Large Inverse-Scattering Solutions with DBIM on GPU-Enabled Supercomputers

Mert Hidayetoğlu, Carl Pearson, Weng Cho Chew, Levent Gürel, and Wen-Mei Hwu
Department of Electrical and Computer Engineering
University of Illinois at Urbana-Champaign, Urbana, Illinois 61801, U.S.A.

***Abstract*—We report inverse-scattering solutions on supercomputers involving large numbers of graphics processing units (GPUs). The distorted-Born iterative method (DBIM) is employed for the iterative inversions. In each iteration, the required forward problems are distributed among computing nodes equipped with GPUs, and solved with the multilevel fast multipole algorithm. A tomographic reconstruction of a synthetic object with a linear dimension of one hundred wavelengths is obtained on 256 GPUs. The results show that DBIM obtains images approximately four times faster on GPUs, compared to parallel executions on traditional CPU-only computing nodes.**

I. Introduction

Linear diffraction tomography techniques are inadequate for imaging when the multiple-scattering effects are strong. In that case, the scattered field is a nonlinear function of the unknown object, and therefore a nonlinear optimization is required for finding an object which minimizes the residual field at the receivers. The distorted-Born iterative method (DBIM) performs this optimization by solving

$$[\mu\overline{\boldsymbol{I}} + \overline{\boldsymbol{F}}^{\dagger} \cdot \overline{\boldsymbol{F}}] \cdot \delta\boldsymbol{O} = \overline{\boldsymbol{F}}^{\dagger} \cdot \boldsymbol{b} \tag{1}$$

in each iteration [1]. Here, $\overline{\boldsymbol{F}}$ is the functional derivative of the scattered field with respect to the object, and is formulated with the distorted-Born approximation. Each step is taken with the solution of (1), and then $\overline{\boldsymbol{F}}$ is updated with the new background object. Each multiplication involving $\overline{\boldsymbol{F}}$ implies forward-scattering solutions, where an object is known and the scattered field is found. Therefore these multiplications are expensive, especially when the size of the imaging window and the number of transceivers/receivers are large.

To decrease the computational cost, we employ the multilevel fast multipole algorithm (MLFMA) for the forward solutions [2]. It decreases the computational complexity of the dense matrix multiplications (representing free-space Green's function) from $\mathcal{O}(N^2)$ down to $\mathcal{O}(N)$, where N is the number of pixels (unknowns) in the imaging problem. Even with the reduced complexity, the computational cost of DBIM is humongous because many forward solutions are required for each iteration. As a remedy, the computational burden is distributed among message-passing (MPI) processes, where each process employs an MLFMA solver, each handling the corresponding forward problem(s) [3]. This strategy can parallelize DBIM among at most the number of nodes equal to the number of object illuminations. Further speedup is achieved with parallel MLFMA implementation employing multi-processing (OpenMP) threads inside the MPI processes.

TABLE I
NCSA Blue Waters Node Composition

	Socket 1	Socket 2
CPU Node	AMD Opteron 6276 (8 cores, 32 GB RAM)	AMD Opteron 6276 (8 cores, 32 GB RAM)
GPU Node	AMD Opteron 6276 (8 cores, 32 GB RAM)	NVIDIA Tesla K20x (14 SMX, 6 GB RAM)

The MLFMA multiplications are the main computational cost of DBIM and this paper introduces faster DBIM solutions with a GPU implementation of MLFMA. This proposal uses GPU-enabled computing nodes in a supercomputer and can provide more parallel speedup than a traditional CPU-only implementation does. We use the National Center for Supercomputing Applications (NCSA) Blue Waters supercomputer. There are two types of nodes available, as shown in Table I. The first type (noted as CPU node) has two CPUs and the second type (noted as GPU node) has a CPU and a GPU. The hardware specifications of these nodes are given. When CPU nodes are used, the DBIM solver can efficiently employ 16 threads per node. When GPU nodes are used, the DBIM solver uses a single GPU per node for MLFMA solutions, and eight threads for other computations. The GPU implementation is achieved with CUDA programming. This implementation requires a careful programming with MPI, OpenMP, and CUDA together.

II. GPU Implementation of MLFMA

MLFMA is efficiently implemented on GPUs for solving two-dimensional (2-D) volumetric (inhomogeneous) problems. A similar GPU implementation is previously reported for solving surface problems in [4]. Slightly different from that case, we formulate the MLFMA operations as matrix-matrix multiplications (which are performed by GPUs). The required data structures (matrices) are prepared in CPU and moved to GPU in the setup stage, prior to the iterative solution. We use certain symmetries found in volumetric problems to minimize memory allocations. Separate CUDA kernels are implemented for each MLFMA operation. These kernels are called for evaluation of samples in each level of the MLFMA tree structure. The costly memory fetches are minimized by exploiting caches on the GPU card.

Table II shows execution times (in miliseconds) of each MLFMA stage with sequential and parallel implementations. The parallel executions use either 16 CPU threads (third column), or a single GPU (fourth column). The corresponding

TABLE II
EXECUTION TIME (MS) OF A SINGLE MLFMA MULTIPLICATION

Stage	Sequent.	CPU Node	GPU Node	S/up
Aggregation	5,441.6	379.2 (14.35)	72.9 (74.60)	5.20
Translation	2,413.2	206.0 (11.71)	77.8 (31.01)	2.65
Disaggreg.	5,314.3	371.9 (14.29)	107.5 (49.44)	3.46
Farfield	**13,169.2**	**957.1 (13.76)**	**258.2 (50.99)**	**3.71**
Nearfield	**23,964.1**	**1,597.7 (15.00)**	**479.0 (50.03)**	**3.34**
Total	**37,193.8**	**2,595.5 (14.33)**	**737.7 (50.42)**	**3.52**

speedups over the sequential execution are shown in parentheses. The table shows that GPU can perform MLFMA multiplications approximately 50 times faster than a sequentially programmed CPU. This also demonstrates that a GPU node can perform MLFMA multiplications approximately 3.5 times faster than a CPU node. The demonstrated multiplication involves 10 levels and 16.8 million unknowns. The multiplication domain can include objects as large as 400λ, where λ is the wavelength in free space. Both CPU and GPU executions are highly optimized with careful in-house programming as well as standard compiler optimizations.

III. A LARGE INVERSION RESULT

To demonstrate the proposed parallelization strategy, a large logo of our institution is imaged on 256 computing nodes. The logo consists $1{,}024\times1{,}024$ (approximately one million) pixels. Each side of the imaging window has a length of 102.4λ. A relative permittivity of 1.02 or 1.0 is assigned to the pixels, depending on their positions. The object is illuminated with 256 plane waves from around the object and the scattered field is collected with 1,024 receivers placed in far-zone.

A conjugate-gradient solver is employed for the N-dimensional optimization problem. In each iteration, the negative-gradient direction is found via distorted-Born approximation involving a background object. The iterative solution is terminated after 50 iterations, where each of them requires solution of three forward problems (per illumination). This yields a total of 38,400 forward problems. The forward problems are distributed among the computing nodes where each node solves 150 forward problems, i.e., three problems per iteration. Each problem involves one million unknowns. The nodes are employed with BiCGS solvers for the MLFMA solutions. To prevent transferring data back-and-forth between the CPU and GPU in each BiCGS iteration, the solver is also implemented on GPU. The fields at the receivers are evaluated on-the-fly with MLFMA (on GPU) as well.

Table III shows the execution times of the inverse solution on 256 CPU (first row) and 256 GPU (second row) nodes in seconds. The CPU solution employs a total of 4,096 floating-point cores and the GPU solution employs 2,048 cores and 256 GPUs. The speedup of GPU nodes over CPU nodes is shown in the last row. The construction of the required data structures as well as solution of the known object (which is not available in real-life problems) are included in the setup part (second column). The iterative solution times of the imaging problem are shown in the third column. The last column shows the time spend in the MLFMA multiplications, which constitute the most of the computation. Table III shows that the GPU nodes provide the final image approximately four times faster than the CPU nodes. Fig. 1 shows a few instances from the iterative solution of the inverse (imaging) problem.

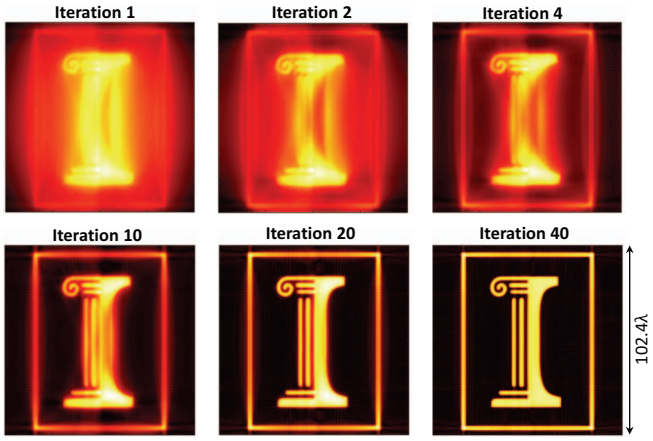

Fig. 1. Imaging of a 2-D logo larger than a hundred wavelengths. The object details can be clearly seen after 40 iterations. A total of 38,400 forward-scattering problems with one million unknowns are solved in 116 seconds on 256 computing nodes with GPUs. The same number of CPU nodes (refer to Table I for hardware details) solves the same problem in about 8.4 minutes.

TABLE III
EXECUTION TIMES (S) OF THE INVERSE SOLUTION

	Setup	Solution	Total	MLFMA
CPU Nodes	5.73	498.98	504.70	440.00
GPU Nodes	3.94	112.38	116.32	93.51
Speedup	1.45	4.44	4.34	4.71

IV. CONCLUSIONS

Supercomputer solutions of large inverse-scattering problems is demonstrated. The immense number of forward-scattering problems are distributed among GPU nodes and solved with MLFMA. For this purpose, MLFMA solver is efficiently implemented on GPUs. As a result, a very large object (with more than 100λ size and one million unknowns) is imaged on 256 GPU nodes in 116 seconds (near-real time). This is approximately four times faster than the same solution on an identical number of CPU nodes.

ACKNOWLEDGMENT

The authors would like to acknowledge allocation of computer time from NCSA Blue Waters (through NSF grants OCI-0725070 and ACI-1238993), NVIDIA GPU Center of Excellence grant, and NSF grant EECS-1609195.

REFERENCES

[1] W. C. Chew and Y.-M. Wang, "Reconstruction of two-dimensional permittivity distribution using the distorted Born iterative method," *IEEE Trans. Med. Imag.*, vol. 9, no. 2, pp. 218–225, 1990.

[2] A. J. Hesford and W. C. Chew, "Fast inverse scattering solutions using the distorted Born iterative method and the multilevel fast multipole algorithm," *J. Acoust. Soc. Am.*, vol. 128, pp. 679–690, Aug. 2010.

[3] M. Hidayetoğlu, C. Yang, L. Wang, A. Podkowa, M. L. Oelze, W.-M. Hwu, and W. C. Chew, "Parallel solutions of inverse multiple scattering problems with Born-type fast solvers," *PIERS Proc.*, Aug. 2016.

[4] J. Guan, S. Yan, and J.-M. Jin, "An OpenMP-CUDA implementation of multilevel fast multipole algorithm for electromagnetic simulation on multi-GPU computing systems," *IEEE Trans. Antennas Propag.*, vol. 61, no. 7, pp. 3607–3616, 2013.