The Analysis of the Sparse Multi-GPU Parallel Method on the Large Sparse Power Flow Calculation

Lei Zeng
Electrical & Computer Engineering
Oakland University
Rochester, MI, USA
leizeng@oakland.edu

Shadi Alawneh
Electrical & Computer Engineering
Oakland University
Rochester, MI, USA
shadialawneh@oakland.edu

***Abstract*—Power Flow (PF) calculation serves to analyze the power system efficiently, aiming to obtain the magnitude voltage and phase angle, as the development of multiple energy supplies. Due to the high sparsity and increasingly complexity of power systems, solving large-scale sparse systems in a parallelized fashion has become a bottleneck for power engineers. To address this challenge, this paper proposes a sparse multi-GPU Fast Decouple (FD) method to accelerate the PF calculation. Specifically, data parallelism is designed to enhance scalability and maintain the load balancing across the multi-GPUs. Additionally, GPUDirect technology is employed to reduce the communication overhead between multiple GPUs. As a result, this method in the paper achieves nearly 4x on a power system with over 10, 000 buses, compared to the MATLAB-based optimized library, MATPOWER, on a large-scale power system.**

Keywords—Multi-GPU, FD method, sparse matrix, data parallelism

I. INTRODUCTION

PF calculation is critical for the power system analysis, planning and operation. The PF calculation is applied to solve the nonlinear equations of such power systems [1], [2]. Due to the introduction of new energy supplies and heavier loading, it brings great challenges and pressure for PF calculation [3]. In practice, the classic Newton-Raphson (NR) method is utilized to evaluate the voltage magnitude and phase angle of buses inside the power system, and the sparse NR method is often employed on a single CPU or GPU platform, the accelerating efficacy is limited due to the high data interdependence and irregular sparse format. Furthermore, this limitation in the computation architecture severely hampers the efficiency, parallelism and scalability in PF calculation. As a result, it is a necessity to exploit data parallelism and improve scalability of sparse matrix formats to accelerate PF calculation, particularly in the large-scale and complicated power systems.

High performance computing (HPC) cluster provides substantial fine-grained parallelism with orders of magnitude higher throughput than the CPUs [4]. For instance, each GPU node has 4 NVIDIA Tesla V100 GPUs with 5120 cores in Matilda HPC cluster, which has a peak performance of over 15 TFLOPS versus about 200-400 GFLOPS of Intel i9 series 8 core CPUs [5]. To reduce communication overhead, GPUDirect RDMA is introduced to facilitate the communication among NVIDIA GPUs in remote systems, which eliminates the system CPUs and required buffer copies of data via the system memory, resulting in 10x better performance [6]. Furthermore, NVIDIA offers a range of highly-optimized libraries [7], such as cuBLAS, cuSPARSE, and cuSOLVER. These salient features of high-performance parallel computing systems provides researchers an opportunity to explore data parallelism and devise the parallel sparse computing architecture [6].

In practice, most methods in PF calculation can generally be categorized into two groups, i.e., iteration methods [8-12] and direct methods [13-23]. Specifically, although the iteration method can save much memory, the execution time severely depends on the preconditioning matrix. Generally, the initial preconditioning matrix can easily lead to the PF calculation divergence, compelling researchers repeatedly to test the different preconditioning matrices until finding a stable and robust preconditioner for the PF calculation. The direct solver, on the other hand, tends to be more robust for ill-conditioned problems compared to the iteration methods [7]. As for direct methods, most utilize the single GPU approaches to accelerate the PF calculation in a vectorization manner, improving the performance of PF calculation. Reference [3] takes the sparse NR method and achieves a 1.4x speedup, compared to its MATLAB counterpart. However, the accelerating efficacy is limited, due to the high data dependency and irregular memory access in the sparse matrix format [24]. Reference [22] introduces a multi-thread and multi-GPU method to accelerate the large-scale PF calculation. This method, however, suffers from severe data race issues in shared memory programming fashion, leading to performance degradation, achieving only a modest 2x improvement when compared to the optimized CPU counterpart.

Sooknanan and Singh et al. [14], [25] introduce the sing-GPU method, employing a dense matrix format, which indeed accelerates the speedup of PF calculation and enhances the scalability of computing architecture. These methods, however, may fail to converge as the power system scales up, due to the limited global memory of the single GPU. Besides, massive nonzero elements in the dense matrix are involved in the process of linear system solver, leading to significant resource consumption and increased the execution time of PF calculation. Reference [6] proposes the multi-process and multi-GPU method improve the scalability of the computing architecture. The approach achieves notable speedups ranging from 9x ~ 33x, compared to the single-GPU counterpart on dense large-scale power systems. The method reduces the data

race when operating on the same elements, but it still faces the loading unbalanced challenges, similar to those encountered in [14], [25].

To address such challenges, this paper proposes a sparse multi-GPU parallel computing method, aimed at reducing the involvement of nonzero elements in the PF calculation and leveraging the floating-operating capability of multi-GPUs, leading to performance improvement in both acceleration and scalability. Furthermore, GPUDirect technology is employed to alleviate communication overhead among multiple GPUs by facilitating direct data transfer between them in a peer-to-peer (P2P) manner. In addition, one of GPUs is designated to broadcast and distribute the intermediate data to other GPUs, thereby simplifying the complexity of management and scheduling for the computing system.

The rest of this paper is organized as follows: Section II reviews the heterogeneous HPC cluster and introduces the sparse multi-GPU method. In Section III, this paper presents implementation details of this method. Section IV presents the benchmarking results and analysis. Finally, the overall conclusion and future work will be presented in the last section.

II. HPC CLUSTER AND SPARSE FD MULTI-GPU METHOD

In this section, the heterogeneous HPC cluster mode is reviewed, outlining its features and components. Subsequently, the sparse multi-GPU method is introduced, detailing its strategies for accelerating PF calculation on such a computing architecture.

A. Heterogenous HPC cluster

HPC cluster can be abstracted as a heterogeneous programming model that consists of hosts and devices. Fig. 1 describes the HPC cluster architecture in the 2D fashion, showcasing its components and their interconnections.

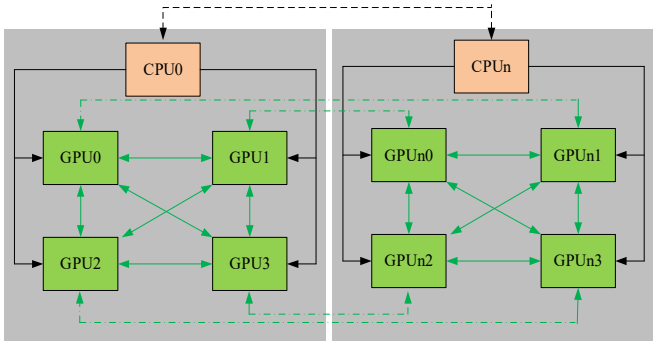

Fig. 1. HPC cluster heterogeneous architecture.

Each node in the HPC cluster is grouped through host and devices, respectively. The host, CPU, is responsible for data preconditioning and scheduling. The computationally intensive tasks are then offloaded to the GPUs. Specifically, each GPU will simultaneously initiate numerous threads upon invoking the CUDA kernel function, which lays the foundation for parallelization to accelerate the PF calculation. These threads are organized into two-level hierarchy, namely grid and block, while each level supports up to 3-dimensions [22]. At the grid level, multiple thread blocks are grouped into a grid and then massive threads are assigned to a block. A warp is formed by 32 consecutive threads within a block, and all threads in a warp execute instructions in a lock-step manner in accordance with the Single Program Multiple Data (SPMD) concept [26]. However, divergences can occur when the parts of threads encounter the if-statement within the same warp, limiting the program parallelization. Furthermore, the main bottleneck becomes communication overhead between GPUs rather than computing capability of individual GPUs in the multi-GPU architecture. Therefore, GPUDirect technology is utilized to facilitate P2P communication among GPUs, reducing the cost of data migration, and resulting in performance improvement. To further exploit the capability of multi-GPUs, a data parallelism approach is adopted, and the data tiles are evenly distributed across the multi-GPU platforms to maintain the loading balance. One of the GPUs is responsible for the data broadcast and distribution, minimizing the back-and-forth transmission between host and devices and maximizing the parallel processing potential of the devices.

B. Sparse Multi-GPU FD Method

The sparse FD method, an extension of the NR method originally proposed by Stott and Alsac [27], simplifies the calculation of Jacobian matrix in the NR framework. As a result, the classic NR formulas can be expressed as follows:

$$\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}=\begin{bmatrix}J_1 & J_2\\ J_3 & J_4\end{bmatrix}\begin{bmatrix}\Delta\delta\\ \Delta V\end{bmatrix} \tag{1}$$

Where ΔP and ΔQ represent the real and reactive power mismatch, respectively. J_1, J_2, J_3 and J_4 denote the partial derivatives with respect to voltage phase angle, $\Delta\delta$, and magnitude ΔV. Due to the high X/R ration of transmission line in the power systems, the submatrices, J_2 and J_3, are approximately zero. With such approximation, equation (1) can be modified as follows:

$$\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix}=\begin{bmatrix}J_1 & 0\\ 0 & J_4\end{bmatrix}\begin{bmatrix}\Delta\delta\\ \Delta V\end{bmatrix} \tag{2}$$

Therefore, equation (2) can be further split into two independent equations, as follows:

$$\Delta P = J_1\Delta\delta \tag{3}$$

$$\Delta Q = J_4\Delta V \tag{4}$$

J_1 and J_4 are stored in compressed sparse row (CSR) format, due to the sparse nature of the power system.

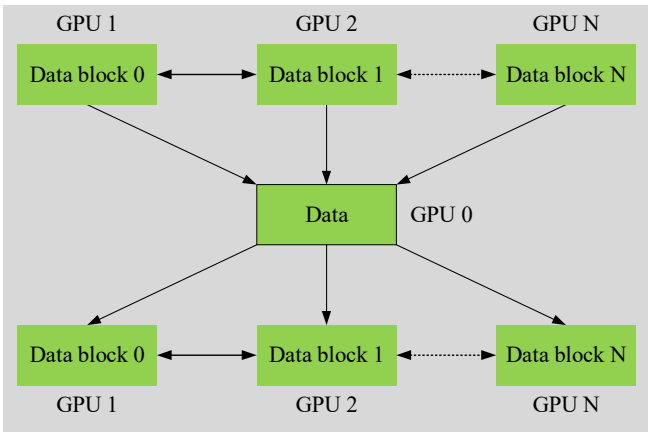

Fig. 2. The data parallelism of the sparse multi-GPU method.

Additionally, the fixed coefficient matrices, J_1 and J_4, are reused throughout the PF calculation process, eliminating the necessity for updating them for each iteration. While the CSR format conserves a considerable amount of computing resources and memory, it restricts the scalability of the multi-GPU architecture, due to its irregular memory access pattern. In practice, the results of (3) and (4) still exhibit dependency. Specifically, the results of one equation can facilitate the other, and vice versa. Therefore, the task parallelism method in [6] will cause the loading unbalanced issues. The sparse data parallelism is introduced to mitigate these challenges. Specifically, the sparse data are evenly divided into numerous consecutive data blocks and distributed across multiple GPUs, with one GPU managing the data collection and distribution, as illustrated in Fig. 2. Furthermore, the sparse elements are stored in the consecutive locations. In the process of the PF calculation, the elements in the consecutive location can be accessed through a single dynamic random-access memory (DRAM) request at extremely high speed. This technique is referred to as DRMA burst, leveraging the aligned coalesced memory access of the GPUs.

III. Implementation

In this section, it presents the specifics of the sparse multi-GPU method, including the program flowchart and implementation details.

A. Triangular Linear System Solver

In the process of PF calculation, equation (3) and (4) can be expanded into a linear system, using the first-order Taylor series at the fixed point, and then they are represented in matrix format, as follows:

$$Ax = b \tag{5}$$

Where A and b are Jacobian matrix and power mismatch, respectively. The state vector, x, is either the voltage magnitude or phase angle. Due to the nature of power systems, LU solver is a high efficiency numerical method for the numerical stability and acceleration in the process of PF calculation [2]. Therefore, the Jacobian matrix, A, can be decomposed into lower (L) matrix and upper (U) matrix before transferring them to the multi-GPUs, as follows:

$$A = LU \tag{6}$$

Algorithm 1 Triangular Linear System Solver in PF Calculation

```
1 Collect the results from other GPUs to GPU0
2 //Request for workspace
  cusparseDcsrsv_bufferSize(handle, trans, m, nnz, descr, \
                            csrV, csrP, csrI, \
                            info, &Size);
3 if (Size != 0) then
     //allocate the global memory
     cudaMalloc((void**)&pBuffer, Size);
  end
4 //Analysis of the sparse matrix
  cusparseDcsrsv2_analysis(handle, trans, m, nnz, descr, \
                           csrV, csrP, csrI, \
                           info, policy, pBuffer);
5 //Solve the triangular linear systems
  cusparseDcsrsv2_solve(handle, trans, m, nnz,  &alpha, \
                        descr, csrV, csrP, csrI, \
                        info, b, x, \
                        policy, pBuffer);
6 Broadcast the solutions to other GPUs
7 Update state vectors on the multi-GPUs
```

The L and U matrices, resulting from the decomposition of the Jacobian matrix, will be transferred to multi-GPU global memory, and utilized repeatedly in each iteration of the PF calculation, eliminating the need to update the Jacobian matrix during the calculation process. The highly optimized CUDA library, CUSPARSE, further speeds up the solution of the sparse linear equations, as outlined in Algorithm 1.

B. The Flowchart of the Sparse Multi-GPU Method

The PF calculation process includes several steps, outlined below, with a corresponding flowchart depicted in Fig. 3.

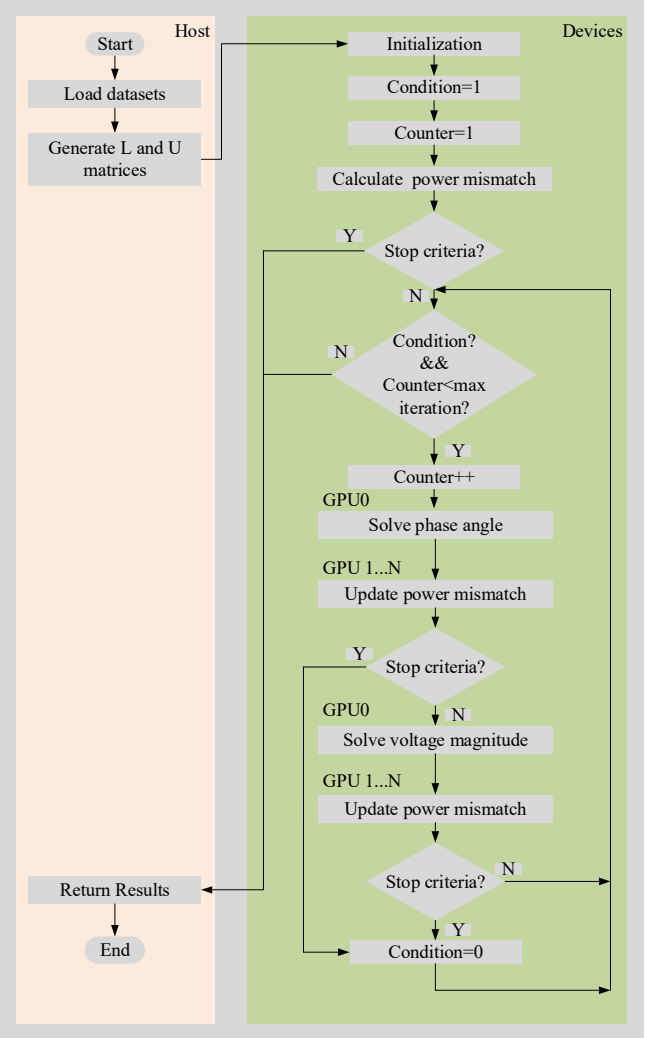

Fig. 3. The flowchart of the sparse multi-GPU method.

- Preprocess: Sparse data is processed and decomposed into LU matrices and reused on multi-GPUs.
- Data Transfer: Sparse data and preconditioner are evenly transferred to the multi-GPU global memory.
- State Vector Solution: the state vectors, phase angle, are solved and power mismatch is updated on the multi-GPUs.
- Convergence Check: The norm of the power mismatch is checked to determine if it satisfies the stop criteria.
- Termination: The process terminates if the stop criteria are satisfied, otherwise 6).

- State Vector Solution: The state vectors, including voltage magnitude, are solved and power mismatch is updated on the multi-GPUs.
- Convergence Check: The norm of power mismatch is checked to determine if it satisfies the stop criteria.
- Termination: Stop when the stop criteria are satisfied, otherwise 9).
- Iteration: Steps from 3) to 8) are repeated until the stop criteria are met.

IV. Result and Analysis

This subsection introduces the test setup and presents the results of the sparse multi-GPU method, comparing it to the CPU counterpart. The comprehensive analysis of the results is discussed in the subsequent subsection.

A. Experiment Setup

The overall experiments were carried out on the Matilda HPC cluster, which is equipped with 4 NVIDIA Tesla V100 GPU nodes. For software libraries, the CUDA toolkit version 10.2 is utilized to accelerate the PF calculation. The convergence tolerance of $1e-4$ was set to ensure both the speed and accuracy in the PF calculation. Table I presents the detailed parameters of the computing platform.

TABLE I
Test Platform Parameters

Item	Description
CPU	Intel(R) Xeon® 2.5GHz
GPU	NVIDIA Tesla V100
OS	Red Hat Enterprise 8.6
Complier	NVCC
CUDA Tool	CUDA Toolkit 10.2
Library	Eigen 3.4 / MATPOWER 7.1

Also, all the test cases are sourced from the MATPOWER, as presented in Table II.

TABLE II
Large-scale Test Cases

Case	Description
Case 1354	Test case form MATPOWER, Pan-European system
Case 3120	Test case form MATPOWER, Polish system
Case 6515	Test case form MATPOWER, French system
Case 9241	Test case form MATPOWER, Pan-European system
Case 13659	Test case form MATPOWER, Pan-European system

B. Results of Implementation

The sparse multi-GPU method was executed on the HPC cluster with varying numbers of GPUs in comparison with the optimized MATPOWER. Table III presents the implementation results of the sparse multi-GPU method, including the execution time and speedup.

TABLE III
Execution Results On the Multiple GPUs (unit: second)

Case	Platform	Time	Speedup
Case 1354	CPU	0.029	1.000
	Single GPU	0.442	0.066
	Two GPUs	0.714	0.041
	Three GPUs	1.101	0.026
	Four GPUs	1.418	0.020
Case 3120	CPU	0.109	1.000
	Single GPU	1.624	0.067
	Two GPUs	1.642	0.066
	Three GPUs	1.781	0.061
	Four GPUs	1.992	0.055
Case 6515	CPU	2.384	1.000
	Single GPU	3.684	0.647
	Two GPUs	2.857	0.834
	Three GPUs	3.231	0.738
	Four GPUs	3.557	0.670
Case 9241	CPU	1.770	1.000
	Single GPU	3.151	0.562
	Two GPUs	2.948	0.600
	Three GPUs	4.097	0.432
	Four GPUs	4.112	0.430
Case 13659	CPU	3.428	1.000
	Single GPU	0.958	3.578
	Two GPUs	0.908	3.775
	Three GPUs	1.288	2.661
	Four GPUs	1.495	2.293

From Table III, it suggests that the execution time of the sparse multi-GPU method decreases as the scale of power system increases. Also, while the CPU achieves outperformance, its accelerating efficacy is constrained as the power system exceeds 10, 000 buses.

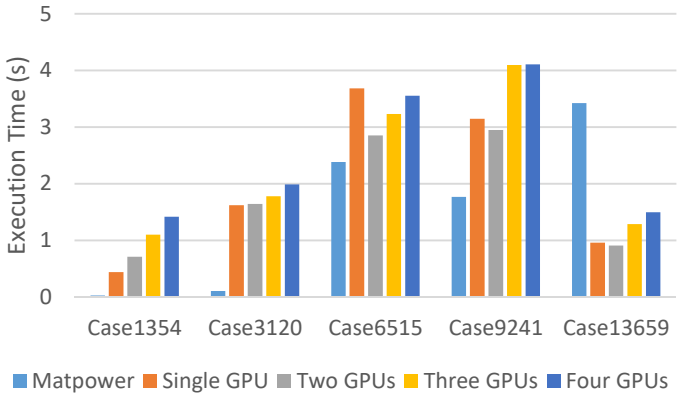

Fig. 4. The execution time on CPU and multi-GPUs.

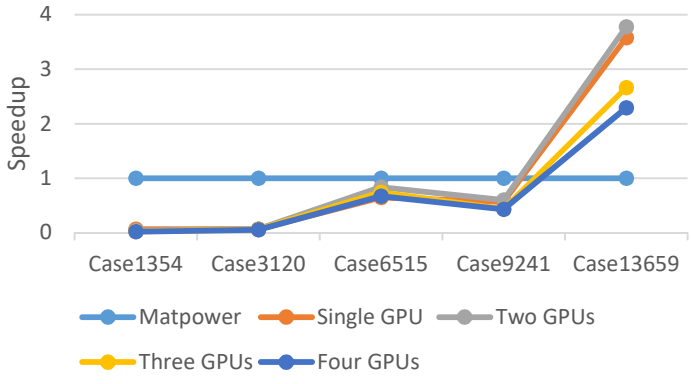

Fig. 5. The speedup of CPU in comparision with multi-GPUs.

Fig. 4 depicts a significant performance enhancement with the multi-GPU method when two GPUs are utilized. However, it also implies that the addition of more GPUs does not necessarily guarantee further performance improvement. Curves in Fig. 5 show the speedup of both CPU and multi-

Identify applicable funding agency here. If none, delete this text box.

GPUs on different power systems. The speedup of the sparse multi-GPU method maintains a consistent improvement until the power system reaches over 9, 000 buses, after which the improvements becomes notably significant. These will be discussed in the next subsection.

C. Analysis of Results

*a) Influence of the sparse format:*The sparsity of power system, especially those with over 1,000 buses, typically reaches around 97% [3]. Consequently, the sparse power system data is stored in the CSR format, where the non-zero elements are packed in triplets, significantly reducing much memory usage. As a result, CPU can outperform GPUs when the number of power system buses is less than 10, 000. Massive zeros in the data are not involved in the PF calculation, when taking the CSR format. However, the design philosophy of the GPU is based on the throughput-oriented principle, meaning that the limited small-scale size of the CSR data fails to fully leverage the high floating-point computing capabilities of the GPU. As a results, significant improvements over multi-GPU performance may not be fully realized, even when dealing with the power systems exceeding 9,000 buses.

*b) Influence of the GPU numbers:*Theoretically, the speedup of the multi-GPU performance will improve as the increasing number of GPUs. However, curves in Fig. 5 suggest that the sparse multi-GPU method achieves the outperformance on the parallel devices with just two GPUs. In practice, while the multi-GPU method is well-suited for the large-scale and data-parallelism programs because it can hide the inter-communication latency among the GPUs through intelligently scheduling computing resources. Since the sparse format and compressed small-scale data are employed, the cost of communication experiences a significant increase with the addition of more GPUs. That is why a single GPU outperforms configurations with three and four GPUs when the number of power buses is less than 9,000. Furthermore, some synchronization operations in the sparse multi-GPU method are utilized to guranteen the result security and accuracy during the PF calculation, causing an increase in execution time. Consequently, the limitations of small-scale sparse data hinder the full utilization of multi-GPU capabilities, increasing the communication burden. In spite of using GPUDirect technology to facilitate P2P communication among GPUs, the transparent data transmission still fails fully to mitigate the communication disadvantages inherent in the sparse compressed small-scale datasets. Nevertheless, considering the observed trend in Fig. 5, it is expected that the multi-GPU method will eventually achieve overall performance enhancement in future power networks, particularly as the current power systems continue to expand in size and complexity.

V. Conclusion and Future Work

This paper introduces a sparse multi-GPU method, designed to accelerate the PF calculation through data parallelism. As a result, it enhances the scalability of parallel computing architecture. By leveraging the CSR format, this method efficiently utilizes and saves computing resources excluding the zero-entry calculation in the coefficient matrix during the PF calculation. Furthermore, it overcomes the disadvantages of loading unbalance in [6], achieving almost a speedup of almost 4x compared to the optimized MATPOWER counterpart.

To reduce the communication overhead, GPUDirect technology is utilized to facilitate P2P communication between multi-GPUs. Although it reduces the cost of communication between multiple devices, the accelerating efficacy is limited when the number of GPUs exceeds two. During the state vector update, some synchronization operations consume much time for the result security and accuracy. Especially, scheduling of multi-GPU fails to hide the latency in small-sized power systems. Consequently, inter-GPU communication becomes the main limited factor rather than the computing capabilities of the devices. Future work will concentrate on mitigating the communication overhead among multi-GPUs, with a focus on leveraging NVSHEM to enhance the efficiency of the multi-GPU communication, ultimately leading to overall performance enhancement.

References

[1] Yoon, D.H. and Han, Y., "Parallel power flow computation trends and applications: A review focusing on GPU," Energies, vol. 13, pp. 2147, 2020.

[2] Alawneh, S.G., Zeng, L. and Arefifar, S.A., "A Review of High-Performance Computing Methods for Power Flow Analysis," Mathematics, vol. 11, pp. 2461, 2023.

[3] Zeng, L., Alawneh, S.G. and Arefifar, S.A., "GPU-based sparse power flow studies with modified Newton's method," IEEE Access, vol. 9, pp.153226-153239, 2021.

[4] Peng S., and Tan SX., "GLU3. 0: Fast GPU-based parallel sparse LU factorization for circuit simulation," IEEE Design & Test. Vol. 37, pp. 78-90, Feb 2020.

[5] B. David Kirk and Wen-Mei Hwu, Programming Massively Parallel Processors: A Hands-on Approach, 2nd ed. Morgan Kaufmann Publiishers Inc., San Francisco, CA, 2013.

[6] Zeng, L., Alawneh, S.G., and Arefifar, S.A., "Parallel multi-GPU implementation of fast decoupled power flow solver with hybrid architecture," Cluster Computing, pp. 11-12, Jun 2023.

[7] Huang, S., and Dinavahi V., "Performance analysis of GPU-accelerated fast decoupled power flow using direct linear solver," IEEE Electrical Power and Energy Conference, pp. 1-6, Oct 2017.

[8] Y. Saad, Iterative Methods for sparse Linear Systems, 2nd ed. PWS:Boston, MA, 2004.

[9] Green, RC., Wang, L., and Alam, M., "High performance computing for electric power systems: Applications and trends," IEEE Power and Energy Society general meeting, pp. 1-8, Jul 2011.

[10] Dag, H., and Semlyen, A., "A new preconditioned conjugate gradient power flow," IEEE Transactions on Power Systems, vol. 18, pp. 1248-55, Nov 2003.

[11] Chen, Y., and Shen, C., "A Jacobian-free Newton-GMRES (m) method with adaptive preconditioner and its application for power flow calculations," IEEE Transactions on Power Systems, vol. 21, pp. 1096-103, Jul 2006.

[12] Flueck, A.J., and Chiang, H.D., "Solving the nonlinear power flow equations with a Newton process and GMRES," IEEE International Symposium on Circuits and Systems, vol. 1, pp. 657-660, May 1996.

[13] T.A., Davis, Direct Methods for Sparse Linear Systems, SIAM: Philadelphia, PA, USA, 2006.

[14] Singh, J., and Aruni, I., "Accelerating power flow studies on graphics processing unit," Annual IEEE India Conference, pp. 1-5, Dec 2010.

[15] Guo, C., Jiang, B., Yuan, H., Yang, Z., Wang, L., and Ren, S., "Performance comparisons of parallel power flow solvers on GPU system," IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, pp. 232-239, Aug 2012.

[16] Li, X., Li, F., and Clark, J.M., "Exploration of multifrontal method with GPU in power flow computation," IEEE Power & Energy Society General Meeting, pp. 1-5, Jul 2013.

[17] Gopal, A., Niebur, D., and Venkatasubramanian, S., "DC power flow based contingency analysis using graphics processing units," IEEE Lausanne Power Tech, pp. 731-736, Jul 2007.

[18] Su, X., He, C., Liu, T., and Wu, L., “Full parallel power flow solution: A gpu-cpu-based vectorization parallelization and sparse techniques for newton–raphson implementation,” IEEE Transactions on Smart Grid, vol. 11, pp. 1833-44, Sep 2019.

[19] Huang, S., and Dinavahi, V., “Performance analysis of GPU-accelerated fast decoupled power flow using direct linear solver,” IEEE Electrical Power and Energy Conference, pp. 1-6, Oct 2017.

[20] Schäfer, F., and Braun, M., “An efficient open-source implementation to compute the Jacobian matrix for the Newton-Raphson power flow algorithm,” IEEE PES Innovative Smart Grid Technologies Conference Europe, pp. 1-6, Oct 2018.

[21] Gnanavignesh, R., and Shenoy, U.J., “Parallel sparse LU factorization of power flow Jacobian using GPU,” IEEE Region 10 Conference, pp. 1857-1862, Oct 2019.

[22] Wang, Z., Wende-von Berg, S. and Braun, M., “Fast parallel Newton–Raphson power flow solver for large number of system calculations with CPU and GPU,” Sustainable Energy, Grids and Networks, vol. 27, pp.100483, Sep 2021.

[23] Chen, X., Wu, W., Wang, Y., Yu, H. and Yang, H., “An escheduler-based data dependence analysis and task scheduling for parallel circuit simulation,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 58, pp.702-706, Sep 2011.

[24] He, K., Tan, S.X.D., Wang, H. and Shi, G., “GPU-accelerated parallel sparse LU factorization method for fast circuit analysis,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 24, pp.1140-1150, Apr 2015.

[25] Sooknanan, D.J. and Joshi, A., “GPU computing using CUDA in the deployment of smart grids. SAI Computing Conference,” pp. 1260-1266, Jul 2016.

[26] S. Cook, CUDA Programming: A Developer’s Guide to Parallel Computing with GPUs. Waltham, MA: Morgan Kaufmann, 2012.

[27] Stoot, B., Alsac, O., "Fast decoupled load flow." Power Appar. Syst., pp. 859-869, May 1974.