An MPIxOpenMP Implementation of the Hierarchical Parallelization of MLFMA

Mert Hidayetoğlu and Levent Gürel

ABAKUS Computing Technologies, Cyberpark, Bilkent, Ankara, TR-06800, Turkey
lgurel@gmail.com

***Abstract*—We propose an MPIxOpenMP parallelization scheme based on the hierarchical partitioning strategy for the multilevel fast multipole algorithm. The parallelization scheme reduces data duplications via sharing data structures among processing cores. Therefore, this scheme can employ hundreds of cores efficiently without requiring extra memory.**

I. Introduction

The multilevel fast multipole algorithm (MLFMA) provides accurate solutions of electromagnetic radiation and scattering problems with $\mathcal{O}(N Log\, N)$ complexity [1], where N is the number of unknowns in a problem. For solving large problems, MLFMA is parallelized on distributed-memory computer architectures. An efficient implementation of the parallel MLFMA uses the hierarchical partitioning strategy with the message passing interface (MPI) [2]. The hiearchical partitioning provides excellent efficiency by distributing the MLFMA data structures in a load-balanced manner and by minimizing communications among processes. However, it is inevitable to duplicate some frequently-used data structures, e.g., the MLFMA tree structure, since passing them from one processor to another via communications is costly. Even if the duplicated data structures are relatively small, the memory required for storing multiple copies of them accumulates rapidly with the number of processes and ruins the efficiency of memory parallelization for large numbers of processes. As a result, when solving large problems within a limited amount of available memory, the inefficiency prevents employing more processing cores even when they are available.

In order to avoid excessive duplications of data structures, for each MPI process, we employ t open multi-processing (OpenMP) threads, which can share the data of their MPI parent. The OpenMP threads employ idle processing cores and paralelize the task of their parent without requiring extra memory. In other words, the MPIxOpenMP parallelization employs $p \times t$ processing cores using the same amount of memory as that of pure-MPI parallelization with p processes.

We categorize the parallel regions of MLFMA into two; MPI regions use the conventional hierarchical parallelization with p MPI processes and MPIxOpenMP regions use $p \times t$ OpenMP threads within the parallelization scheme. MPIxOpenMP parallelization is rather simple when no communication is required. Essentially, sibling threads execute different sections of large loops of their MPI parent in parallel. However, special care must be taken in order to implement the communications among threads.

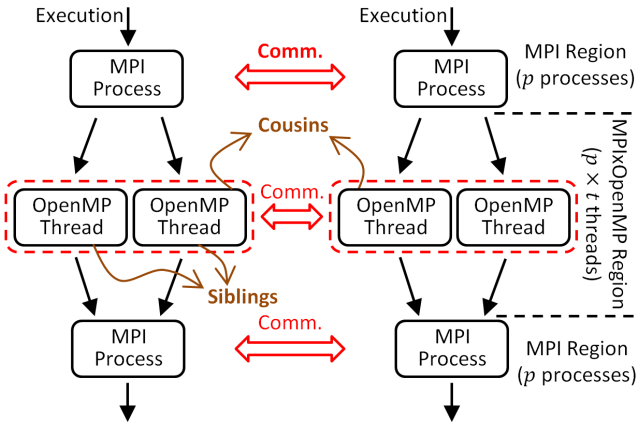

Fig. 1. Each MPI process forks off a number of OpenMP threads in MPIxOpenMP regions. As an example, two MPI processes in the MPI region are shown, and each of them summons two OpenMP threads in the MPIxOpenMP region. The OpenMP threads are employed on idle processing cores by the operating system.

II. MPIxOpenMP Parallelization

When the code execution reaches an MPIxOpenMP region, each MPI process forks off a number of OpenMP threads, as depicted in Fig. 1, and each thread employs an available idle processing core. The children threads of a process share the computational task of their parent in a load-balanced manner.

An OpenMP thread can share its memory with its sibling threads, however, it cannot address the memory of its cousin threads because the cousins belong to a distinct MPI parent. When two cousin threads need each other's data, thread-safe communications [3] must be performed among the threads, where each thread uses its parent's MPI communicator. For ensuring thread safety, no sibling threads communicate at the same time. An OpenMP thread prevents their siblings to intervene its communication by initiating a blocking communication in a critical section.

The critical sections may lead to a deadlock when two cousin threads, which will communicate, cannot meet in critical sections at the same time. To solve the issue, a thread, expecting a communication, continuously probes whether a communication request is received from its cousin threads. By doing so, it does not block its sibling threads to communicate. When a communication request is received, a thread enters a critical section and initiates the blocking communication, while its siblings wait idle for the communication to be completed.

978-1-4673-7197-1/15/$31.00 ©2015 IEEE

III. Numerical Results

In order to demonstrate the efficiency of the proposed MPIxOpenMP parallelization, we report the solution of a scattering problem involving a conducting sphere with 240λ diameter, where λ is the wavelength of the illuminating plane wave. The same problem is solved with MPIxOpenMP and pure-MPI parallelizations and with various numbers of processing cores on a 16-node parallel computer cluster. The cluster has 2 TB memory and 512 processing cores. The problem involves approximately 53 million unknowns with 0.1λ meshing and is solved with all possible $p \times t$ combinations. The amount of required memory for each solution is recorded. The solutions use out-of-core method, which stores large data structures on disk [4], [5], saving approximately 96 GB of memory.

TABLE I
CPU Times and Required Memories with MPI

Num. Process	16	32	64	128	256	512
CPU Time (s)	22633	12090	6645	4117	3050	3199
Memory (GB)	72	78	90	112	166	264

Table I shows the CPU times and required memories for pure-MPI solutions. Each column represents an MLFMA solution and the first row shows the number of MPI processes. CPU times are decreasing with the number of processes, as expected, except for the solution with 512 processes because of the communication overhead. Note that the required memory increases immensely with the number of processes and the solution with 512 process requires 3.7 times memory that of the solution with 16 processes. This inefficiency would prevent the solution of the problem with 128 (and more) processes if the cluster had 100 GB of memory.

TABLE II
CPU Times and Required Memories with MPIxOpenMP

Num. Threads	64x1	64x2	64x4	64x8
CPU Time (s)	6645	4100	2777	2126
Memory (GB)	90	91	93	97

Table II shows the CPU times and required memories for MPIxOpenMP solutions with 64 MPI parents. CPU times decrease with the number of threads, as expected. Note that the MPIxOpenMP parallelization is significantly effective with 512 cores, requiring less time than the pure-MPI parallelization (though the time efficiency of the proposed parallelization is not the main theme of this paper). The required memory increases slightly with OpenMP threads because of threads' communication buffers. Neverheless, MPIxOpenMP uses 512 cores with 2.7 times less memory than that of pure MPI.

Figure 2 shows the memory-parallelization efficiency of all possible $p \times t$ combinations when a pure-MPI solution with 16 processes (one process for each node) is regarded as the reference. The memory efficiency E_c^M with $c = p \times t$ processing cores is defined as $E_c^M = M_{16}/M_c$, where M_c is the required memory for a solution with c processing cores. Figure 2 shows that the efficiency of pure MPI drops significantly with p, while MPIxOpenMP uses $p \times t$ cores and its efficiency decreases slightly with t. For example, MPIxOpenMP uses 512 processing cores with 86 GB while the pure MPI requires 264 GB.

In addition to the sphere with 240λ diameter, an extremely large sphere with 1000λ diameter is solved. The large problem involves approximately 1.1 billion unknowns and solved on 64 cores (cannot be solved on more processes due to memory limitations) with pure MPI and 512 cores with MPIxOpenMP. The MPIxOpenMP solves the problem in 23.5 hours with 64×8 threads while the pure MPI solves the problem in 71.3 hours with 64 processes. Other relevant computational results are presented in [5].

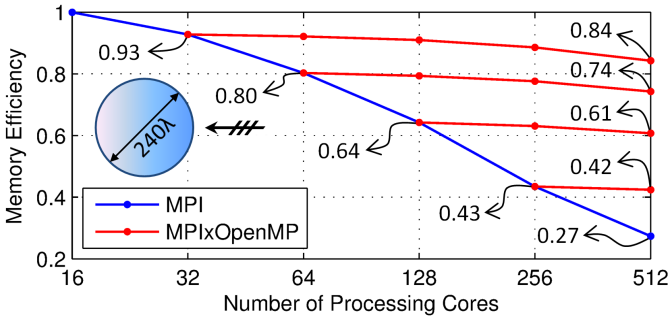

Fig. 2. Memory efficiencies for solutions of 53 million unknowns with various $p \times t$ combinations. The red curves represent MPIxOpenMP solutions and the blue curve represents pure-MPI solutions. MPIxOpenMP provides better efficiency with large numbers of processing cores than pure MPI since memory duplication among OpenMP threads is limited to their communication buffers.

IV. Conclusions

The MLFMA solver is parallelized with the MPIxOpenMP scheme. This implementation is based on the hierarchical partitioning strategy and does not require extra memory for employing more processing cores, and therefore provides better memory efficiency than the pure-MPI parallelization. To demonstrate the benefits of the MPIxOpenMP parallelization, a scattering problem involving more than 1.1 billion unknowns is solved with 512 processing cores in 23.5 hours within 2 TB memory, while it can be solved with at most 64 cores in 71.3 hours using the pure-MPI parallelization.

Acknowledgment

This work was supported by Schlumberger-Doll Research (SDR). The authors would like to thank Intel Corporation and Jamie Wilcox for a generous allocation of parallel-computer time.

References

[1] J. Song, C.-C. Lu, and W. C. Chew, "Multilevel fast multipole algorithm for electromagnetic scattering by large objects," *IEEE Trans. Antennas Propag.*, vol. 45, no. 10, pp. 1488–1493, Oct. 1997.

[2] Ö. Ergül and L. Gürel, "A hierarchical partitioning strategy for an efficient parallelization of the multilevel fast multipole algorithm," *IEEE Trans. Antennas Propag.*, vol. 57, no. 6, pp. 1740–1750, June 2009.

[3] W. Gropp and R. Thakur, "Issues in developing a thread-safe MPI implementation," *The 13th European PVM/MPI User's Group Meeting*, Bonn, Germany, Sep. 2006.

[4] M. Hidayetoğlu and L. Gürel, "MLFMA memory reduction techniques for solving large-scale problems," *2014 IEEE Int. Symp. Antennas Propagation USNC-URSI Nat. Radio Science Meeting*, Memhpis, TN, USA, July 2014.

[5] M. Hidayetoğlu and L. Gürel, "Parallel out-of-core MLFMA on distributed-memory computer architectures," *CEM'15 Computational Electromagnetics Workshop*, Izmir, Turkey, July 2015.