Porting an Explicit Time-Domain Volume Integral Equation Solver onto Multiple GPUs Using MPI and OpenACC

Saber Feki [1], Ahmed Al-Jarro [3], and Hakan Bagci [2]

[1] KAUST Supercomputing Laboratory
[2] Division of Computer, Electrical and Mathematical Sciences and Engineering
King Abdullah University of Science and Technology (KAUST), Thuwal, 23955-6900, KSA
{saber.feki, hakan.bagci}@kaust.edu.sa

[3] Department of Electronic and Electrical Engineering
University College London, Torrington Place, WC1E 7JE, London, UK
ahmed.aljarro@ucl.ac.uk

Abstract ― A scalable parallelization algorithm to port an explicit marching-on-in-time (MOT)-based time domain volume integral equation (TDVIE) solver onto multi-GPUs is described. The algorithm makes use of MPI and OpenACC for efficient implementation. The MPI processes are responsible for synchronizing and communicating the distributed compute kernels of the MOT-TDVIE solver between the GPUs, where one MPI task is assigned to one GPU. The compiler directives of the OpenACC are responsible for the data transfer and kernels' offloading from the CPU to the GPU and their execution on the GPU. The speedups achieved against the MPI/OpenMP code execution on multiple CPUs and parallel efficiencies are presented.

Index Terms ― Explicit marching-on-in-time scheme, GPU, MPI, OpenACC, time-domain volume integral equation.

I. INTRODUCTION

The use of hardware accelerators, including multi and many-core architectures, has been increasing in many emerging applications of high performance computing (HPC) as they provide cost effectiveness, power efficiency, and physical density. Nevertheless, one of the limiting factors to a wider spread use of multi-core accelerators, such as GPUs, is the human-labor intensive porting process required by low-level programming models, such as CUDA [1] and OpenCL [2]. To overcome this limit, HPC research has focused on developing high-level directive based programming models, such as OpenACC [3], which provide compiler directives and clauses to annotate codes originally developed for CPUs in a manner similar to how OpenMP [4] is used on codes executed on multicore CPU architectures. This high-level approach, when carefully implemented, significantly reduces the re-programming efforts while maintaining the efficiency of the resulting codes.

In this work, we report on our recent efforts on parallelizing a fully explicit marching-on-in-time (MOT)-based time-domain volume integral equation (TDVIE) solver [5] for efficient execution on multiple GPUs. The MOT-TDVIE solvers are becoming attractive alternatives to finite difference time domain (FDTD) schemes for analyzing transient electromagnetic scattering from inhomogeneous dielectric objects [5, 6]. However, their effective use in practical problems of photonics, optoelectronics, and bio-electromagnetics, where electrically large scatterers need to be discretized with millions of degrees of freedom, relies on acceleration algorithms such as the plane-wave time domain (PWTD) method [7] and/or hardware-based acceleration [8-11].

Our recent research has focused on the latter; we developed highly scalable parallelization algorithms [8, 9] to enable the explicit MOT-TDVIE solver of [5] in analyzing scattering from electrically large structures. Additionally, we used OpenACC to enable the execution of the same solver on GPUs [10, 11]. Significant performance improvements with up to 30X and 11X speedups relative to the sequential and multi-threaded CPU codes were achieved. Furthermore, we demonstrated that the (single) GPU-accelerated MOT-TDVIE solver could leverage energy consumption gains on the order of 3X relative to its multi-threaded CPU version [10]. In this paper, we describe in detail the process of porting the same MOT-TDVIE solver onto *multi*-GPUs using MPI/OpenACC. Additionally, we present numerical results, which demonstrate that the ported code executes up to 11.2X faster on multi-GPUs than on conventional CPUs.

Submitted On: September 20, 2015
Accepted On: July 4, 2016

II. MOT-TDVIE SOLVER

A. MOT-TDVIE algorithm

Let V represent the volumetric support of dielectric scatterer with permittivity $\varepsilon(\mathbf{r})$ residing in an unbounded background medium with permittivity ε_0. The scatterer is excited by a band-limited incident electric field $\mathbf{E}_0(\mathbf{r},t)$. Upon excitation, scattered field $\mathbf{E}^{\text{sca}}(\mathbf{r},t)$ is generated. Scattered and incident fields satisfy $\mathbf{E}(\mathbf{r},t)=\mathbf{E}_0(\mathbf{r},t)+\mathbf{E}^{\text{sca}}(\mathbf{r},t)$, where $\mathbf{E}(\mathbf{r},t)$ is the unknown "total" field. One can construct a TDVIE as [5, 6]:

$$\mathbf{E}(\mathbf{r},t)=\mathbf{E}_0(\mathbf{r},t)+\left[\nabla\nabla\cdot-\partial_t^2/c_0\right]\mathbf{A}(\mathbf{r},t),\ \mathbf{r}\in V, \quad (1)$$

where $\mathbf{A}(\mathbf{r},t)$ is given by:

$$\mathbf{A}(\mathbf{r},t)=\int_V\frac{(\varepsilon(\mathbf{r}')-\varepsilon_0)\mathbf{E}(\mathbf{r}',t-R/c_0)}{4\pi\varepsilon_0 R}dv',\ \mathbf{r}'\in V. \quad (2)$$

Here, c_0 is the speed of light in the background medium, and $R=|\mathbf{r}-\mathbf{r}'|$ is the distance between points $\mathbf{r}$ and $\mathbf{r}'$. TDVIE (1) is solved by time marching, which makes use of an explicit predictor-corrector algorithm as described next [5]. First V is discretized using N_e cubic elements. Let $\mathbf{r}_k$, $k=1:N_e$, and Δt represent the centers of these elements and time step size. Assume n represents the index of the "current" time step. At the predictor step, first $\mathbf{A}_{k,n}=\mathbf{A}(\mathbf{r}_k,n\Delta t)$ are computed using $\mathbf{E}_{l,m}=\mathbf{E}(\mathbf{r}_l,m\Delta t)$, $l=1:N_e$, $m=\max(1,n-N_g):n\Delta t$, in the integral given in (2). For this operation, $\mathbf{E}(\mathbf{r},m\Delta t)=\mathbf{E}_{l,m}$ is assumed within cubic element l and linear interpolation is used to approximate $\mathbf{E}(\mathbf{r}_l,n\Delta t-R_{kl}/c_0)$, where $R_{kl}=|\mathbf{r}_k-\mathbf{r}_l|$, from $\mathbf{E}_{l,m-1}$ and $\mathbf{E}_{l,m}$ for $[m-1]\Delta t<n\Delta t-R_{kl}/c_0<m\Delta t$. Note that here $N_g=\lfloor R_{\max}/c_0\Delta t\rfloor+2$, where $R_{\max}=\max\{R_{kl}\}$, for any $\mathbf{r}_k,\mathbf{r}_l\in V$. Then, finite differences (FD), which approximate the spatial derivative operator "$\nabla\nabla\cdot$", are applied to $\mathbf{A}_{k,n}$ to yield "predicted" samples $\mathbf{E}_{k,n}$. Differentiation "∂_t^2" in (1) is approximated using backward FD for pairs $(\mathbf{r}_k,\mathbf{r}_l)$ that satisfy $R_{kl}<2c_0\Delta t$ and using central FD for all other pairs. At the corrector step, differentiation "∂_t^2" is *recomputed* using a central difference formula for pairs $(\mathbf{r}_k,\mathbf{r}_l)$ that only satisfy $R_{kl}<2c_0\Delta t$. Note that use of central FD is now allowed since field samples that are not known at the predictor step (due to causality) can now be replaced by the predicted fields' samples. At the end of time step n, $\mathbf{E}_{k,m}$ are stored as part of the "history" of field samples to be used in the computation of $\mathbf{A}_{k,n+1}$.

Note that FD evaluations and corrector updates are spatially local operations while computation of $\mathbf{A}_{k,n}$, $k=1:N_e$, is global. Samples $\mathbf{E}_{l,m}$ that satisfy the condition $[n-m]c_0\Delta t>R_{kl}$ do not contribute to $\mathbf{A}_{k,n}$ since the fields radiated from point $\mathbf{r}_l$ at time $m\Delta t$ have not yet reached point $\mathbf{r}_k$ at time $n\Delta t$. This also means that for $n\geq N_g$, all fields radiated from all points reach to all other points. Consequently, they all contribute to all samples $\mathbf{A}_{k,n}$, $k=1:N_e$, rendering the computational cost of the integral evaluation $O(N_e^2)$ per time step for all $n\geq N_g$. As N_e increases, the cost of computing $\mathbf{A}_{k,n}$ limits the solver's applicability to electrically large problems. This limitation can be overcome by using acceleration algorithms such as the PWTD method [6-7] and/or highly scalable parallelization algorithms [8-11]. In this work, we implement and fine-tune the parallelization algorithm of [8, 9], which is originally developed for CPUs, for multi-GPUs to further increase the applicability of the MOT-TDVIE solver to electrically large problems.

B. MPI parallelization

Operations required by the MOT-TDVIE solver at each time step can be grouped into two: (i) computation of $\mathbf{A}_{k,l}$, $k=1:N_e$, which requires access to samples $\mathbf{E}_{l,n-m}$, $l=1:N_e$, $m=1:\min(n-1,N_g)$ and (ii) computation of samples $\mathbf{E}_{k,n}$ by applying FD to $\mathbf{A}_{k,n}$. The parallelization scheme used here, first, ensures the even distribution of the memory via application of the graph-based partitioning scheme to the distribution of the points $\mathbf{r}_k$, $k=1:N_e$, representing the discretization of V. This results in an unstructured partitioning of the points $\mathbf{r}_k$ [9]. In this partitioning, each process stores only $\mathbf{E}_{k,n}$ and $\mathbf{E}_{l,n-m}$ that belong to the partition assigned to it. The computational load of step (i) is distributed using a one-way pipeline communication strategy, so-called the "rotating tiles" paradigm [8]. The test tiles (partitions that contain test points) are initially same as the source titles (partitions that contain source points) at the beginning of the rotation but they are rotated among the processors during the computation of $\mathbf{A}_{k,l}$. When a processor receives a test tile, it first adds the contribution from the source titles it stores to $\mathbf{A}_{k,l}$ associated with the received test tile, then it passes the (updated) tile to its "neighboring" processor. At the end of a full rotation all contributions to $\mathbf{A}_{k,l}$ are computed. It is noted here that, this strategy eliminates the need for globally executed collective routines such as MPI_Reduce [8]. The computational load of step (ii) is distributed using the same grouping of the test points provided by the graph-partitioning algorithm. This reduces the communication costs associated with spatial FD computations by ensuring that the data communication only happens between points residing on the boundary of any two partitions [9].

III. PORTING TO MULTIPLE GPUS

The state-of-the-art GPU-nodes can include up to 8 K80 GPUs, which is essentially equivalent to having 16 independent GPUs. On the other hand, OpenACC standard, as a stand-alone programming model, provides very limited support for code development on multiple devices. Therefore, one typically relies on using OpenACC/OpenMP together with the MPI standard to port codes onto a cluster of nodes equipped with multiple GPUs/multicore CPUs. In this work, OpenACC is used to accelerate the time marching loop of the MOT-TDVIE

solver. Both memory- and compute-bound operations are executed on GPUs, which benefit from the improved memory bandwidth and higher flop rate, respectively. However, because the amount of compute-bound operations is significantly higher that memory-bound operations, benefits from increased memory bandwidth might be considered negligible. The main advantages of OpenACC over CUDA are the significantly increased programming efficiency and code portability on different hardware platforms. More specifically, OpenACC offers an easy way to port codes onto accelerators using simple descriptive compiler directives. Additionally, the same OpenACC-annotated code can be compiled on different hardware platforms, including the host itself (multicore CPU architecture) as well as any other accelerator supported by the OpenACC standard. In contrast, the CUDA programming model is more tedious to implement and can be used on only NVIDIA GPUs.

The code is designed such that the number of MPI processes spawn on each node is equal to the number of GPUs per node. Each MPI process is assigned to a GPU using the runtime API function *acc_set_device_num* to set the GPU target to the MPI rank modulo the number of GPUs per node, as shown in the pseudo code in Fig. 1. The data directive *#pragma acc data* is applied to the outermost time loop in order to minimize data transfers between the host and the device. Input and output arrays are annotated with clauses *present_or_copyin* and *present_or_copyout*, respectively. However, the arrays needed for the MPI communications, which are of very limited memory size, are copied in and out at each iteration so that they are accessible to the MPI routines. Each enclosed code block in the MOT-TDVIE solver is annotated with *#pragma acc kernels* and offloaded to the assigned GPU. The code blocks implementing the computation of $\mathbf{A}_{k,l}$ consist of two nested loops yielding a quadratic computational complexity. The second loop is further annotated with *#pragma acc loop reduction* and the associated variables to further optimize the sum operation of all source contributions. The OpenACC standard offers the ability to further tune loop execution using the *gang* and *vector* clauses, which can be used to modify the number of blocks of threads and threads per block to be executed, respectively. Since there are only two nested loops in the kernels of the parallel MOT-TDVIE solver, values assigned to these two parameters by the compiler already result in good performance improvements. Having said that, tuning these parameters in the presence of three or more nested loops may significantly increase the performance. Indeed, this was demonstrated for the serial version of the code, with structured grid, when executed on single GPUs. The tuning of these two parameters improved the acceleration performance by up to 23X [10]. For some of the loops that are not parallelized by the compiler due to perceived false data dependencies, the code block is annotated with the loop pragma accompanied with the independent clause to avoid unnecessary synchronization between the loop iterations in absence of data dependencies. Note that, the code design using multi-GPU kernels allows for MPI synchronizations and communications to take place between the compute kernels as necessary. That is the case with the rotating tiles communications implemented to compute $\mathbf{A}_{k,l}$, and the halo cells exchange communications implemented to compute $\mathbf{E}_{k,n}$ using FD.

```
// Get number of MPI processes = # of GPUs
MPI_Init(&argc, &argv);
MPI_Comm_size(MPI_COMM_WORLD, &size);
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
// Assign each MPI process to a GPU
acc_set_device_num(rank%ngpus,acc_device_nvidia);

#pragma acc data present_or_copyin(input
arrays) present_or_copyout(output arrays)

for (int t=0;  t<nt; ++t) {  // time loop
  for(rot=0; rot<=size; ++rot){
    // MPI communication for rotating tiles
    MPI_Sendrecv();
    MPI_Barrier();
    #pragma acc kernels
    // Spatio-temporal convolutions
    for (int k=0; k<Ne; ++k){
      #pragma acc loop reduction
      for (int l=0; l<Ne; ++l){
          A[t][k] = A[t-tkl][l] + ...
  }  }
  MPI_Barrier();
    // Loops with no data dependencies
    #pragma acc kernels
    #pragma acc loop independent
    for(i=0; i<ni; ++i){

    }
  } // end rotation

  // MPI communication for Halo Exchange
  MPI_Sendrecv();
  MPI_Barrier();
  #pragma acc kernels
  // spatial finite difference operations
  for (int k=0; k<Ne; ++k){
    B[t][k] = A[t][k] + ....
  }
} // end time loop
```

Fig. 1. Pseudo code for the implementation of the MOT-TDVIE solver using MPI and OpenACC.

IV. NUMERICAL EXPERIMENTS

The test bed used for performance evaluation consists of a system of two nodes connected using an Infiniband FDR high-speed network. Each node is a dual socket CPU system hosting four NVIDIA Kepler K20c GPUs. Each socket is an eight-core Sandy Bridge

Intel(R) Xeon(R) CPU E5-2650.

In our performance evaluation, as shown in Fig. 2, a significant speedup ranging from 7.4X to 11.2X is recorded comparing the MPI and OpenMP implementation on 16 cores SandyBridge to the MPI and OpenACC implementation on four K20c GPUs. It is also observed that as N_e increases, higher speedup is achieved. This is due to the fact that the GPUs are supplied with larger computational loads; therefore, taking better advantage of its computational capacity. It has been shown before that the MPI implementation demonstrated a great scalability on large super-computers [8-9]. Figure 3 shows the parallel efficiency of the MPI and OpenACC implementation executed on two and eight GPUs, which ranges from 82% to 94%. Another advantage of using NVIDIA GPUs is their energy efficiency as the simulation consumed 2.4X less energy on GPUs than on CPUs. For all of the above, the GPUs are identified as the preferred computing platform in our overall performance analyses of the explicit MOT-TDVIE solver.

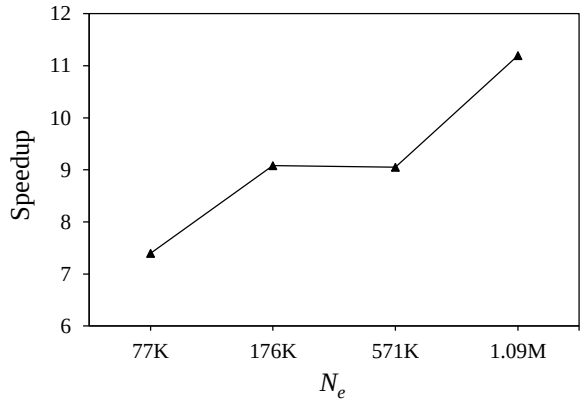

Fig. 2. Performance speedup of MPI and OpenACC on four K20c GPUs compared to MPI and OpenMP on 16 cores SandyBridge CPU.

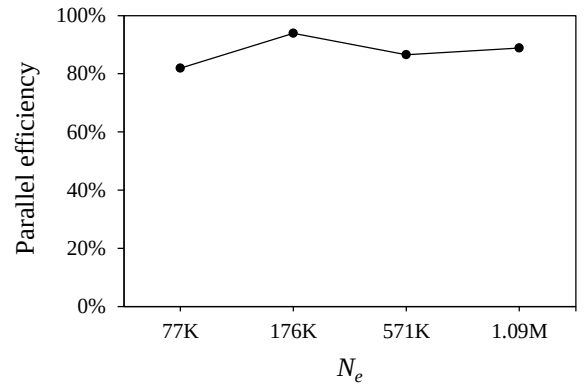

Fig. 3. Parallel efficiency of the MPI and OpenACC implementation scaling from two to eight GPUs.

V. CONCLUSION

The porting of the explicit MOT-TDVIE solver using MPI and OpenACC to multi-GPUs resulted in a highly efficient implementation. The simulations executed on multi-GPUs were faster by up to an order of magnitude compared to those executed on CPUs (using the MPI and OpenMP version of the code). The OpenACC API has the advantage of easily porting the MPI code to multi-GPU environment; therefore, increases the developer productivity while keeping the legacy of the original CPU code. Furthermore, the parallelization allows the explicit TDVIE solver to efficiently simulate transient electromagnetic wave interactions on electrically large structures discretized using a large number of spatial elements on GPUs.

REFERENCES

[1] CUDA, www.nvidia.com, 2016.

[2] OpenCL, www.khronos.org/opencl, 2016.

[3] OpenACC, www.openacc-standard.org, 2016.

[4] OpenMP, www.openmp.org, 2016.

[5] A. Al-Jarro, M. A. Salem, H. Bagci, T. M. Benson, P. Sewell, and A. Vukovic, "Explicit solution of the time domain volume integral equation using a predictor-corrector scheme," *IEEE Trans. Antennas Propag.*, vol. 60, no. 11, pp. 5203-5214, 2012.

[6] N. T. Gres, A. A. Ergin, E. Michielssen, and B. Shanker, "Volume-integral-equation-based analysis of transient electromagnetic scattering from three-dimensional inhomogeneous dielectric objects," *Radio Sci.*, vol. 36, no. 3, pp. 379-386, May 2001.

[7] Y. Liu, A. Al-Jarro, H. Bagcı, and E. Michielssen, "Parallel PWTD-accelerated explicit solution of the time domain electric field volume integral equation," *IEEE Trans. Antennas Propag.*, vol. 64, no. 6, pp. 2378-2388, 2016.

[8] A. Al-Jarro, M. Cheeseman, and H. Bagci, "A distributed-memory parallelization of the explicit time-domain volume integral equation solver using a rotating tiles paradigm," in Proc. *28th Int. Review of Progress in Appl. Comp. Electromagn.*, 2012.

[9] A. Al-Jarro and H. Bagci, "An unstructured mesh partitioning scheme for efficiently parallelizing an explicit time domain volume integral equation solver," in Proc. *29th Int. Review of Progress in Appl. Comp. Electromagn.*, 2013.

[10] S. Feki, A. Al-Jarro, A. Clo, and H. Bagci, "Porting an explicit time-domain volume-integral-equation solver on GPUs with OpenACC," *IEEE Antennas Propag. Mag.*, vol. 56, pp. 265-277, 2014.

[11] S. Feki, A. Al-Jarro, and H. Bagci, "Multi-GPU-based acceleration of the explicit time domain volume integral equation solver using MPI-OpenACC," in Proc. *IEEE Int. Symp. Antennas Propag. and USNC/URSI National Radio Sci.*, Meet., 2013.