RE-INTEGRATE EMT Simulation Software: Distributed Parallelism for BBD Solver Using Multiple GPUs and Streams

Phuong H. Hoang
Oak Ridge National Laboratory
Knoxville, USA
hoangh@ornl.gov

Rahul Mishra
Oak Ridge National Laboratory
Knoxville, USA
mishrar1@ornl.gov

Harry Hughes
Oak Ridge National Laboratory
Knoxville, USA
hugheshn@ornl.gov

Suman Debnath
Oak Ridge National Laboratory
Knoxville, USA
debnaths@ornl.gov

Sri Kainkaryam
NVIDIA Corporation
Houston, USA
skainkaryam@nvidia.com

Pavel Dimitrov
NVIDIA Corporation
Houston, USA
pdimitrov@nvidia.com

Ahsan Yousufzai
NVIDIA Corporation
Houston, USA
ayousufzai@nvidia.com

***Abstract*—Electromagnetic transient (EMT) simulations are known for their superior accuracy in capturing fast transients associated with power electronics or other fast-acting devices in the power grid compared to phasor-domain transient stability or power flow simulations. However, the time taken to simulate in EMT presents significant challenges to widespread adoption and to perform large-scale simulations. Distributed parallelism, particularly leveraging graphic processing units (GPUs), offers one of the practical solutions to this bottleneck by significantly accelerating computations. This paper presents a distributed parallelism scheme for solving bordered-block diagonal (BBD) matrix equations, which commonly arise in EMT simulations. The proposed method was tested on matrices generated from a power plant with collector system and inverters, demonstrating its capability to reduce computation time significantly. This advancement effectively addresses critical computational and scalability challenges, paving the way for more efficient EMT simulation frameworks in modern power system studies.**

***Index Terms*—EMT, GPU, Distributed Parallelism, BBD Solver.**

I. Introduction

The increasing integration of power electronics equipment into power grids—such as generation plants with inverters, high-voltage direct current (HVdc) systems, loads with inverters (e.g., data centers), and solid-state transformers—makes it challenging for existing phasor-domain transient stability or power-flow-based tools to accurately capture fast transient events associated with these components, thereby reducing the reliability of power grid models. This necessitates the use of high-fidelity electromagnetic transient (EMT) models to effectively capture these fast transients and enhance the reliability of power system through improved planning and integration studies. As a matter of fact, regulatory entities have initiated guidelines and requirements for the use of EMT models in certain power system studies [1], [2]. However, one of the major challenges with EMT simulations is the time they take to complete, which makes large-scale simulations challenging and multi-scenario studies very time consuming.

Optimizing distributed parallelism in computing and memory allocation for EMT simulations presents one of the promising solutions to speed-up simulations. Advances in parallel computing and processing hardware, such as graphic processing units (GPUs), have achieved significant progress and have been successfully applied across various industries and applications, specifically in training of artificial intelligence (AI) models. There is scope to utilize GPUs for parallel computations in EMT. In the EMT simulations, a system of linear equations is formed after discretization. AI-enabled matrix form detection methods, which can automatically and effectively identify the matrix structure like bordered-block diagonal (BBD) form, among others, and reorder it to maximize parallelism, provide new opportunities to enhance parallelism in EMT simulations [3]. There are practical cases where a system's matrix can be reordered into the doubly bordered-block diagonal (BBD) form, which supports parallelism in EMT simulations. For example, this is applicable to generation plant's collector systems with inverters that comply with IEEE Standard 2800 [4], [5] and/or data center loads with distribution system and inverters.

Parallelism in EMT simulations has recently been emphasized in NERC's guidelines [6] as a means to enhance the performance of these simulations. Effective software tools are needed to automatically identify opportunities for parallelism and implement them, reducing reliance on user input. This is important because user input can be prone to errors and/or require a high level of expertise. Existing EMT tools are implemented primarily for central processing units (CPUs). To the best of the authors' knowledge, the implementation for graphics processing units (GPUs) for EMT simulations are still in the research phase, and their optimality remains a topic of

Research sponsored by Advanced Grid Modeling (AGM) Office of U.S. Department of Energy. The views expressed herein do not necessarily represent the views of the U.S. Department of Energy or the United States Government. This manuscript has been authored in part by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy (DOE). The US government retains and the publisher, by accepting the article for publication, acknowledges that the US government retains a nonexclusive, paid-up, irrevocable, worldwide license to publish or reproduce the published form of this manuscript, or allow others to do so, for US government purposes. DOE will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan (http://energy.gov/downloads/doe-public-access-plan).

979-8-3315-8138-1

discussion [7], [8], [9]. In [7], the authors explore and evaluate existing GPU-based solvers to accelerate EMT simulations without investigating specific matrix structures and distributed schemes. In [8], a GPU-based implementation is explored, where components of a multiterminal direct current (MTdc) grid are individually modeled and deployed into GPUs. In [9], a thread-oriented strategy is introduced, which discriminates between electrical and control systems. Implementing each individual component or system separately for GPU deployment imposes limitations on the scalability and automation of the approach.

The main contributions of the paper are as follows. It introduces a distributed parallelism framework for the linear solver in EMT simulations, which solves systems of linear equations characterized by the BBD matrix structure. The proposed scheme leverages the computational power of GPUs to achieve high-performance parallel computation. The solver is inherently component-agnostic and is integrated into a unified EMT simulation platform, making it applicable to solving general systems of linear equations, regardless of the specific physical components involved. This paper also provides insight into the implementation of the BBD solver across multiple GPUs and their streams.

The structure of this paper is organized as follows. Section II provides a brief introduction to the practical context of the BBD solver, along with the mathematical derivations of matrix operations. Section III gives an overview of the GPU architecture as background information, followed by the presentation of a distributed parallelism scheme for the BBD solver. Section IV discusses the simulation results, and Section V concludes the paper with conclusions and future work.

II. Collector System Modeling

It is important to note that the proposed distributed parallelism scheme can be applied to any system whose discretized systems of linear equations exhibit a BBD matrix structure. In this paper, however, we present a practical application of the scheme to demonstrate its utility. Specifically, the practical case considered involves generation plant's collector systems with inverters designed to meet the IEEE 2800 standard [5], as illustrated in Fig. 1.

In [4], the authors investigate the characteristics and properties of collector systems [5]. After the discretization of the system's differential-algebraic equations (DAEs), the resulting system of linear equations takes the general form $\boldsymbol{A}\mathbf{x}=\boldsymbol{b}$. Here, $\boldsymbol{A}$ represents the system matrix, which has a nested doubly-bordered block diagonal structure. Meanwhile, $\mathbf{x}$ represents the vector of state variables, and $\mathbf{b}$ represents the right-hand side vector originating from the discretized system inputs and derivatives.

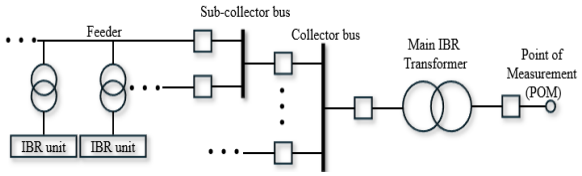

Fig. 1: Illustration of generation plant's collector systems

With N collector buses, the following system of linear equations can be obtained:

$$\begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} & \dots & \boldsymbol{A}_{1N} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} & 0 & 0 \\ \dots & 0 & \ddots & 0 \\ \boldsymbol{A}_{N1} & 0 & 0 & \boldsymbol{A}_{NN} \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_N \end{bmatrix} = \begin{bmatrix} \boldsymbol{b}_1 \\ \mathbf{b}_2 \\ \vdots \\ \mathbf{b}_N \end{bmatrix} \quad (1)$$

It is pointed in [4] that once $\mathbf{x}_1$ is obtained, the remaining $\mathbf{x}_i, i \in \{2, \dots, N\}$, can be solved independently, thereby creating opportunities for parallelism. The solutions to Equation (1) are expressed as follows.

$$\mathbf{x}_1 = (\boldsymbol{A}_{11} - \sum_{i=2}^{N} \boldsymbol{A}_{1i}\boldsymbol{A}_{ii}^{-1}\boldsymbol{A}_{i1})^{-1} (\mathbf{b}_1 - \textstyle\sum_{i=2}^{N} \boldsymbol{A}_{1i}\boldsymbol{A}_{ii}^{-1}\mathbf{b}_i) \quad (2)$$

$$\mathbf{x}_i = \boldsymbol{A}_{ii}^{-1}(\mathbf{b}_i - \boldsymbol{A}_{i1}\mathbf{x}_1), i \in \{2, \dots, N\} \quad (3)$$

Based on Equations (2) and (3), the following section will discuss and derive a distributed parallel scheme for the BBD solver.

III. Distributed Paralleism

A. Background

The previous section identified the practical use case and highlighted opportunities for leveraging parallelism to implement it on computing devices equipped with GPUs, such as high-performance computers (HPCs). HPCs are typically composed of both CPUs and GPUs. During EMT simulations, data preparation and system updates are often performed on CPUs, which communicate with GPUs via the Peripheral Component Interconnect Express (PCIe) bus. This communication introduces latency. If some computations are implemented on GPUs, data may need to be copied back to the CPU for system updates or further processing. Additionally, if multiple GPUs are used, data may need to be transferred between GPUs. In the case of Nvidia hardware devices, GPUs communicate with each other through high-speed hardware interconnects, such as NVLink. For clarity, data preparation can encompass multiple tasks, which may include the formation of the system of linear equations. Therefore, optimizing data transfer during simulations is critically important to minimize latency and maximize overall performance.

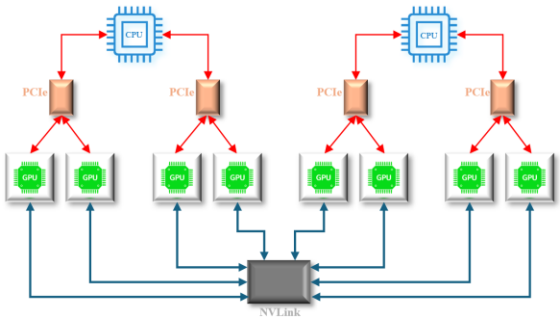

Fig. 2: Illustration of computing architecture

Each GPU consists of many cores, and the number of cores in a GPU is typically much higher than that of a CPU. In Nvidia GPUs, a CUDA core is the smallest computational unit where a parallel thread is executed, and 32 CUDA cores are grouped together to form a warp. In NVIDIA A100 GPU, as illustrated in Fig. 2, 4 warps form a streaming multiprocessor (SM), which is the upper layer of the warp. This GPU model contains a total of 108 SMs. A CUDA stream is a sequence of GPU operations that execute in order they are added to the stream. CUDA kernels can be executed concurrently if they are in different streams.

B. Distributed Parallelism

In Section II, the matrix operations required to solve the linear equation $\boldsymbol{Ax=b}$, where $\boldsymbol{A}$ has a BBD structure, are briefly presented. In this subsection, the distributed parallelism scheme implemented on GPUs is also introduced.

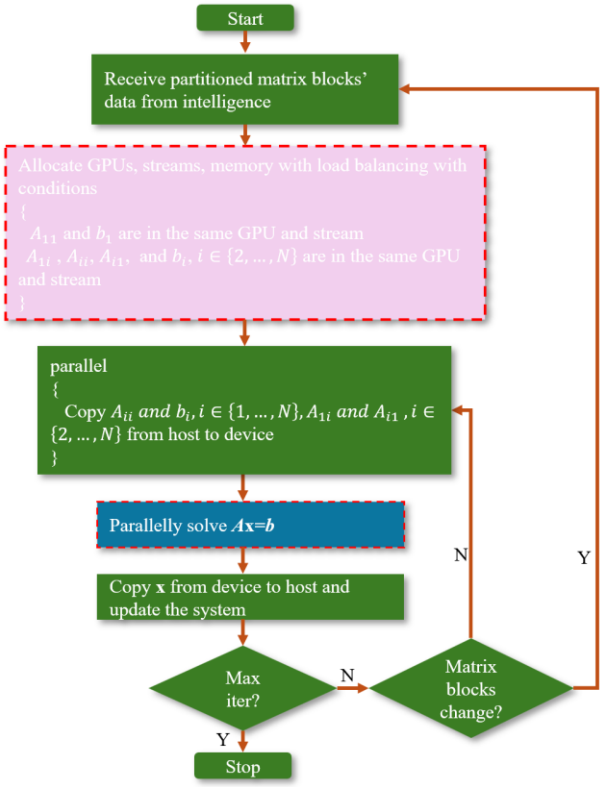

Fig. 3: Illustration of the main steps of the BBD solver

As mentioned earlier, while leveraging the fast matrix operations and processing capabilities of GPUs, there is a trade-off involving the data transfer time between CPUs and GPUs, as well as between GPUs. Therefore, this work aims to minimize the transfer time while maximizing parallel execution. Fig. 3 illustrates the main steps of the BBD solver. An intelligent program detects the matrix structure of $\boldsymbol{A}$, implements reordering, and partitions the matrix into blocks. If the matrix exhibits the BBD structure, the BBD solver is invoked, and the program provides information about the matrix blocks to the BBD solver. After receiving partitioned matrix blocks' data, the solver allocates GPUs, streams, and memory such a way that $\boldsymbol{A}_{11}$ and $\boldsymbol{b}_1$ are in the same GPU and stream, $\boldsymbol{A}_{1i}$, $\boldsymbol{A}_{ii}$, $\boldsymbol{A}_{i1}$, and $\boldsymbol{b}_i$, $i \in \{2, \dots, N\}$ are in the same GPU and stream. The underlying idea behind this allocation is to maximize efficiency by allocating matrices and vectors on the same GPUs and streams as much as possible, while ensuring load balancing across GPUs and streams during the calculations that require them. To optimize efficiency, load balancing is applied to evenly distribute workloads across GPUs and streams. Additionally, temporary auxiliary variables, matrices, and vectors are created to store values required for subsequent calculations. Fig. 4 illustrates the GPU and stream allocation strategy employed in this work. When new data arrives, GPU_k, the GPU with the smallest workload, is selected and assigned to handle the incoming data. Within the chosen GPU, SM_e, the SM with the least workload, is also allocated handle the incoming data. The definition of workload is flexible and can be adjusted, and the total number of matrix blocks is used as the workload metric.

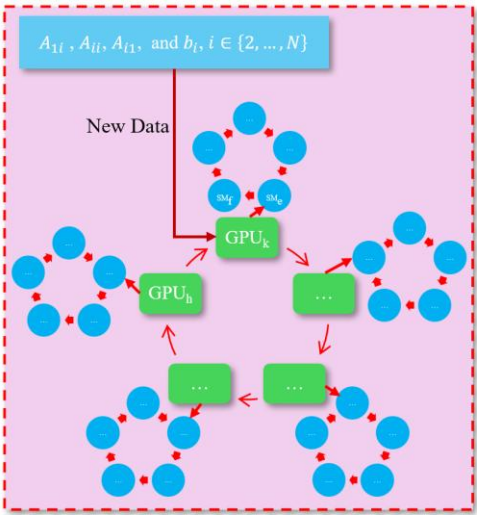

Fig. 4: Illustration of allocating GPUs and streams

After setting up the GPUs and streams, in each iteration, the solver copies data from the CPUs to the allocated memory on the assigned GPUs and streams, solves the linear equations, and copies the solution vector **x** to update the system. It is noted that the GPU and stream configuration step is only required if the matrix blocks change during the simulation iterations. In this work, OpenMP is utilized to implement parallelism on CPUs, taking into account the feasibility and hardware limitations.

Fig. 5 illustrates the main steps involved in solving the linear system equation $\boldsymbol{Ax=b}$ in parallel. In the figure, the GPU and stream associated with each computational task are illustrated. For example, GPU_h and SM_l are used to invert $\boldsymbol{A}_{ii}$. In this work, the cuBLAS and cuSOLVER libraries are utilized for matrix operations, such as matrix multiplication and decomposition. These libraries are optimized for dense matrix operations on GPUs. Additionally, customized kernels are developed to handle tasks such as copying, subtraction, and initialization. These kernels utilize streams that are previously

allocated to correspond to the matrices and vectors they serve. In the presented scheme, certain steps require data to be transferred to different GPUs and streams, such as Step 4 and Step 6. In Step 4, the data is moved to the GPU and stream responsible for handling $\boldsymbol{A}_{11}$ and $\boldsymbol{b}_1$. Additionally, the lower-upper (LU) decomposition method is employed to compute the inverse of $\boldsymbol{A}_{ii}$, as well as $\boldsymbol{A}_{11} - \sum_{i=2}^{N} \boldsymbol{A}_{1i}\boldsymbol{A}_{ii}^{-1}\boldsymbol{A}_{i1}$, where $i \in \{2, \dots, N\}$. In Step 6, $\mathbf{x}_1$ is transferred to the GPU and stream responsible for handling $\boldsymbol{A}_{i1}$ and $\boldsymbol{b}_i$. It is noted that, for generation plants' collectors, the dimension of $\mathbf{x}_1$ can be significantly smaller than that of $\mathbf{x}_i$, where $i \in \{2, \dots, N\}$.

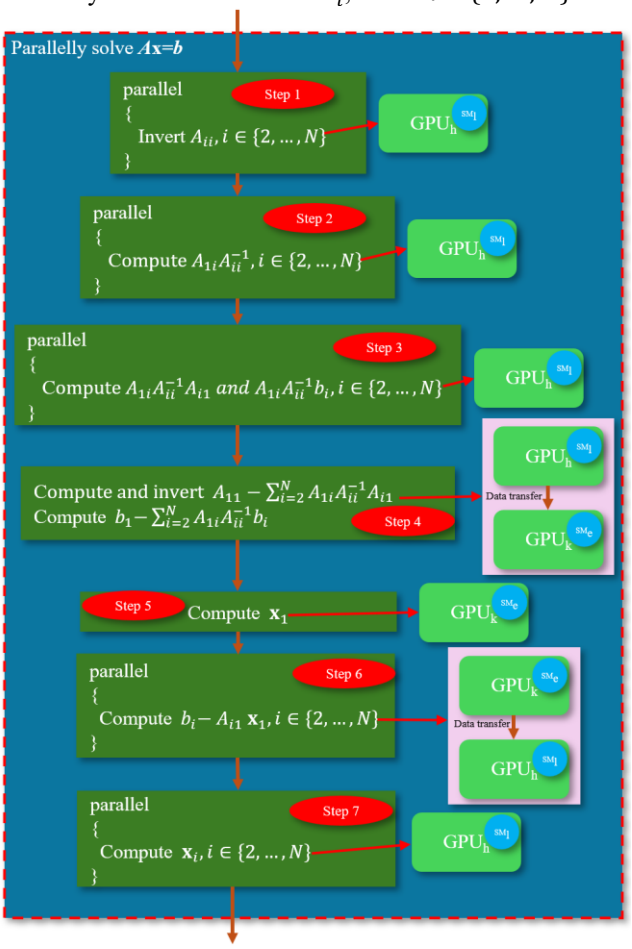

Fig. 5: Illustration of solving the linear system in parallel on GPUs

IV. SIMULATION

In this section, the newly developed BBD solver for EMT simulations is tested and analyzed to evaluate its performance. The solver's implementation is compared between a CPU-only system and a GPU-based system. For clarity and ease of reference, we refer to the GPU-based implementation as the GPU BBD solver and the CPU-based implementation as the CPU solver. It is noted that the CPU solver was parallelized using OpenMP and utilized the LAPACK inverse and BLAS libraries, employing the same number of threads as the GPU versions. The simulations are performed on an NVIDIA DGX A100 system, with both solvers, the GPU BBD solver and the CPU solver, executed on the same machine. The DGX A100 system is equipped with 8 NVIDIA A100 GPUs and two AMD Rome 7742 CPUs. The GPU BBD solver supports two options: single GPU and multiple GPUs. In the multiple GPU option, computations are distributed across multiple GPUs. Matrices and vectors are divided among the GPUs with load balancing to ensure efficient utilization of resources. In contrast, the single GPU option performs all computations on a single GPU, with all matrices and vectors stored and processed within that GPU.

The RE-INTERGATE EMT simulation platform [10], which was developed at Oak Ridge National Laboratory, was used to generate matrices from a generation plant's collector, which was introduced in [11]. To create the generation plant's collector, an OpenDSS file was utilized and parsed within the platform. The OpenDSS file contains information about power lines, transformers, and generation sources. More detailed parameters of inverters were obtained from Extensible Markup Language (XML) files, which were also parsed into the platform. The matrix $\boldsymbol{A}$ is generated by the platform after parsing the files, and the intelligence further partitions the matrix into smaller matrix blocks, forming a BBD matrix structure. The submatrices $\boldsymbol{A}_{11}$, $\boldsymbol{A}_{12}$, $\boldsymbol{A}_{21}$, and $\boldsymbol{A}_{22}$, with dimensions 29×29, 29×475, 475×29, and 475×475 respectively, were extracted and utilized for performance testing and comparison. In this simulation, $\boldsymbol{A}_{i1}$ and $\boldsymbol{A}_{1i}$, $i \in \{3, \dots, N\}$, are assigned the values of $\boldsymbol{A}_{21}$ and $\boldsymbol{A}_{12}$, respectively. This setup was designed to simplify testing the impact of N on the performance of the solvers.

The parameter N was varied from 2 to 15. For each value of N, 10 iterations were executed. The values of $\boldsymbol{b}_i$, where $i \in \{1, \dots, N\}$, were fixed across all iterations. The runtime for each iteration was recorded. For the GPU solver versions, the recorded runtime includes the data transfer time between the host and the device, as well as vice versa. To represent the simulation time for a specific value of N, the average runtime of the last 9 iterations was computed. The runtime of the first iteration was excluded from the averaging process because it encompasses the context setup for GPUs and the allocation of threads on CPUs, which is typically higher than the runtimes of subsequent iterations for both the CPU and GPU BBD solvers. For EMT simulations, where the number of iterations is large, excluding the first iteration is acceptable as it does not significantly affect the overall results. While this setup is not sufficient for a comprehensive comparison of CPU and GPU performance, it offers a basic assessment of the performance of GPU and CPU models, which rank among the highest-performing in their respective categories, specifically on the BBD solver. Fig. 6 shows the performance of the BBD solvers. As shown, solving the linear equation using multiple GPUs provides significant benefits and achieves the best overall performance. On the other hand, solving the linear equation on a single GPU offers improved performance compared to the CPU version. However, as the number of blocks increases beyond a certain number, the single GPU approach becomes slower than the CPU version.

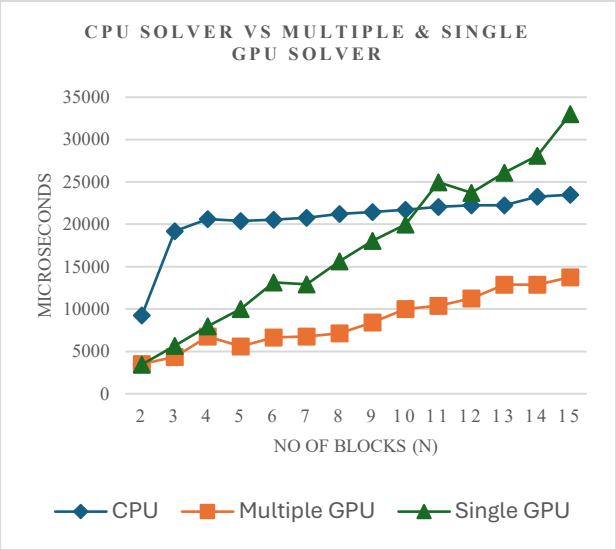

Fig. 6: Performance of BBD solvers

The runtime for each step, as depicted in Fig. 6, was recorded to analyze the performance of individual steps. To avoid including the time spent recording each step into files, these simulations were conducted separately from the previous simulations used to measure the overall performance of solving the entire linear system on GPUs and streams. Fig. 7 presents the performance of each step in the computation process. Step 1, which involves inverting diagonal matrices, has the highest runtime. The host-to-device (H2D) data transfer step accounts for the second-largest runtime overall. In contrast, the device-to-host (D2H) data transfer takes less time than the H2D step, as only the results (i.e., the solution vector **x**) are transferred back to the host.

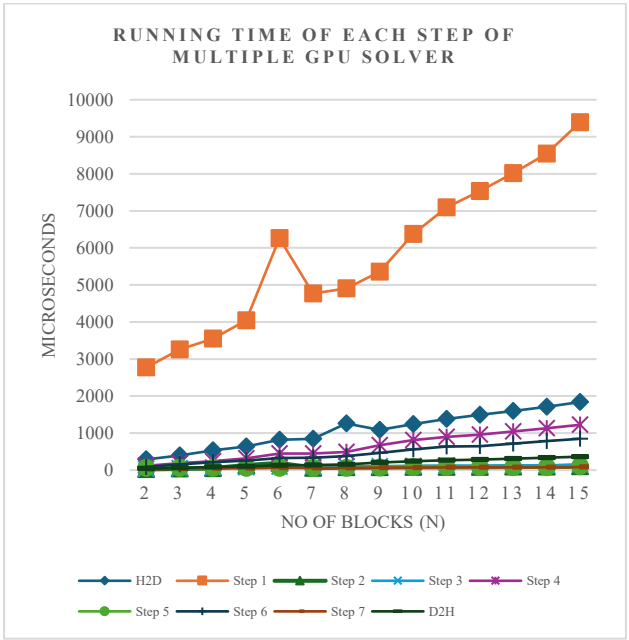

Fig. 7: Performance of the GPU BBD solver using multiple GPUs.

V. CONCLUSIONS

This paper introduces a distributed parallelism scheme for the BBD solver utilizing multiple GPUs and streams. The simulation results highlight significant performance benefits of the multi-GPU approach compared to both single-GPU and CPU implementations. These improvements are particularly critical for EMT simulations, which are known for their time consuming nature. This addresses the need to improve the time taken to simulate in a high-performance EMT simulation tool that is capable of meeting the growing requirements and demands from EMT modeling. The current implementation is developed for a single-node machine. Future work will focus on extending the implementation to multi-node systems to leverage distributed computational capabilities and achieve greater scalability. Another potential direction for future work is to investigate sparse matrix operations to further evaluate and enhance the performance of the BBD solver to reduce the inversion times observed.

REFERENCES

[1] Southwest Power Pool, "SPP electromagnetic transient (EMT) model requirements: For inverter-based resource interconnection," March 2023.

[2] North American Electric Reliability Corporation, "Reliability guideline electromagnetic transient modeling for BPS connected inverter-based resources-recommended model requirements and verification practices," March 2023.

[3] M. R. Hossain, Q. Xie and S. Debnath, "RE-INTEGRATE EMT Simulation software: Graph convolutional neural network for sparse matrix pattern detection," in *2025 IEEE Power & Energy Society General Meeting*, 2025.

[4] J. Choi and S. Debnath, "Linear solvers for collector systems of generalized large-scale inverter-based resources," in *2024 IEEE Power & Energy Society General Meeting*, 2024.

[5] IEEE, "IEEE Standard for interconnection and interoperability of inverter based resources (IBRs) interconnecting with associated transmission electric power systems," IEEE Std 2800-2022, 2022.

[6] North American Electric Reliability Corporation, "NERC DRAFT white paper: Electromagnetic transient analysis in operations planning," June 2025.

[7] D. Aluthge, I. Jeffrey, S. Filizadeh and D. Muthumuni, "Accelerating electromagnetic transient simulations using graphical processing units," *Electric Power Systems Research,* vol. 252, 2026.

[8] J. Sun, S. Debnath, M. Saeedifard and P. R. V. Marthi, "Real-Time electromagnetic transient simulation of multi-terminal HVDC–AC grids based on GPU," *IEEE Transactions on Industrial Electronics,* vol. 68, no. 8, pp. 7002-7011, August 2021.

[9] Y. Song, Y. Chen, S. Huang, Y. Xu and W. Xue, "Efficient GPU-based electromagnetic transient simulation for power systems with thread-oriented transformation and automatic code generation," *IEEE Access,* vol. 6, pp. 25724-25736, 2018.

[10] P. R. V. Marthi, S. Gangopadhyay, J. Choi, S. Debnath and N. Chaudhuri, "RE-INTEGRATE EMT Simulation software: DAE solvers and automation," in *2025 IEEE Power & Energy Society General Meeting*, 2025.

[11] M. F. M. Montejano, P. R. V. Marthi, S. Debnath, S. Samanta and N. R. Chaudhuri, "A high-fidelity electromagnetic transient model of inverter-based resources integrated to an IEEE-9 bus system for benchmarking studies," in *2024 IEEE Power & Energy Society Innovative Smart Grid Technologies*.