607 An Implementation of Low-Frequency Fast Multipole BIEM for Helmholtz' Equation on GPU

Toru Takahashi, Department of Mechanical Science and Engineering, Nagoya University
Cris Cecka, Institute for Computational and Mathematical Engineering, Stanford University
Eric Darve, Department of Mechanical Engineering, Stanford University

Key Words : Boundary Integral Equation Method, Fast Multipole Method, M2L, Helmholtz, GPU, CUDA

1. Introduction

Acceleration of the fast multipole method (FMM), which is the fast and approximate algorithm to compute the pairwise interactions among many bodies, with graphics processing units (GPUs) has been investigated for the last couple of years. In view of the type of kernel functions, the non-oscillatory kernels (especially, the Laplace kernel) were studied by many researchers (*e.g.* Gumerov[1]), and then the Helmholtz kernel in low-frequency regime was recently pioneered by Cwikla[2].

As well as Cwikla, we aim to implement the low-frequency FMM in three dimensions (3D) with NVIDIA's CUDA-capable GPUs, which are massively multithreaded coprocessors, and then develop the fast multipole boundary integral equation method (FMBIEM)[3] for Helmholtz' equation on the GPUs. In this paper, following the study on the GPU-accelerated black-box FMM (bbFMM)[4, 5], we present efficient implementations of the multipole-to-local (M2L) operation, which is the most time-consuming part in the FMBIEM and thus worth being accelerated by GPU.

This paper organizes as follows: Sections 2 and 3 give the overview of the BIEM and FMBIEM, respectively. Section 4 addresses the implementation of the M2L operation for the FMBIEM on GPU. Section 5 shows the numerical results.

2. BIEM for Helmholtz' Equation in 3D

2·1 Statement of Problem Suppose a homogeneous domain $D \subset \mathbb{R}^3$ having a smooth boundary $S := \partial D$ with the unit outward normal $\boldsymbol{n}$. Then we solve $u : \mathbb{R}^3 \to \mathbb{C}$ from the 3D Helmholtz' equation

$$\Delta u(\boldsymbol{x}) + k^2 u(\boldsymbol{x}) = 0 \quad \text{for } \boldsymbol{x} \in D \tag{1}$$

subject to appropriate boundary conditions (typically, u or $q := \partial u/\partial n$ is given at any point of S) and, if D is infinite, the radiation condition at infinity. Here, $k \in \mathbb{R}$ stands for the wavenumber given *a priori*.

2·2 Boundary Integral Equation As well known, (1) yields the following boundary integral equation (BIE):

$$\frac{1}{2}u(\boldsymbol{x}) + \int_S \left(\frac{\partial \Gamma}{\partial n_y}(\boldsymbol{x},\boldsymbol{y})u(\boldsymbol{y}) - \Gamma(\boldsymbol{x}-\boldsymbol{y})q(\boldsymbol{y}) \right) dS_y = 0 \quad \text{for } \boldsymbol{x} \in S, \tag{2}$$

where $\Gamma(\boldsymbol{r}) := e^{ik|\boldsymbol{r}|}/(4\pi|\boldsymbol{r}|)$ ($\boldsymbol{r} \in \mathbb{R}^3$; i is the imaginary unit) denotes the fundamental solution of (1).

2·3 Discretization We discretize (2) with N triangular piecewise-constant elements. We solve the resulting linear equations of order N with an iterative method because N of interest is quite large.

3. Low-Frequency FMBIEM

If k is sufficiently small, we can apply the low-frequency FMM to evaluate the RHS (layer potential) of (2) with $O(N)$ computational complexity[3]. By combining the FMM to the iterative solver, we can construct an $O(mN)$ fast BIEM, where m stands for the number of iterations required for convergence. For this low-frequency FMBIEM, let us derive the M2L operation, which is the prime target of this study. We follow the notations of Yoshida[6].

3·1 Multipole Expansion We can expand Γ as

$$\Gamma(\boldsymbol{x}-\boldsymbol{y}) = \frac{ik}{4\pi} \sum_{n=0}^{\infty} \sum_{m=-n}^{n} (2n+1)\mathcal{O}_n^m(\boldsymbol{x}-\boldsymbol{s})\overline{\mathcal{I}_n^m(\boldsymbol{y}-\boldsymbol{s})}$$

if $|\boldsymbol{x}-\boldsymbol{s}| > |\boldsymbol{y}-\boldsymbol{s}|$ holds, where $\mathcal{O}_n^m(\boldsymbol{x}) := h_n^{(1)}(k|\boldsymbol{x}|)Y_n^m(\hat{\boldsymbol{x}})$ and $\mathcal{I}_n^m(\boldsymbol{x}) := j_n(k|\boldsymbol{x}|)\ Y_n^m(\hat{\boldsymbol{x}})$. Here, $\hat{\boldsymbol{x}} := \frac{\boldsymbol{x}}{|\boldsymbol{x}|}$, $h_n^{(1)}$ is the Hankel function of the first kind and nth order, j_n is the spherical Bessel function of the nth order, and Y_n^m is the spherical harmonics of the nth order of degree m.

Let S_o and S_s be non-overlapping parts of S (Fig.1), and $\boldsymbol{o}$ (resp. $\boldsymbol{s}$) be a point close to S_o (resp. S_s). From the expansion of Γ, the layer potential with respect to S_s can be expressed as

$$\int_{S_s} \left(\frac{\partial \Gamma}{\partial n_y}(\boldsymbol{x},\boldsymbol{y})u(\boldsymbol{y}) - \Gamma(\boldsymbol{x}-\boldsymbol{y})q(\boldsymbol{y}) \right) dS_y = \frac{ik}{4\pi} \sum_{n=0}^{\infty} \sum_{m=-n}^{n} (2n+1)\mathcal{O}_n^m(\boldsymbol{x}-\boldsymbol{s})M_n^m(\boldsymbol{s}) \tag{3}$$

if $|\boldsymbol{x}-\boldsymbol{s}| > |\boldsymbol{y}-\boldsymbol{s}|$ for $\boldsymbol{x} \in S_o$ and $\boldsymbol{y} \in S_s$, where

$$M_n^m(\boldsymbol{s}) := \int_{S_s} \left(\frac{\partial}{\partial n_y}\overline{\mathcal{I}_n^m(\boldsymbol{y}-\boldsymbol{s})}u(\boldsymbol{y}) - \overline{\mathcal{I}_n^m(\boldsymbol{y}-\boldsymbol{s})}q(\boldsymbol{y}) \right) dS_y \tag{4}$$

is the multipole moment with respect to $\boldsymbol{s}$.

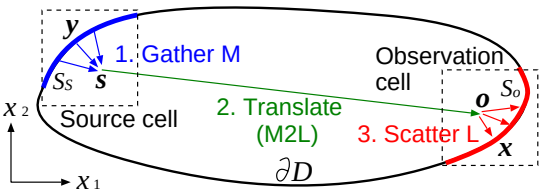

Fig. 1 Far-field evaluation of the layer potential for S_s via three steps of the FMM (in 2D for illustration purposes).

3·2 Local Expansion Expanding (3) in the Taylor series with respect to $\boldsymbol{o}$, we can obtain the following local expansion of the layer potential:

$$\text{Eq.(3)} = \frac{ik}{4\pi} \sum_{n=0}^{\infty} \sum_{m=-n}^{n} (2n+1)\overline{\mathcal{I}_n^m(\boldsymbol{x}-\boldsymbol{o})}L_n^m(\boldsymbol{o}),$$

where

$$L_n^m(\boldsymbol{o}) = \sum_{n'=0}^{\infty} \sum_{m'=-n'}^{n'} \mathcal{T}_{n,m,n',m'}(\boldsymbol{o}-\boldsymbol{s}) M_{n'}^{m'}(\boldsymbol{s}) \qquad (5)$$

denotes the local coefficient with respect to $\boldsymbol{o}$. See Yoshida[6] for the definition of the coefficients $\mathcal{T}_{n,m,n',m'}$.

In the numerical analysis, the summation over n' is truncated to $p-1$ according to the desired accuracy.

4. Implementation of M2L on GPU

4·1 Reduction to Matrix-Vector Products Eq.(5) gives the M2L operation for the FMBIEM. In terms of the FMM hierarchy (octree), $\boldsymbol{o}$ and $\boldsymbol{s}$ in (5) express the centers of an observation cell O and its source cell S in the interaction-list of O, denoted by $\mathcal{I}(O)$, respectively. Replacing the double indices (m,n) (resp. (m',n')) with a single index i (resp. j), we can reduce (5) to the sum of at most 189 matrix-vector products:

$$L_i(\boldsymbol{o}) = \sum_{S\in\mathcal{I}(O)} \sum_{j=0}^{p^2-1} \mathcal{T}_{ij}(\boldsymbol{o}-\boldsymbol{s}) M_j(\boldsymbol{s}) \quad \text{for } 0 \le i < p^2, \qquad (6)$$

where $\mathcal{T}_{ij}$ represents a $p^2 \times p^2$ dense matrix.

4·2 Application of M2L-Schemes We can apply the M2L-schemes proposed for the bbFMM[5] to accelerate the computation of (6), because it has the same form as the M2L computation for the bbFMM. We briefly overview the proposed schemes.

Scheme 1 (Basic) For a given level of the hierarchy, we assign one thread-block to one observation cell O and use p^2 threads for each thread-block. In each thread-block, the ith thread computes the ith row of the vector L_i in (6).

In this approach, we can expect a good load balance among both thread-blocks and threads. Meanwhile, the primary bottleneck is memory transfers of $\mathcal{T}_{ij}$ for the device-memory for GPU.

Scheme 2 (Sibling blocking) In order to reduce the data traffics, we perform (6) at the grain of *clusters* instead of cells, where a cluster is defined by a group of eight *sibling cells*, which share the same parent cell. A pair of adjacent clusters generally includes 8×8 interactions (matrix-vector products), among which we observe that two or more interactions can typically be associated with a common M2L-transfer vector. Thus, we can combine the matrix-vector products into a single matrix-matrix product, in which the $\mathcal{T}_{ij}$ can be reused. As a result, the data traffic for $\mathcal{T}_{ij}$ is nearly reduced by half.

Scheme 3 (Cluster blocking) We can further reduce the traffic of $\mathcal{T}_{ij}$ by sharing it within groups of clusters. Such a group of $B \times B \times B$ clusters is referred to as a *chunk* of size B. The traffic is B^3 times less than the second scheme.

Scheme 4 ("ij" blocking) This scheme uses a large-scale blocking of cells but in a manner different from the third one. Essentially, we chose as inner loop the interaction-list loop so that we can reduce the traffic of $\mathcal{T}_{ij}$ significantly.

We assign one thread-block to one chunk G, and one thread to one cell O in G. First, for a given column index j, all the threads share the vectors M_j for all the cells S that enclose G. Next, for a given row index i, all the threads share $\mathcal{T}_{ij}$ for all the possible vectors $\boldsymbol{o}-\boldsymbol{s}$. Finally, using the shared data, each thread computes $\sum_{S\in\mathcal{I}(O)} \mathcal{T}_{ij}(\boldsymbol{o}-\boldsymbol{s}) M_j(\boldsymbol{s})$. Last, the result is accumulated into L_i. This process is repeated for all i and j.

4·3 Adaptation to FMBIEM We give two remarks on importing the aforementioned schemes from the bbFMM to the FMBIEM.

1. The (complex) double-precision floating-point arithmetic is indispensable instead of the (real) single-precision arithmetic. This is because the values of multipole moments in (4) can exceed the maximum number for single-precision when the order is high or the size of cells is small.

   Although the algorithms of the schemes are independent of the arithmetics, the double-precision arithmetic can affect the implementations (thus performances) of the schemes because of its high memory requirement, particularly for shared memory and registers on the GPU.

2. The adaptive octree is necessary because the distribution of boundary elements is usually non-uniform in space. This does not matter to the first scheme because the M2L operations are performed in the grain of cells. Meanwhile, the other schemes may degenerate their performances in case of non-uniform distribution. This is because the efficiency to reuse or share the common data can be low due to the possible presence of *empty* cells in a cluster or chunk.

4·4 CPU and GPU Codes We developed a multi-level FMBIEM program for shared memory machines with multiple CPUs (cores). We call this program as *CPU code*, from which we developed the *GPU code* that can execute the M2L operation (6) with the help of a GPU.

As of now, we have applied the first and second M2L-schemes to both codes, where we used $B = 2$ for the second scheme. We refer, for example, the CPU code using the first scheme to as CPU1.

5. Numerical Test

We compared the four codes, that is, CPU1, GPU1, CPU2, and GPU2 with respect to performance and accuracy.

5·1 Setup We solved an external scattering problem, in which the scatterer is a rigid (*i.e.* $q = 0$ on S) and unit sphere that is irradiated by a plane wave $u^{\text{in}}(\boldsymbol{x}) = e^{ikx_3}$. Here, we used $k = 8$; the wave length λ is then $\frac{\pi}{4}$, thus the diameter of the sphere is about 2.6λ.

The number of boundary elements (N) to discretize the sphere was 20,480 to 1,310,720 (8 cases). For the

element integrals, the number of Gaussian quadrature points per element was set to 3.

The octree was built so that the maximum number of boundary elements per leaf was less than 100. Accordingly, the maximum (finest) level of octree ranged from 4 to 7 for the above Ns.

In order to solve the linear algebraic equations resulting from the BIE (2), the GMRES with the point-Jacobi preconditioner was used. The initial guess was set to zero and the iteration was stopped when the relative residual was less than 10^{-5}.

We used a Dell Poweredge with a quad-core CPUs (Intel Xeon E5345 2.33 GHz) and a CUDA-capable GPU (NVIDIA GeForce GTX 275). In double precision, the theoretical peak performance of the CPU is 37 Gflop/s, while that of the GPU is about 80 Gflop/s.

In this benchmark, we considered two accuracy levels by setting the truncation number (p) to 8 and 16, respectively. Notice that GPU2 is currently unavailable for $p = 16$ because of the shortage of shared memory on the GPU.

5·2 Accuracy The error of the four codes are plotted in Fig.2. Here, the error is defined by $\max_{0 \le i < N} \left| u_i - u_i^{\text{exact}} \right|$, where u_i and u_i^{exact} are the numerical and exact solutions, respectively, for the ith boundary element. As we had expected, the same accuracy was obtained by different codes and different schemes for each p and that the accuracy improved as p increased for each N.

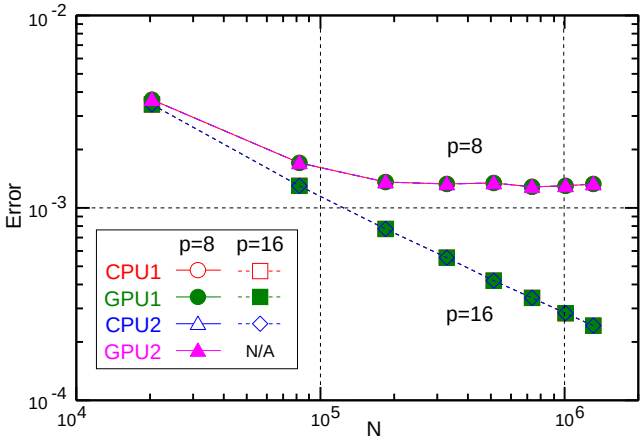

Fig. 2 Computational error.

5·3 Timing Fig.3 shows the total computational time. GPU1 and GPU2 outperformed CPU1 and CPU2, but their speedups were small. Note that the GMRES converged in 7 iterations in any N, p, and codes.

5·4 Performance of M2L Fig.4 shows the performance of the M2L operation in Gflop/s, in which we are primarily interested. Indeed, the GPU could accelerate the M2L operations. Regarding to GPU code, the second scheme was not advantageous to the first scheme at least $p = 8$ in this test.

6. Conclusion

We investigated the acceleration of the M2L operation for the low-frequency FMBIEM with a CUDA-capable GPU, following the two schemes proposed for the bbFMM [5]. In the numerical test, the performance of the M2L operation was improved by the GPU, though the overall

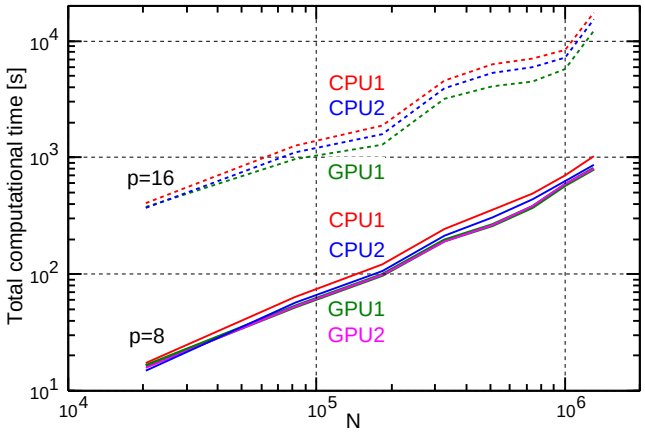

Fig. 3 Total computational time.

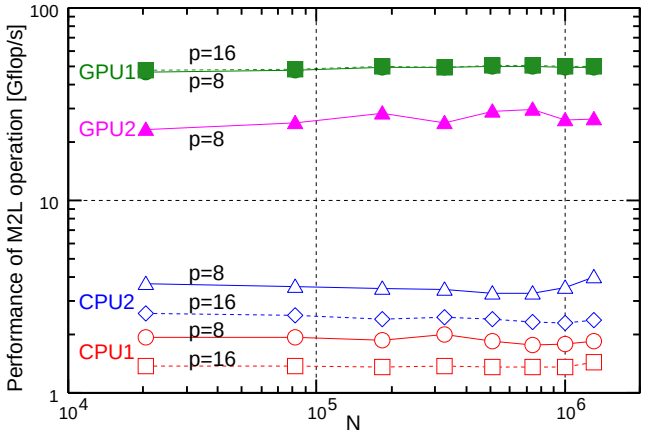

Fig. 4 Performance of M2L operation.

runtime was not reduced significantly. Hence, we will need to implement other routines (*e.g.* nearby interactions) on GPU, exploring more efficient implementations/schemes of the M2L operation in succession. In addition, it is worthwhile to optimize our program for the new GPUs, *i.e.* NVIDIA's Fermi, the double-precision performance of which is improved over GTX 200.

Acknowledgments This work was supported by MEXT KAKENHI (22760062) and the Hori Information Science Promotion Foundation.

References

(1) N. Gumerov, R. Duraiswami, Fast multipole methods on graphics processors, *J. Comput. Phys.*, **227**, 8290–8313, 2008.

(2) M. Cwikla, J. Aronsson, V. Okhmatovski, Low-frequency MLFMA on graphics processors, *IEEE Antennas and Wireless Propagation Letters*, **9**, 8–11, 2010.

(3) N. Nishimura, Fast multipole accelerated boundary integral equation methods, *Appl. Mech. Rev.*, **55**, 299–324, 2002.

(4) W. Fong, E. Darve, The black-box fast multipole method, *J. Comput. Phys.*, **228**, 8712–8725, 2009.

(5) T. Takahashi, C. Cecka, W. Fong, E. Elsen, E. Darve, The black-box fast multipole method on graphical processing units, *Proc. the 10th US National Congress on Computational Mechanics*, CD-ROM, 2009.

(6) K. Yoshida, Application of fast multipole method to boundary integral equation method, Doctoral dissertation (Japan), Kyoto University, 2001. http://gspsun1.gee.kyoto-u.ac.jp/yoshida/doctoral_thesis/index.html.