Petascale turbulence simulation using a highly parallel fast multipole method

Rio Yokota
Dept. of Mechanical Engineering, Boston University
Boston, MA 02215
yokota@bu.edu

Tetsu Narumi
Dept. of Computer Science, University of Electro-Communications
Tokyo, Japan
narumi@cs.uec.ac.jp

L. A. Barba
Dept. of Mechanical Engineering, Boston University
Boston, MA 02215
labarba@bu.edu

Kenji Yasuoka
Dept. of Mechanical Engineering, Keio University
Yokohama, Japan
yasuoka@mech.keio.ac.jp

Abstract—We present a 0.5 Petaflop/s calculation of homogeneous isotropic turbulence in a cube of $2048^3$ particles, using a highly parallel fast multipole method (FMM) using 2048 GPUs on the TSUBAME 2.0 system. We compare this particle-based code with a spectral DNS code under the same calculation condition and the same machine. The results of our particle-based turbulence simulation match quantitatively with that of the spectral method. The calculation time for one time step is approximately 30 seconds for both methods; this result shows that the scalability of the FMM starts to become an advantage over FFT-based methods beyond 2000 GPUs.

I. INTRODUCTION

For the past few decades, the preferred method for the simulation of homogeneous isotropic turbulence in a periodic cube has been the pseudo-spectral method. The largest direct numerical simulation of isotropic turbulence to date was performed by Ishihara et al. [1], [2] using $4096^3$ grid points at a maximum Taylor-microscale Reynolds number $R_\lambda \approx 1130$. This record-breaking simulation was done on Earth Simulator—a large, shared-memory system that can efficiently perform large-scale FFT. The successive generations of supercomputers have not been so FFT-friendly, and this record has not been broken even though the peak performance of supercomputers has increased nearly 50-fold since then.

Future high-performance computing systems will have ever more nodes, and ever more cores per node, but will probably not be equipped with the bandwidth required by many popular algorithms to transfer the necessary data at the optimal rate. This situation is detrimental to parallel scalability. Therefore, it is becoming increasingly important to look at alternative algorithms that may achieve better performance on these extremely parallel machines of the future.

In most standard methods of incompressible CFD, the greatest fraction of the calculation runtime is spent solving a Poisson equation. Equations of this type can be efficiently solved by means of an FFT-based algorithm, a sparse linear solver, or a fast multipole method (FMM). For the sake of our argument, we will not differentiate between FFT-based Poisson solvers and pseudo-spectral methods because they both rely on FFT. The fast multipole method has been less popular due to the fact that it is substantially slower—depending on implementations, around an order of magnitude slower—than FFT and multigrid solvers when compared using a small CPU cluster. We aim to show with this ongoing study that the relative performance of FMM improves as one scales to large numbers of GPUs.

The highly scalable nature of the FMM algorithm, among other features, makes it a top contender in the algorithmic toolbox for Exascale systems. One point of evidence of this argument is given by the winning Gordon Bell entry of last year [3], which achieved 0.7 Petaflop/s with an FMM algorithm on 200k cores of the Jaguar system at Oak Ridge National Laboratory. In the previous year, FMM also figured prominently at the Supercomputing Conference, with a paper among the finalists for the Best Paper award [4] and the Gordon Bell prize in the price/performance category going to work with hierarchical $N$-body methods on GPU architecture [5].

In fact, the FMM algorithm is well adapted to the architectural features of GPUs, which is an important consideration given that it is considered an architecture of likely dominance as we move towards Exascale. The work presented in 2009—winner in the price/performance category in great measure thanks to the ingenious and resourceful system design using gaming hardware—reported (at the time of the conference) 80 Teraflop/s. [5] This work progressed to an Honorable Mention in the 2010 list of awardees, with 104 Teraflop/s. [6]

At the level of the present work, where we present a 0.5 Petaflop/s calculation of homogeneous isotropic turbulence, the FMM moves firmly into the arena of Petascale GPU computing. The significance of this advance is that we are now in a range where the FMM algorithm shows its true capability. The scalability of FMM using in the order of
1000 GPUs shows advantage over the dominance of FFT-based algorithms. Showcasing the FMM in a simulation of homogeneous isotropic turbulence is then especially fitting, given that a years-old record there remains unchallenged. Based on our results, we are confident that a turbulence simulation repeating the 4096³ record, but with an FMM-based algorithm on GPUs, will be attained soon. Conceivably, we will see this nine-year old record finally broken using GPU hardware.

II. PARALLEL FAST MULTPOLE METHOD ON GPUs

A. Tree Partitioning

When parallelizing hierarchical N-body algorithms, the fact that the particle distribution is dynamic makes it impossible to precompute the decomposition and to balance the work-load or communication a priori. Warren and Salmon [11] developed a parallel algorithm for decomposing the domain into recursive subdomains using the method of orthogonal recursive bisection (ORB). The resulting process can be thought of as a binary tree, which splits the domain into subdomains with equal number of particles at every bisection.

Another popular technique for partitioning tree structures is to use Morton ordering [8], where bits representing the particle coordinates are interleaved to form a unique key that maps to each cell in the tree. Following the Morton index monotonically will take the form of a space filling curve in the shape of a “Z”. Equally partitioning the list of Morton indices assures that each partition will contain an equal number of cells, regardless of the particle distribution.

For an adaptive tree, a naïve implementation of the Morton ordering could result in large communication if the “Z” is split in the wrong place. Sundar et al. [9] proposed a bottom-up coarsening strategy that ensures a clean partition at the coarse level while using Morton ordering. On the other hand, ORB always partitions the domain into rectangular subdomains, and works fine with treecodes. The problem with ORB for FMM usually occurs when one attempts to use square oct-trees with ORB instead of using the rectangular recursive bisection throughout the whole tree. The stack based treewalk described in the following subsection permits the use of rectangular bisections throughout the whole tree, while enabling cell-cell interactions in the FMM with minimum complications.

Our current partitioning scheme is an extension of the ORB, which allows multisections instead of bisections. We developed a parallel version of the “nth-element” algorithm to find the global median (N/2-th element). The same algorithm can be used to find the N/3-th element as well, and therefore the extension to multisections was straightforward. This recursive multisection allows efficient partitioning for number of processes that are not a power of two.

B. Stack Based Treewalk

The \(O(N)\) treewalk [10] enables cell-cell interactions in the traditional treecode framework. Unlike conventional treecodes, the \(O(N)\) treewalk is not performed per target cell. We describe the \(O(N)\) treewalk in Algorithm 1 which calls an internal routine for the interaction of a pair of cells, given in Algorithm 2. First, a pair of cells is pushed into the stack. It is most convenient to start from the root cell although there is no chance of two root cells interacting. For every step in the while-loop, a pair of cells is popped from the stack and the larger cell is subdivided. Then, Algorithm 2 is called to perform either particle-particle (P2P), multipole-particle (M2P), or multipole-local (M2L) interactions. Unlike a pure treecode which calculates only P2P and M2P, or a pure FMM which calculates only P2P and M2L the current algorithm chooses whichever operation is faster. If none of the above operations are worth doing at this stage of the subdivision, the pair is pushed to the stack and will be handled later. It is a trivial matter to run each of these kernels at the beginning of the calculation and to auto-tune the criteria for choosing which kernel is faster for a given pair of cells. This turned out to be extremely useful for auto-tuning the kernel selection on CPUs and GPUs simultaneously.

Algorithm 1 Evaluate()

\[\text{A=B=\text{rootcell}}\]

\[\text{push pair(A,B) into a stack}\]

\[\text{while the stack is not empty do}\]

\[\text{Pop stack to get a pair(A,B)}\]

\[\text{if radius of A > radius of B then}\]

\[\text{for all child cells “a” of cell A do}\]

\[\text{Interact(a,B)}\]

\[\text{end for}\]

\[\text{else}\]

\[\text{for all child cells “b” of cell B do}\]

\[\text{Interact(A,b)}\]

\[\text{end for}\]

\[\text{end if}\]

\[\text{end while}\]

Algorithm 2 Interact(cell A, cell B)

\[\text{if there are very few particles in both cells, or they don’t have child cells then}\]

\[\text{Evaluate particle-particle (P2P)}\]

\[\text{else if cells A and B are well separated then}\]

\[\text{if there are very few particles in cell A, or there are no child cells in cell A then}\]

\[\text{Evaluate multipole-particle (M2P)}\]

\[\text{else}\]

\[\text{Evaluate multipole-local (M2L)}\]

\[\text{end if}\]

\[\text{else}\]

\[\text{Push pair(A,B) into a stack}\]

\[\text{end if}\]

We will give a simple example of what will typically happen when this algorithm is executed. During the initial stages of the while loop in Algorithm 1, the cells will have too many particles to perform P2P, and will be too close to each other.
to perform M2P/M2L, so the pairs will be pushed to the stack. After a few steps, the pairs will become well separated and M2P interactions will be performed. If the distribution of particles is highly non-uniform and the tree has a very uneven structure, some cells will contain very few particles and M2P interactions will take place. At the final stage of the treewalk, the cells will reach the terminal level where there are no more subcells to divide into, and P2P interactions will be performed. Any cell that has not been covered by the M2L and M2P interactions at this point will be handled by P2P interactions. When all the P2P interactions are finished the stack will be empty and Algorithm 1 will terminate. In the end this will result in a similar interaction list as the one described by Cheng et al. [11], but the implementation is much simpler and the algorithm itself is much more general and flexible.

This framework allows $O(N)$ treewalks on not only Morton ordered square oct-trees, but also binary trees with rectangular ORB or any other hierarchical decomposition of the domain. It also enables the hybridization of treecodes and FMMs in the most natural way, and at the same time, adds the capability to auto-tune the kernel selection to optimize the performance on heterogenous architectures.

C. Local Essential Tree

Once the domain is partitioned, each process will own a local portion of the global tree structure. It is then necessary to communicate the local essential tree (LET), which is a significantly smaller subset of the global tree that contains exactly all the information that is necessary to perform the evaluation. Salmon and Warren [7] introduced two key techniques; one for finding which cells to send, and the other for communicating the data efficiently.

The determination of which data to send is related to how the data will be used in the evaluation stage. How the data will be used can be predicted by redefining the multipole acceptance criteria (MAC) as the distance from the cell in the other process to the edge of the domain of the current process. This way the tree is traversed only once per process, which is a negligible cost compared to the actual tree traversal that is done per leaf cell. This method can be applied directly to our stack-based treewalk approach, since the LET itself is identical to a standard treecode.

The communication of the LET requires global communication and is a potential bottleneck for the scalability of hierarchical N-body algorithms. In order to communicate LET efficiently, the bisectors in the ORB can be used to repeatedly exchange the data required on the other side of the bisector. This N-D hypercube or All-reduce-like communication is used in state-of-the-art treecodes [5] and FMMs [4] to this day. The present implementation combines this technique with the recursive multissection described in section [1-3] to allow the same efficient communication when the number of processes is not a power of two. We also extend this entire framework to the case of periodic boundary conditions.

D. Periodic FMM

The FMM originally solves potential problems with free-field boundary conditions. However, the method can be extended to handle periodic boundary conditions by placing periodic images around the original domain and clustering them into large cells, as shown in Fig. 1. There are two reasons why periodic FMMs add almost no computational overhead to the original FMM. The first is the fact that further domains are clustered into larger and larger cells, so the extra cost of considering another layer of periodic images is constant. The second reason is that only the sources need to be duplicated and the evaluation points exist only in the original domain. Since the work load for considering the periodicity is independent of the number of particles, it becomes negligible as the problem size increases. We have confirmed in an earlier study that periodic boundary conditions adds 3% more calculation time for a million particles. [13]

The current implementation of the periodic FMM uses each bit of an integer to store the information of the periodic images. For convenience, let us refer to this integer as the “periodic flag”. When considering the 26 periodic images surrounding the original domain, one traverses the tree 27 times (including the original domain) and flips the bit of the periodic flag if it satisfies the MAC. In the current FMM, the tree traversal is separated from the evaluation. Interactions are queued and evaluated in one batch after the tree traversal is finished because this is an efficient model for evaluating the kernels on the GPU as we will describe in section [1-2] Once all 27 tree traversals are finished, the kernels are evaluated in the same manner as a non-periodic FMM, but the loop over the source cells is performed 27 times with a conditional statement for the periodic flag inside. The use of a periodic flag results in minimum modification to the tree construction, traversal, LET communication, and evaluation for parallel periodic FMM on GPUs. We also make use of the C++ STL map container to store the periodic flags with the cell index as the key so that
collisions of periodic flags for the same cell are automatically resolved.

E. GPU kernels

Our hybrid hierarchical N-body algorithm has 7 different kernels, as shown in Fig. 2. All of these kernels are evaluated on the GPU. Out of these 7 kernels, a majority of the time is spent on the P2P, M2L, and M2P kernels. These three kernels are evaluated in one batch after the tree traversal is completed, since there is no data dependency between these three kernels. This batch evaluation can be broken down into multiple calls to the GPU depending on the data size. Therefore we were able handle problem sizes of up to 100 million on a single process if the memory on the host machine was large enough.

As an example of the usage of thread blocks in the GPU execution model, we show an illustration of the M2L kernel on GPUs in Fig. 3. Each coefficient of the multipole/local expansion is mapped to a thread on the GPU, and each target cell is mapped to a thread block while each source cell is loaded to shared memory and evaluated sequentially. All other kernels are mapped to the threads and thread blocks in a similar manner.

III. INTEGRATION OF FMM INTO TURBULENCE SIMULATIONS

A. Vortex Method

The vortex method [15] is a particle based method for fluid dynamics simulations. The particle based discretization allows the continuum physics to be solved as a N-body problem. Therefore, the hierarchical N-body methods that extract the full potential of GPUs can be used for the simulation of turbulence. Unlike other particle based fluid dynamics solvers e.g. SPH[16], the vortex method is especially well suitable for solving turbulence, because the vortex interactions seen in turbulent flows are exactly what it calculates using the interacting vortex particles.

Since the vortex method is neither a standard method for simulating turbulence or a standard application for FMMs, we will give a brief explanation of the method itself. In the vortex method, the Navier-Stokes equation is solved in the velocity-vorticity form, and discretized with vortex particles. The velocity is calculated by

\[ u_i = \sum_{j=1}^{N} \alpha_j g_\sigma \times \nabla G \]  

where \( \alpha \) is the strength of vortex particles. \( G = \frac{1}{4\pi r_{ij}} \) is the Green’s function for the Laplace equation and

\[ g_\sigma = \text{erf} \left( \sqrt{\frac{r_{ij}^2}{2\sigma_j^2}} \right) - \sqrt{\frac{4}{\pi}} \frac{r_{ij}^2}{2\sigma_j^2} \exp \left( -\frac{r_{ij}^2}{2\sigma_j^2} \right), \]  

is the cutoff function, where \( r \) is the distance between the interacting particles, and \( \sigma \) is the standard deviation of the Gaussian function. The vorticity equation is solved in a fractional step manner by calculating the stretching

\[ \frac{D\alpha_i}{Dt} = \sum_{j=1}^{n} \alpha_j \nabla (g_\sigma \times \nabla G) \cdot \alpha_i. \]  

and the diffusion

\[ \sigma^2 = 2\nu t \]  

separately. We perform a radial basis function interpolation for reinitialized Gaussian distributions to ensure the convergence of the diffusion calculation.

Equations (1) and (3) are N-body interactions, and are evaluated using the FMM. The flow of the entire vortex method is shown in Fig. 4. First, the domain is partitioned using the recursive multisection described in section II-A. Then the
FMM kernels for the Local Tree are evaluated while the LET is being communicated in a separate openmp section. After the LET is communicated, the FMM kernels are evaluated again for the remaining parts of the LET. Subsequently, the position, vortex strength $\alpha$, and core radius $\sigma$ are updated locally. This information is communicated in the next round when the LET is exchanged. Furthermore, in Lagrangian CFD simulations it is necessary to reinitialize the particle positions so the distribution of particles remains somewhat even. This actually allows us to reuse the same tree structure since the particles are reinitialized to the same position every time. Therefore, the partitioning is performed only once in this simulation.

**B. Spectral Method**

For the spectral method calculation, we used a standard code for homogeneous isotropic turbulence called HIT3D. The code is available from google code and uses a spectral Galerkin method with primitive variable formulation with pseudo-spectral methods to compute the convolution sums. A schematic of the parallelization of the FFT is shown in Fig. 5. The aliasing was removed by the 2/3-rule and the time integration was performed by a 2nd order Adams-Bashforth method. No forcing was applied since it would be difficult to do so with vortex methods.

The initial condition was generated in Fourier space as a solenoidal isotropic velocity field with random phases and a prescribed energy spectrum. This initial velocity field has a Gaussian PDF and satisfies the incompressibility condition. The vortex method uses the same initial condition by first calculating the vorticity field in physical space, and then using radial basis function interpolation to obtain the vortex strengths.

**C. Calculation of Isotropic Turbulence**

The calculation of homogeneous isotropic turbulence was performed for an initial Reynolds number of $Re_\lambda \approx 500$. The calculation domain is $[-\pi, \pi]$ and has periodic boundary conditions in all directions, and the grid size was $2048^3$. This results in an N-body simulation of a total of $8 \times 10^{10}$ particles. The total number of GPUs used was 2048. For the periodic FMM the number periodic images was $3^3$ in each dimension. The order of multipole expansion was set to $p = 14$ to achieve the required accuracy for the present application.

The isosurface of the second invariant of the velocity gradient tensor is shown in Fig. 6. This is a snapshot at the early stages of the simulation and we do not observe any large coherent structures. In order to take a closer look at the quantitative aspects of the vortex simulation, we compare the kinetic energy spectrum with that of the spectral method in Fig. 7 where $T$ is the eddy turnover time. It is quite clear that we have an excellent quantitative agreement between the vortex method and spectral method. Therefore, we conclude that our FMM based particle method is capable of simulating the same scale of turbulence as the spectral method with the
Fig. 6: Isosurface of the second invariant of the velocity gradient tensor

Fig. 7: Kinetic Energy Spectrum at $t/T=2$

same spatial resolution. We will turn our attention to the speed of the calculation in the next section.

IV. SCALABILITY RESULTS

A. Hardware

The present calculations are run on the TSUBAME 2.0 system, which has 1408 nodes equipped with 12-core Westmere-EP 2.93GHz CPUs, 3 NVIDIA M2050 GPUs, 54 GB (96 GB on 41 nodes) of RAM, and 120 GB (240 GB on 41 nodes) of local SSD storage. Each computing node is interconnected with the InfiniBand device Grid Director 4700 developed by Voltaire Inc., with non-blocking and full bisectional bandwidth. Each node has $2 \times 40$ Gbps bandwidth, and the bisection bandwidth of the system is over 200 Tbps. The total number of M2050 GPUs in the system is 4224, and the peak performance of the entire system is 2.4 Petaflop/s.

B. Weak Scaling

The results of the weak scaling test with 4 million particles per process is shown in Fig. 8. The local evaluation is the P2P kernel evaluation in Fig. 2 and the FMM evaluation is the sum of all the other kernel evaluations. The MPI communication is overlapped with the kernel evaluations but we decided to show the time for the communication by subtracting an equivalent area from the FMM evaluation time. Therefore, the total height of the bar correctly represents the total wall clock time of the calculation. The GPU communication time is the time spent on the cudaMemcpy. We plan to reduce this portion in the future by performing asynchronous memory transfers with double buffering. Finally, the tree construction represents the time required to construct the tree and sort the particles. From Fig. 8 we see that the current FMM is able to completely hide the communication up to 2048 GPUs.

The parallel efficiency of the weak scaling test is shown in Fig. 9. We also performed a similar weak scaling test for the spectral method. The parallel efficiency of the FMM is 72% at 2048 GPUs, while the parallel efficiency of the spectral method is 27% on 2048 CPUs. The bottleneck of the spectral method is the all-to-all communication for transposing the slabs into pencils as shown in Fig. 5. Even though this is not the best implementation of a parallel FFT, the difference in the scalability between the spectral method and FMM is quite obvious. We would also like to note that the actual calculation time for one time step was approximately 30s for both the FMM and spectral method for 2048 processes. Therefore, the superior scalability of the FMM has merely closed the gap. However, we anticipate that this trend will affect the algorithm of choice if we were to move to architectures with higher degree of parallelism.
Fig. 9: Comparison of Weak Scaling between FMM and spectral method

C. Performance Highlights

The present calculation is dominated by the floating point operations in the particle-particle interaction and all other parts are negligible in terms of Flops. Two separate equations are calculated for the particle-particle interaction; the Biot-Savart equation (1) and the Stretching equation (3). The CUDA source code for the Biot-Savart and Stretching kernel are shown in Listings 1 and 2, respectively.

The equation to calculate the Flops for the P2P kernel is

\[
Flops = \frac{(Processes) \times (Target\ particles\ per\ process) \times (Source\ cells\ per\ target) \times (Source\ particle\ per\ cell) \times (Flops\ per\ interaction)}{(Wall\ clock\ time)}
\]

\[
= \frac{2048 \times (4 \times 10^8) \times 19 \times 488 \times 174}{26.8} \times 4.93 \times 10^{14} = 493\ Teraflop/s
\]

Compared to last years peak performance winner (0.7 Petaflop/s), we are able to achieve similar performance (0.5 Petaflop/s) on a problem size that is an order of magnitude smaller (90 billion vs. 8.6 billion), on a GPU cluster using only 1/100 the number of MPI processes (200,000 AMD cores vs. 2,000 GPUs).

Compared to the price/performance winner of 2009 and Honorable mention for price/performance of 2010, we are able to achieve 5 times more Flop/s on 3.5 times more GPUs.

The present vortex method simulation of 8.6 billion particles is also the largest particle-based CFD simulation to date as far as the authors are aware.

The full system of TSUBAME 2.0 offers 4224 GPUs, but we were only able to use about half of the nodes at this time. We hope to perform a full node run by the time of the conference and we expect to exceed a PetaFlop with further tuning. The tuning will include overlap of GPU buffering/communication with GPU computations by using asynchronous data transfers and double buffering. Also, the full node run will allow us to perform a 4096^3 particle run instead of the current 2048^3, which will most likely lead to better scaling.

V. Conclusions

This work represents several milestones. Although the FMM algorithm has been taken to Petascale before (notably, with last year’s winner of the Gordon Bell prize), the present work represents the first time that this is done on GPU architecture. Also, to our knowledge, the present work is the largest direct numerical simulation with vortex methods to date, with 8.6 billion particles used in the cubic volume. Yet another significant event is reaching a range where the highly scalable FMM starts showing advantage over FFT-based algorithms.

### Listing 1: P2P kernel for a single Biot-Savart interaction

```c
__device__ inline void BiotSavartP2P_core(float *target, float *targetX, float *sourceShrd, float3 d, int i) {
    d.x += targetX[0];
    d.x -= sourceShrd[7*i+0];
    d.y += targetX[1];
    d.y -= sourceShrd[7*i+1];
    d.z += targetX[2];
    d.z -= sourceShrd[7*i+2];
    const float SQRT4PI = M_2_SQRTPI;
    const float FOURPI = 0.25 * M_1_PI;
    float R2 = d.x * d.x + d.y * d.y + d.z * d.z + EPS2;
    float SQRT_R2_1 = rsqrtf(R2);
    float RS = R2 * sourceShrd[7*i+6];
    float SQRT_RS = sqrtf(RS);
    float z = SQRT_RS, t, r;
    t = 1.0f / (1.0f + 0.5f*(z));
    r = (t)*expf(-(z)*(z)-1.26551223f + (t)*(1.00002368f+(t)*(0.37409196f+(t)*(0.09678418f+(t)*(-0.18628806f+(t)*(0.27886807f+(t)*(-1.13520398f+(t)*(1.48851587f+(t)*(-0.82215223f+(t)*0.17087277f)))))))));
    float cutoff = FOURPI * SQRT_R2_1 * SQRT_R2_1 * (1.0f - r - SQRT4PI * SQRT_RS * expf(-RS));
    target[0] += (d.y * sourceShrd[7*i+5] - d.z * sourceShrd[7*i+4]) * cutoff;
    target[1] += (d.z * sourceShrd[7*i+3] - d.x * sourceShrd[7*i+5]) * cutoff;
    target[2] += (d.x * sourceShrd[7*i+4] - d.y * sourceShrd[7*i+3]) * cutoff;
}
```

### Table I: Floating point operations per interaction

| Operation | Biot-Savart | Stretching |
|-----------|-------------|------------|
| +         | 19          | 25         |
| -         | 14          | 18         |
| /         | 1           | 1          |
| sqrtf     | 1           | 1          |
| rsqrtf    | 1           | 1          |
| expf      | 2           | 2          |
| Total     | 70          | 104        |
With a 0.5 Petaflop/s calculation of isotropic turbulence in a 2048³ box, using 2000 GPUs, we are within reach of a turning point. The combination of application, algorithm, and hardware used are also notable.

VI. ACKNOWLEDGMENTS

Computing time in the TSUBAME 2.0 system was made possible by the Grand Challenge Program of TSUBAME. LAB acknowledges partial support from NSF grant OCI-0946441, ONR award #N00014-11-1-0356, and Boston University College of Engineering.

REFERENCES

[1] M. Yokokawa, K. Itakura, A. Uno, T. Ishihara, and Y. Kaneda, “16.4-tflops direct numerical simulation of turbulence by a fourier spectral method on the earth simulator,” in Proceedings of the IEEE/ACM SC2002 Conference, 2002.

[2] T. Ishihara, Y. Kaneda, M. Yokokawa, K. Itakura, and A. Uno, “Small-scale statistics in high-resolution direct numerical simulation of turbulence: Reynolds number dependence of one-point velocity gradient statistics,” Journal of Fluid Mechanics, vol. 592, pp. 335–366, 2007.

[3] A. Rahimian, I. Lashuk, S. K. Verrapaneni, 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 Conference on High Performance Computing Networking, Storage and Analysis, 2010.

[4] I. Lashuk, A. Chandramowlishwaran, H. Langston, T.-A. Nguyen, R. Sampath, A. Shringarpure, R. Vuduc, L. Ying, D. Zorin, and G. Biros, “A massively parallel adaptive fast multipole method on heterogeneous architectures,” in Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, 2009.

[5] T. Hamada, K. Nitadori, K. Benkrid, Y. Ohno, G. Morimoto, T. Masada, Y. Shibata, K. Oguri, and M. Taiji, “A novel multiple-parallel walk algorithm for the barnes-hut treecode on gpus- towards cost effective, high performance n-body simulation,” Computer Science - Research and Development, vol. Special Issue Paper, pp. 1–11, 2009.

[6] T. Hamada and K. Nitadori, “190 tflops astrophysical n-body simulation on cluster of gpus,” in submitted to SC10, 2010.

[7] 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, 1992, pp. 570–576.

[8] ———, “A parallel hashed oct-tree n-body algorithm,” in Proceedings of the 1993 ACM/IEEE conference on Supercomputing, 1993, pp. 12–21.

[9] H. Sundar, R. S. Sampath, and G. Biros, “Bottom-up construction and 2:1 balance refinement of linear octrees in parallel,” SIAM Journal on Scientific Computing, vol. 30, no. 5, pp. 2675–2708, 2008.

[10] W. Dehnen, “A hierarchical on-node force calculation algorithm,” Journal of Computational Physics, vol. 179, no. 1, pp. 27–42, 2002.

[11] H. Cheng, L. Greengard, and V. Rokhlin, “A fast adaptive multipole algorithm in three dimensions,” Journal of Computational Physics, vol. 155, no. 2, pp. 468–498, 1999.

[12] C. G. Lambert, T. A. Darden, and A. R. Board, “A multipole based algorithm for efficient calculation of forces and potentials in macroscopic periodic assemblies of particles,” Journal of Computational Physics, vol. 126, no. 2, pp. 274–285, 1996.

[13] R. Yokota, T. K. Sheel, and S. Obi, “Calculation of isotropic turbulence using a pure lagrangian vortex method,” Journal of Computational Physics, vol. 197, pp. 1–11, 2005.

[14] R. Yokota, J. P. Bardhan, M. G. Knepley, L. A. Barba, and T. Hamada, “Biomolecular electrostatics using a fast multipole bem on up to 512 gpus and a billion unknowns,” Computer Physics Communications, vol. 182, pp. 1272–1283, 2011.

[15] P. D. Koumoutsakos and G.-H. Cottet, Vortex Methods -Theory and Practice-. Cambridge University Press, 2000.

[16] R. A. Gingold and J. J. Monaghan, “Smoothed particle hydrodynamics: Theory and application to non-spherical stars,” Monthly Notices of the Royal Astronomical Society, vol. 181, pp. 375–389, 1977.