CPU runs, and from $N = 10^4$ to $N = 10^7$ for the GPU runs. The calculations are performed for the Laplace potential and force. The pure treecode, pure FMM, and hybrid all use the same adaptive tree structure, dual tree traversals, and MAC-based interaction lists. The only difference between the three cases is that the pure treecode only selects between cell-particle and particle-particle interactions, and the FMM only selects between cell-cell and particle-particle interactions, while the hybrid can choose between any of cell-cell, cell-particle, and particle-particle interactions. The MAC is chosen to be $\theta = 0.5$, which is equivalent to a standard FMM neighbor list of 27 cells. All calculations including the pure treecode used spherical harmonics expansions, and auto-tuning was not applied for the selection between different fast translation schemes at this time. The

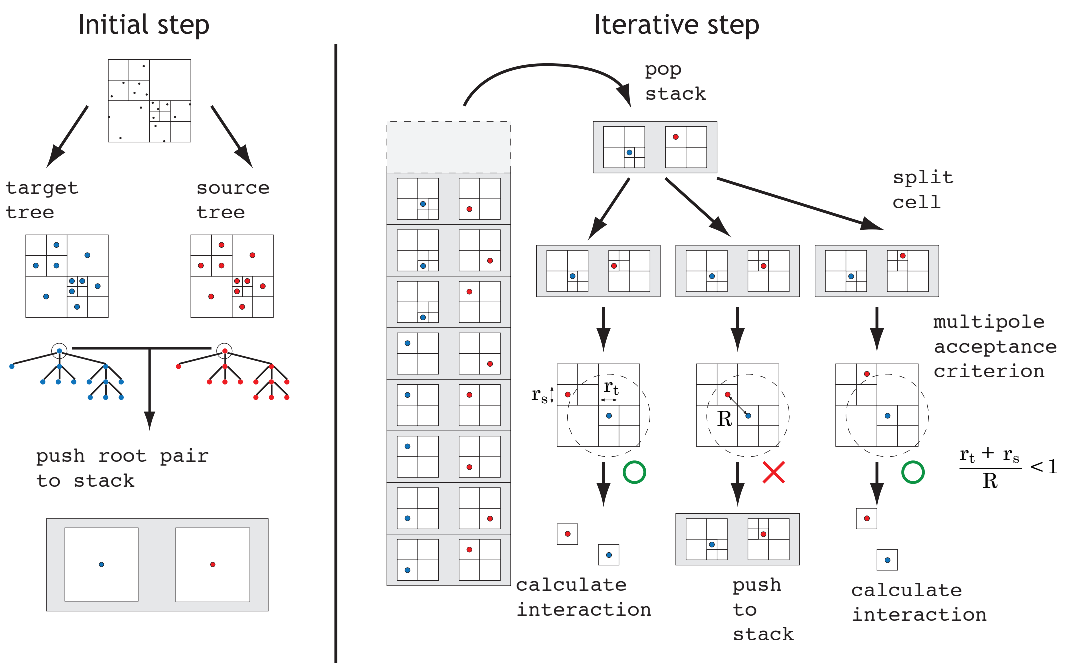

Figure 3: Dual tree traversal: illustration of the stack-based procedure.

auto-tuning was only used to optimize the selection between, cell-cell, cell-particle, and particle-particle interactions in the hybrid code, and was not used in the pure treecode or pure FMM.

A comparison between a pure treecode, a pure FMM, and a hybrid treecode-FMM on a single core of a Xeon X5650 2.67GHz CPU is shown in Figure 4. The order of expansion is $p = 5$, which is equivalent to 4 significant digits of accuracy for the potential and 2 significant digits for the force. Since the pure treecode is using spherical expansions, the performance at low accuracy may be suboptimal compared to highly tuned Cartesian treecodes. Therefore, we observe a large gap between the treecode and FMM in Figure 4. An interesting result is that the hybrid method seems to remove the wavy behavior of the N-dependence in pure FMMs. This is a visible result of the auto-tuning capability in the hybrid method.

A similar comparison to Figure 4 is shown in Figure 5, but this time on a GPU. The calculation conditions are identical to the CPU case, where the order of expansion is $p = 5$ and particles are randomly distributed in a cube. The cell-cell, cell-particle, and particle-particle interactions were calculated on the GPU, while the tree construction was done on the CPU. All GPU kernels used single-precision. Our GPU code is not optimized for small N and has an overhead that distorts the $\mathcal{O}(N)$ scaling for small N.

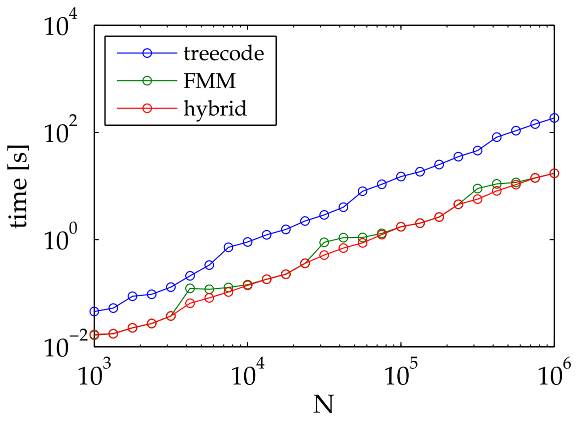

Figure 4: Comparison between a pure treecode, a pure FMM, and a hybrid on CPU. Order of expansion is $p = 5$ and particles are randomly placed in a cube. The kernel is for Laplace potential+force.

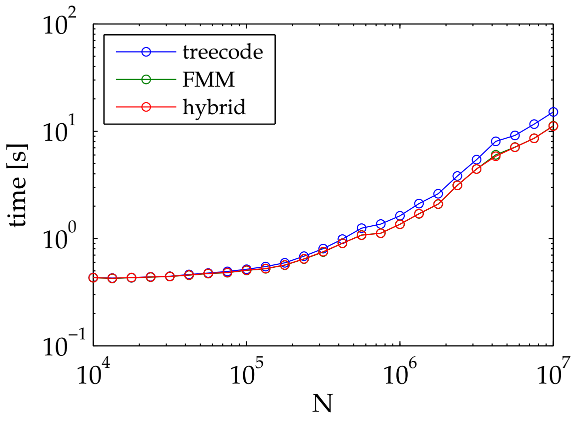

Figure 5: Comparison between a pure treecode, a pure FMM, and a hybrid on GPU. Order of expansion is $p = 5$ and particles are randomly placed in a cube. The kernel is for Laplace potential+force.

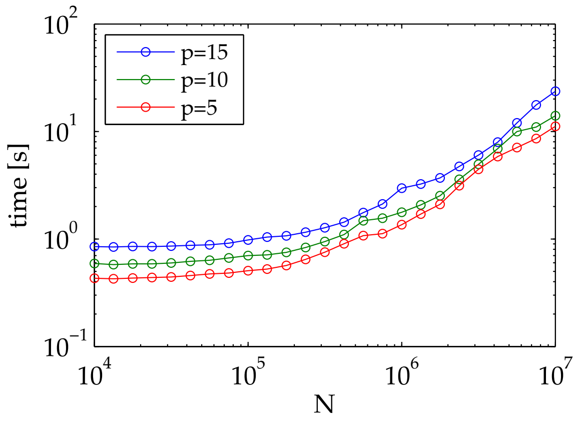

Figure 6: Comparison between different order of expansion for hybrid method on GPU. Order of expansion is $p = 5$ to $p = 15$ and particles are randomly placed in a cube. The kernel is for Laplace potential+force.

The difference between the treecode and FMM is much smaller than in the CPU case, since the cell-particle kernel achieves higher flop/s than the cell-cell kernel on the GPU. There is no clear difference between the pure FMM and auto-tuned hybrid method on the GPU; this reflects the difficulty of auto-tuning on GPUs, which faces the following problem: the calculation time is *not* proportional to the problem size because larger problems are able to utilize more threads, and thus run more efficiently on GPUs. This makes it very difficult to predict the execution time of each kernel, and hence the optimization between cell-cell, cell-particle, and particle-particle kernels will not function properly. Although a polynomial fit for the relation between the problem size and execution time could be used for selecting the optimal kernels, we have not yet investigated this possibility.

The accuracy dependence of our hybrid treecode-FMM is shown in Figure 6. The calculation conditions are identical to the previous calculations except the order of expansion is changed from $p = 5$ to $p = 15$. The overhead in the GPU calculations seen at $N = 10^4 - 10^5$ seems to be proportional to p. Although this overhead can be removed by optimizations for small N, we did not consider it a priority at this time. The p-dependence is rather small considering the fact that we are using a $\mathcal{O}(p^4)$ cell-cell interaction kernel. One reason for this is that at the considered range of p a smaller constant in front of the p^4 term allows the lower order terms to dominate. Another reason is the GPU being able to process the kernels with larger p more efficiently, because they have a higher flop/byte rate. Our code is able to calculate the Laplace potential and force for $N = 10^7$ particles with $p = 15$ in approximately 23 seconds on a single GPU.

With the current hybridization of treecodes and FMMs, combined with auto-tuning capabilities on heterogeneous architectures, the flexibility of fast N-body methods has been greatly enhanced. The fact that the current method can automatically choose the optimal interactions, on a given heterogeneous system, alleviates the user from two major burdens. Firstly, it is no longer necessary to worry about FMMs being suboptimal at low accuracy and treecodes being suboptimal at high accuracy—they are now one algorithm. Secondly, there is no need to tweak parameters, *e.g.*, particles per cell, in order to achieve optimal performance on GPUs—the same code can run on any machine without changing anything. These features are a requirement to developing a black-box software library for fast N-body algorithms *on heterogeneous systems*, which is our immediate goal. Such a library is being made available in the open-source model, and will be released under the MIT license; to obtain the code, the reader can access the website at www.bu.edu/exafmm.

Acknowledgements

We're grateful for the support from the US National Science Foundation and the Office of Naval

Research. Recent grant numbers are NSF OCI-0946441, and ONR award #N00014-11-1-0356.

References

[1] S. von Hoerner. Die numerische integration des n-körper-problemes für sternhaufen i. *Zeitschrift für Astrophysik*, 50:184–214, 1960.

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

[3] B. J. Alder and T. E. Wainwright. Phase transition for a hard sphere system. *Journal of Chemical Physics*, 27:1208, 1957.

[4] A. Rahman. Correlations in the motion of atoms in liquid argon. *Physical Review*, 136(2A):A405–A411, 1964.

[5] D. Sugimoto, Y. Chikada, J. Makino, T. Ito, T. Ebisuzaki, and M. Umemura. A special-purpose computer for gravitational many-body problems. *Nature*, 345:35–37, 1990.

[6] J. Makino and M. Taiji. Astrophysical N-body simulations on GRAPE-4 special-purpose computer. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, page 63, New York, NY, USA, 1995. ACM.

[7] T. Fukushige, M. Taiji, J. Makino, T. Ebisuzaki, and D. Sugimoto. A highly parallelized special-purpose computer for many-body simulations with an arbitrary central force: Md-grape. *The Astrophysical Journal*, 468:51–61, 1996.

[8] M. S. Warren and J. K. Salmon. Astrophysical N-body simulations using hierarchical tree data structures. In *Proceedings of the 1992 ACM/IEEE conference on Supercomputing*, pages 570–576, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[9] T. Fukushige and J. Makino. N-body simulation of galaxy formation on GRAPE-4 special-purpose computer. In *Supercomputing '95: Proceedings of the 1995 ACM/IEEE conference on Supercomputing*, 1996.

[10] M.S. Warren, J.K. Salmon, D.J. Becker, M.P. Goda, T. Sterling, and W. Winckelmans. Pentium Pro inside: I. A treecode at 430 Gigaflops on ASCI Red, II. Price/performance of $50/Mflop on Loki and Hyglac. In *Supercomputing, ACM/IEEE 1997 Conference*, page 61, 1997.

[11] 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.

[12] A. Kawai, T. Fukushige, and J. Makino. $7.0/Mflops astrophysical N-body simulation with treecode on GRAPE-5. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, New York, NY, USA, 1999. ACM.

[13] 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.

[14] T. Narumi, R. Susukita, T. Koishi, K. Yasuoka, H. Furusawa, A. Kawai, and T. Ebisuzaki. 1.34 tflops molecular dynamics simulation for nacl with a special-purpose computer: Mdm. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, 2000.

[15] J. Makino and T. Fukushige. A 11.55 tflops simulation of black holes in a galactic center on GRAPE-6. In *Proceedings of the 2001 ACM/IEEE conference on Supercomputing*, 2001.

[16] 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.

[17] 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.

[18] 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.

[19] 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. ACM, 2009.

[20] T. Hamada and K. Nitadori. 190 tflops astrophysical N-body simulation on cluster of gpus. In *submitted to SC10*, 2010.

[21] J. Barnes and P. Hut. A hierarchical $O(N \log N)$ force-calculation algorithm. *Nature*, 324:446–449, December 1986.

[22] L. Greengard and V. Rokhlin. A fast algorithm for particle simulations. *J. Comput. Phys.*, 73(2):325–348, 1987.

[23] M. S. Warren and J. K. Salmon. A portable parallel particle program. *Computer Physics Communications*, 87:266–290, 1995.

[24] W. Dehnen. A hierarchical $\mathcal{O}(N)$ force calculation algorithm. *J. Comput. Phys.*, 179(1):27–42, 2002.

[25] H. Cheng, L. Greengard, and V. Rokhlin. A fast adaptive multipole algorithm in three dimensions. *J. Comput. Phys.*, 155:468–498, 1999.

[26] 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.

[27] L. Nyland, M. Harris, and J. Prins. Fast n-body simulation with cuda. *GPU Gems*, 3:677–695, 2007.

[28] 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.

[29] E. Gaburov and S. Harfst, S. Portegies Zwart. Sapporo: A way to turn your graphics cards into a grape-6. *New Astronomy*, 14:630–637, 2009.

[30] S. Williams, A. Waterman, and D. Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2008.

[31] J. K. Salmon. *Parallel Hierarchical N-body Methods*. PhD thesis, California Institute of Technology, 1990.

[32] M. S. Warren and J. K. Salmon. A parallel hashed oct-tree N-body algorithm. In *Proceedings of the 1993 ACM/IEEE Conference on Supercomputing*, pages 12–21, New York, 1993. ACM.

[33] J. Dubinski. A parallel tree code. *New Astronomy*, 1:133–147, 1996.

[34] H. Sundar, R. S. Sampath, and G. Biros. Bottom-up construction and 2:1 balance refinement of linear octrees in parallel. *SIAM J. Sci. Comput.*, 30(5):2675–2708, 2008.

[35] A. Rahimian, I. Lashuk, S. Veerapaneni, A. Chandramowlishwaran, D. Malhotra, L. Moon, R. Sampath, A. Shringarpure, J. Vetter, R. Vuduc, D. Zorin, and G. Biros. Petascale direct numerical simulation of blood flow on 200k cores and heterogeneous architectures. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11. IEEE Computer Society, 2010.