

# GPU-Accelerated Optimization Solver for Unit Commitment in Large-Scale Power Grids

Hussein Sharadga

*School of Engineering*

*Texas A&M International University*

Laredo, Texas, USA

hussein.sharadga@tamu.edu

Javad Mohammadi

*Civil, Architectural and Environmental Engineering*

*The University of Texas at Austin*

Austin, Texas, USA

javadm@utexas.edu

**Abstract**—This work presents a GPU-accelerated solver for the unit commitment (UC) problem in large-scale power grids. The solver uses the Primal-Dual Hybrid Gradient (PDHG) algorithm to efficiently solve the relaxed linear subproblem, achieving faster bound estimation and improved crossover and branch-and-bound convergence compared to conventional CPU-based methods. These improvements significantly reduce the total computation time for the mixed-integer linear UC problem. The proposed approach is validated on large-scale systems, including 4224-, 6049-, and 6717-bus networks with long control horizons and computationally intensive problems, demonstrating substantial speed-ups while maintaining solution quality.

**Index Terms**—Power Grid Optimization, Large-Scale Optimization, Unit Commitment, GPU, Optimal Power Flow (OPF), Primal-Dual Hybrid Gradient (PDHG) Algorithm, Primal-Dual Linear Programming (PDLP)

## I. INTRODUCTION

### A. Motivation

Although the electrical power grid is widely regarded as one of the landmark achievements of 20th-century engineering [1], efficiently managing power distribution continues to be a significant challenge. With the growing complexity of modern power systems, ensuring system reliability and resilience has become increasingly important. To support these objectives, the U.S. Department of Energy (DOE) promotes programs aimed at modernizing grid operations through the adoption of advanced technologies and methodologies [2].

Optimal power flow (OPF) plays a critical role in grid management by determining optimal generator outputs, load allocations, energy storage operation, and other control settings [3], [4]. Security-constrained OPF (SCOPF) further strengthens system reliability by explicitly considering potential contingencies within the operational planning process [5].

Historically, SCOPF methods were developed under limited computational capabilities, often employing linearized models such as DCOPF that ignore voltage magnitudes and reactive power effects [6]. Despite improvements in algorithms and computational power, many existing tools continue to rely on these simplifications, making it challenging to solve the full AC power flow problem [2]. To overcome these limitations, the DOE launched the Grid Optimization (GO) challenges, which focus on solving the full AC power flow problem while accounting for unit commitment and security requirements [2].

### B. Prior Work on the Grid Optimization (GO) Challenge

The GO Challenge series focuses on large-scale nonlinear optimization tasks involving unit commitment decisions, which are known to be computationally demanding. To manage this complexity, several research efforts have introduced advanced optimization techniques aimed at improving scalability and solution performance. For example, Chevalier [7] proposed a parallel Adam-based optimization method that speeds up both backpropagation and variable projection through parallel computation, allowing more efficient processing of large problem instances. In related work, [8] showed that enforcing AC power flow constraints, reserve requirements, and unit commitment can produce reliable solutions for AC-based unit commitment challenges. Additionally, Bienstock and Villagra [9] introduced a linear relaxation framework for ACOPF based on cutting-plane methods, enabling the computation of robust lower bounds through outer-envelope cuts and refined cut management procedures. Holzer et al. [10] evaluated multiple solvers submitted to GO Challenge 3, which addressed a multi-period unit commitment problem with AC network modeling and topology switching, illustrating the comparative performance of different optimization strategies at scale.

In prior work [11], [12], we proposed a solution strategy for the full AC power flow challenge, decomposing it into two sequential subproblems: a DC model that handles the binary unit commitment decisions and feasibility constraints, and an AC model that solves continuous variables under all constraints using a nonconvex solver. This decomposition enables efficient handling of computational workloads while preserving solution quality in large-scale power system optimization under strict time limits.

### C. Contribution

Building on our prior research [11], [12], this paper presents a GPU-accelerated solver for the unit commitment (UC) problem in large-scale power systems. The solver leverages the Primal-Dual Hybrid Gradient (PDHG) algorithm to efficiently solve the relaxed linear subproblem, improving bound estimation as well as crossover and branch-and-bound convergence. By exploiting GPU acceleration, the approach offers a scalable and practical framework for real-time large-scale UC

applications. The method is validated on 4224-, 6049-, and 6717-bus networks, demonstrating substantial computational speed-ups while preserving solution quality.

## II. PROBLEM SETUP

### A. Problem Formulation

This study adopts the problem formulation provided by the DOE Advanced Research Projects Agency-Energy (ARPA-E) for the GO challenge 3. The problem is highly detailed, with the primary formulation spanning 53 pages [13], accompanied by an additional 22 pages of supplementary material. It involves a comprehensive set of optimization variables and constraints, such as limits on real and reactive power at individual bus nodes, voltage bounds, zone-specific reserve requirements, device operational statuses, switching complexities, considerations for device downtime and uptime, the total number of device starts over multiple intervals, ramping constraints, and minimum/maximum energy limits across time periods (refer to Fig. 1 of [11]). Notably, all constraints are applied under both base and contingency scenarios. The contingency conditions ensure the system maintains reliability under unexpected events and outages.

### B. Dataset

This study utilizes the ARPA-E dataset provided in [14]. The 4224-bus, 6049-bus, and 6717-bus networks from Trial 3 were adapted for our analysis. These networks correspond to different scheduling horizons: 1 day ahead (Division 1), 2 days ahead (Division 2), and 1 week ahead (Division 3). The optimization model for the 4224-bus network with Division 1 includes approximately 1 million continuous decision variables, around 116,000 binary variables, and roughly 1 million constraints. The complete counts of variables and constraints for all networks are summarized in Table I. The values in Table I correspond to Division 1 (18 steps). To obtain Division 2 values (48 steps), divide by 18 and multiply by 48. For Division 3 (42 steps), divide by 18 and multiply by 42. In total, 57 scenarios across the three networks and three scheduling horizons were tested in this study.

### C. Reference Model

Our previous model introduced in [11], [12] demonstrated excellent performance, with an average scaled score above 0.98 for a range of network sizes and control settings. The scaled score reflects how closely the obtained solution approaches the best-known outcome, and consistently achieving

TABLE I: Problem scale for different network sizes. The values correspond to Division 1 (18 steps). To obtain Division 2 values (48 steps), divide by 18 and multiply by 48. For Division 3 (42 steps), divide by 18 and multiply by 42.

Parameter	Network		
	4224-Bus	6049-Bus	6717-Bus
# Binary Variables	116,154	67,932	104,868
# Continuous Variables	966,220	1,673,301	2,165,998
# Constraints	1,168,114	1,580,545	2,003,974

0.98 over roughly 300 scenarios highlights the reliability of the prior approach.

In this work, we improve the speed of convergence of the unit commitment module by leveraging GPU-based optimization, solved using the Gurobi solver, as depicted in Figure 1.

### D. Hardware and Computing Setup

Running the PDHG algorithm on a GPU with the Gurobi solver requires NVIDIA hardware (preferably an NVIDIA H100 GPU, as noted in the Gurobi documentation [15]), CUDA version 12.9, and a Linux 64-bit operating system. The GPU-based solver utilizes NVIDIA’s cuOpt engine. While Gurobi is typically supported and widely used on Windows platforms, GPU acceleration is currently available only on Linux systems.

The cuOpt engine is NVIDIA’s GPU-accelerated optimization library designed for large-scale linear and mixed-integer programming. It leverages highly parallel numerical kernels to execute optimization steps efficiently on GPUs, providing substantial speed-ups compared to traditional CPU solvers. In Gurobi’s GPU mode, cuOpt handles the PDHG iterations, while Gurobi oversees higher-level tasks such as presolve, crossover, and branch-and-bound.

In this study, we employed an NVIDIA A800 (40 GB) active GPU configured under Linux64 through the Windows Subsystem for Linux (WSL). The Gurobi 13.0.0 beta 2 version was used. The computing node was equipped with an Intel® Xeon® W9-3495X processor with 56 cores (112 threads), operating at a base frequency of 1.90 GHz and a maximum turbo boost of 4.80 GHz, along with 256 GB of RAM to handle the size and complexity of the datasets. Gurobi was configured to use 64 threads (the default is 32). Further increasing the thread count to the maximum available did not improve performance, likely due to memory bandwidth constraints and synchronization overhead.

Table II summarizes the hardware and software specifications used in our experiments. The GPU and multicore CPU operated collaboratively to accelerate the optimization process. The PDHG algorithm was executed on the GPU, while the remaining steps—such as crossover and branch-and-bound—benefited from multithreading on the CPU. In contrast, when linear programming and barrier methods are employed instead of PDHG, they primarily benefit from CPU-based multithreading rather than GPU acceleration.

TABLE II: Hardware and Software Specifications

Component	Specification
GPU	NVIDIA A800, 40 GB VRAM
GPU Engine	NVIDIA cuOpt library
CPU	Intel® Xeon® W9-3495X, 56 cores / 112 threads, 1.90–4.80 GHz
RAM	256 GB
Operating System	Linux 64-bit (via WSL on Windows)
Python Version	3.11
Gurobi Version	13.0.0 beta 2
CUDA Version	12.9
Gurobi Threads	64 (default 32)

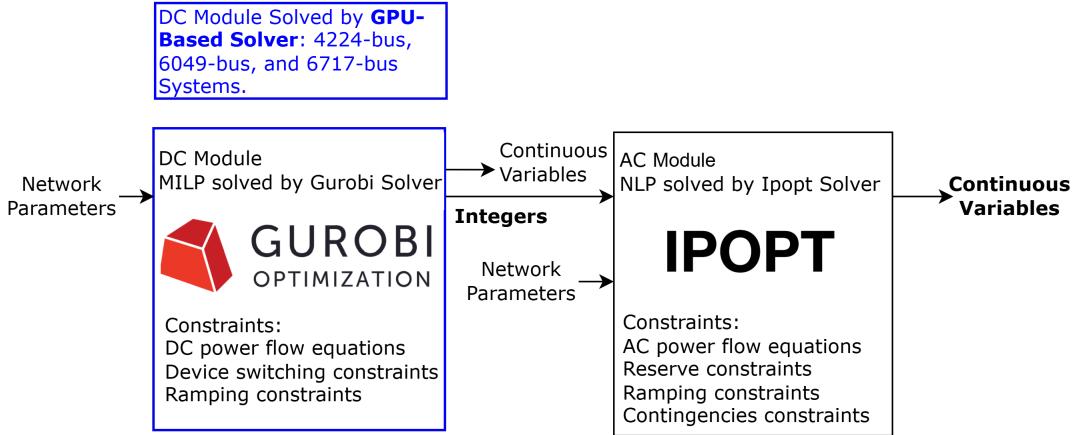


Fig. 1: Our two-stage solution framework integrates the DC and AC modules presented in [11], [12]. In this work, we improve the computational speed of the DC module (first stage) using GPU-based optimization. The GPU-based solver accelerates computations for large-scale networks, including the 4224-, 6049-, and 6717-bus systems.

### E. PDHG Background

For many years, the field of linear programming (LP) has been dominated by two primary algorithms: the Simplex Method and Interior Point Methods (IPMs). These approaches have been extensively refined and remain powerful and reliable for a wide range of optimization problems. However, adapting them for GPU acceleration is challenging due to their dependence on complex matrix operations and factorization routines.

The Primal-Dual Hybrid Gradient (PDHG) method, also known as PDLP [16] when adapted for LPs, offers a first-order alternative designed for large-scale problems. Unlike traditional methods, PDHG relies on gradient-based updates and lightweight operations such as sparse matrix-vector multiplications, which are well-suited for GPU implementation. The algorithm builds on the primal–dual formulation of LPs, iteratively updating both primal and dual variables to solve saddle-point problems. This framework was initially introduced by Chambolle and Pock [17], and subsequent enhancements by Applegate et al. [16] incorporated adaptive step size control and restart mechanisms to accelerate convergence.

A standard LP and its dual can be expressed as:

$$\text{Primal: } \min_x c^\top x \quad \text{s.t. } Ax \leq b, \quad x \geq 0,$$

$$\text{Dual: } \max_y b^\top y \quad \text{s.t. } A^\top y \geq c, \quad y \geq 0.$$

This pair of problems can be equivalently written in a saddle-point form as:

$$\min_x \max_y L(x, y) = c^\top x - y^\top Ax + b^\top y.$$

The PDHG method solves this by performing iterative updates:

$$x_{k+1} = \text{proj}_{\mathbb{R}_+^n} \left( x_k + \tau A^\top y_k - \tau c \right),$$

$$y_{k+1} = y_k - \sigma A(2x_{k+1} - x_k) + \sigma b,$$

where  $\tau$  and  $\sigma$  denote the primal and dual step sizes, respectively. Building on this foundation, Applegate et al. [16] introduced practical modifications—including adaptive step size selection and restart strategies—that enhance the performance of PDHG on large LPs.

## III. RESULTS AND DISCUSSION

### A. Evaluation of Solver Runtime Performance

The computation times for the 4224-, 6049-, and 6717-bus networks were evaluated across multiple scenarios, Divisions 1–3, and solver configurations, comparing GPU-based PDHG against multi-threaded CPU solvers using the barrier method. Figures 2, 3, and 4 show boxplots of the total runtime for each network and division. Overall, the GPU solver provides faster computation, particularly for larger networks and computationally intensive divisions (e.g., long control horizons in Divisions 2 and 3).

In the 4224-bus network, GPU-based PDHG outperforms CPU threads in Divisions D2 and D3. The most significant speedups are observed in the 6049-bus network, where CPU runtimes for certain divisions spike, while PDHG consistently reduces total computation time and exhibits lower variance. For the 6717-bus network, GPU-based PDHG also demonstrates overall superior performance compared to CPU threads.

These improvements are largely attributed to the higher-quality solutions produced by PDHG, which provide a superior warm start for the crossover and branch-and-bound stages, thereby accelerating convergence. While CPU-based barrier solvers benefit from multi-threading, they cannot exploit the parallelism offered by GPU execution. The boxplots also reveal scenario-dependent variability, reflecting differences in convergence behavior between PDHG and the barrier method. Overall, GPU-based PDHG shows the greatest advantage for large-scale and computationally demanding problems, such as Divisions 2 and 3 of the 6049-bus system, providing faster and more consistent runtimes with reduced variability.

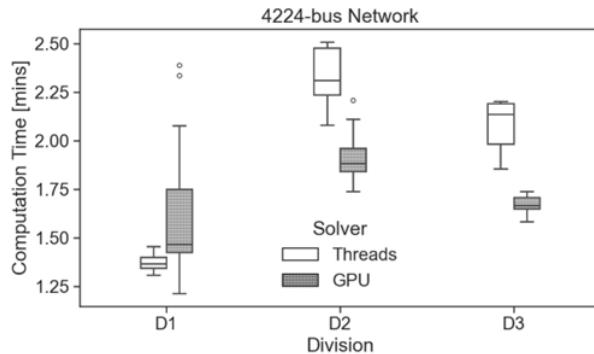


Fig. 2: Computation time distribution for the 4224-bus network (Divisions 1–3). The GPU-based solver achieves faster convergence and lower variance in Divisions 2 and 3.

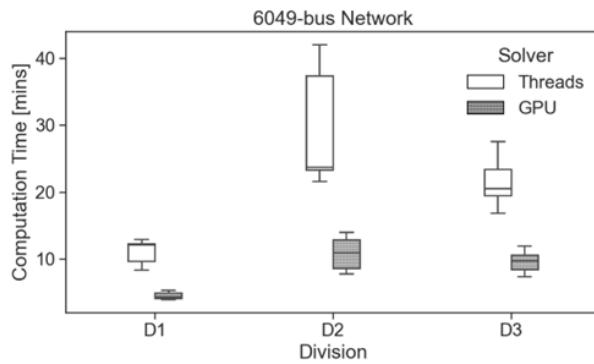


Fig. 3: Computation time distribution for the 6049-bus network (Divisions 1–3). The GPU-based solver achieves faster convergence and lower variance across all Divisions.

### B. PDHG Execution & Crossover Refinement

The PDHG iteration log is similar to that of the barrier method, displaying the current primal and dual objective values, primal and dual residuals, and complementarity. As the iterations progress, the gap between the primal and dual objectives narrows, and both the residuals and complementarity decrease.

The termination criteria used in this study were set to their default values (see Termination for more details in [15]). Once

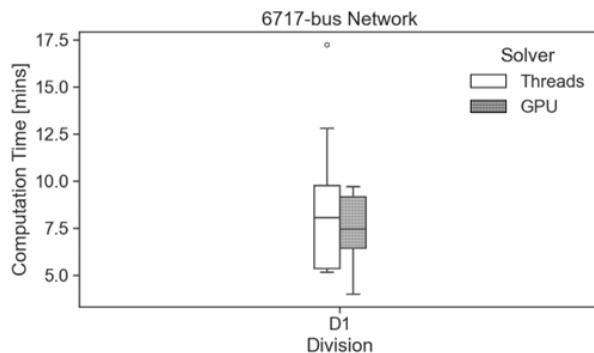


Fig. 4: Computation time distribution for the 6717-bus network (Division 1). The GPU-based solver shows improved convergence.

the criteria are satisfied, the crossover step converts the PDHG solution into a basic feasible solution.

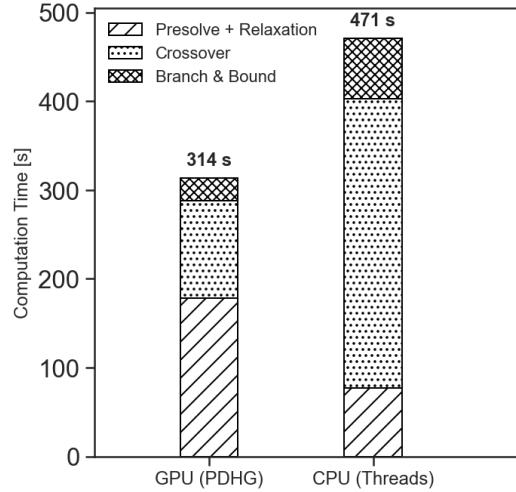
Unlike the Simplex method, which directly produces a basic feasible solution, PDHG and interior algorithms generate dense solutions that may slightly violate feasibility or optimality conditions. Rather than switching to the Simplex algorithm—which tends to be slower and less suitable for GPU acceleration—the crossover phase efficiently refines the output of PDHG or interior algorithms into a clean, sparse basic solution. It should be noted that only the PDHG algorithm itself is executed on the GPU.

It is instructive to experiment with different termination tolerances. For example, increasing the PDHG termination tolerances, such as the relative LP duality gap, reduces the computational effort spent within the PDHG iterations. However, this comes at the cost of solution accuracy, requiring the crossover algorithm to spend more time refining the approximate solution into a high-quality feasible one.

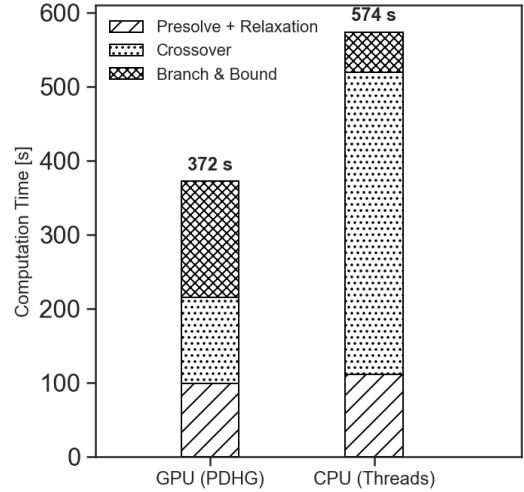
By design, first-order methods such as PDHG converge linearly, meaning they rarely achieve highly accurate solutions at absolute tolerances. Therefore, PDHG is typically terminated once relative constraint violations fall below a practical threshold. To improve accuracy, Gurobi automatically invokes the crossover algorithm, which refines PDHG’s approximate solution into a high-accuracy one. Although crossover and branch-and-bound still run on the CPU, they benefit from the improved solution quality produced by the GPU-based PDHG. Specifically, PDHG provides a better warm start for crossover, a higher-quality initial basis, and tighter LP relaxation bounds for MILP problems. Consequently, even CPU-only stages complete faster, as they begin from a more accurate and well-conditioned solution.

Figure 5 illustrates the computation time breakdown for solving the problem using GPU-based PDHG and multi-threaded CPU solvers for two random scenarios of the 6049-bus system. In both cases, the GPU solver demonstrates a clear advantage in overall runtime, particularly in the crossover stage, which dominates CPU computation time. In the first scenario, the presolve and relaxation phases of the CPU solver—corresponding to the barrier method—are faster on the CPU, but the GPU-based relaxation using PDHG significantly reduces the crossover and branch-and-bound stages, resulting in a lower total runtime. In the second scenario, the GPU is slightly faster in the PDHG relaxation phase, and although the branch-and-bound stage is slower, the substantial reduction in crossover time still enables the GPU-based approach to achieve a lower total computation time. This improvement is attributed to the high-quality solution obtained by PDHG, which provides a better warm start and a more accurate initial basis for the crossover, accelerating this stage.

Specifically, the crossover stage starting from the GPU-based relaxation is more than  $2.96 \times$  faster than when starting from the CPU-based relaxation for these two scenarios. Overall, the GPU-based solver is approximately  $1.5 \times$  faster than the CPU-based solver across both scenarios.



(a) Random Scenario 1



(b) Random Scenario 2

Fig. 5: Comparison of computation times between GPU-based PDHG and multi-threaded CPU solvers for the 6049-bus system across two random scenarios. Each bar represents the total runtime, decomposed into three stages: (1) presolve and relaxation, (2) crossover, and (3) branch-and-bound. For the CPU solver, the relaxation stage corresponds to the barrier method, while for the GPU solver, it corresponds to PDHG. The GPU-based relaxation consistently provides a higher-quality initial solution, reducing overall computation time by accelerating convergence in the crossover and/or branch-and-bound stages.

### C. Speedup Summary

Solver runtimes highlight the computational benefits of utilizing GPUs to solve the relaxed MILP for the unit commitment problem. Tables III and IV summarize the maximum solution times and the speedups achieved with the GPU-based solver. The GPU solver reduces the maximum solution time and overall computation time for the 4224-bus system in divisions 2 and 3 (speedups  $> 1.2\times$ ), for the 6049-bus system across all divisions (speedups  $> 2.27\times$ ), and for the 6717-bus system (speedup =  $1.23\times$ ). Notably, for the 6049-bus system in division 2, the GPU solver decreases the maximum solution time from 42 minutes to 14 minutes.

### D. Solution Quality

The GPU-accelerated solver maintains the quality of the solutions, as summarized in Table V. This table presents the distribution of scores, defined as the ratio between the GPU-computed objective and the thread-based objective. Since the objective corresponds to the maximized market surplus [13], a score greater than 1 indicates an improved solution.

Out of 57 test scenarios, 36 scenarios achieved a score of 1 or higher, while the remaining 21 scenarios had scores slightly below 1, yet still above 0.999. The overall mean score across all test scenarios is approximately 0.9999, demonstrating that the GPU-based solver effectively preserves high-quality solutions.

### E. Mitigating Threads Spin/Wait Overhead

In the CPU-based runs, the solver may execute the primal simplex, dual simplex, and barrier algorithms in parallel using multithreading. However, once one of these algorithms converges, Gurobi continues to wait for the others to finish, which can introduce spin/wait overhead and result in idle CPU

cycles. This behavior was observed in the 6049-bus system, as reported in Table III. To mitigate this effect and more accurately reflect the computational performance of the CPU-based solver, we enabled the setting that terminates the remaining algorithms once a converged solution is reached, thereby reducing the spin/wait behavior. This adjustment improved the Threads-based runtimes and ensured a fair and consistent comparison with the GPU-based solver. The speedup achieved by the GPU over the CPU-based solver after addressing the spin/wait time is shown in Table VI, with the average speedup ranging from  $1.31\times$  to  $1.93\times$ . Overall, the GPU-based solver continues to outperform the CPU-based solver.

The improvement in average computation time when enabling the setting to terminate remaining algorithms once one completes is summarized in Table VII. This indicates that the adjustment leads to more efficient CPU utilization. These results highlight that careful management of multithreaded solver behavior can substantially improve performance and provide a more accurate and fair comparison with GPU-based implementations.

### F. Future Outlook

Given the rapid evolution of GPU hardware and supporting libraries—such as NVIDIA’s cuDSS and cuSparse, which were utilized in this GPU-based implementations—the performance outcomes reported here are expected to improve over time, as the solver stands to benefit directly from advances in parallel sparse linear algebra kernels and memory throughput.

## IV. CONCLUSION

This paper presents a GPU-accelerated approach for solving the unit commitment problem in large-scale power systems. By leveraging the Primal–Dual Hybrid Gradient (PDHG)

TABLE III: Maximum solution times for Threads-based and GPU-based solvers. Arrows indicate whether GPU is faster ( $\downarrow$ , blue) or slower ( $\uparrow$ , red) than Threads. Time in minutes.

Network	Division (D)	Maximum Time [mins]	
		Threads	GPU
4224-bus	D1	1.46	2.39 $\uparrow$
	D2	2.51	2.21 $\downarrow$
	D3	2.20	1.74 $\downarrow$
6049-bus	D1	12.94	5.41 $\downarrow$
	D2	42.03	14.0 $\downarrow$
	D3	27.57	12.0 $\downarrow$
6717-bus	D1	17.24	9.71 $\downarrow$

TABLE IV: Average computation times for Threads-based and GPU-based solvers and resulting speed-up. Colors indicate whether GPU is faster (blue) or slower (red) than Threads. Time in minutes.

Network	Division (D)	Avg. Time [mins]		Avg. Speed-up
		Threads	GPU	
4224-bus	D1	1.37	1.61	0.85x
	D2	2.34	1.92	1.22x
	D3	2.08	1.67	1.25x
6049-bus	D1	11.13	4.56	2.44x
	D2	29.36	10.19	2.88x
	D3	21.85	9.64	2.27x
6717-bus	D1	8.76	7.12	1.23x

algorithm on GPUs, the proposed solver achieves faster convergence in the relaxed linear subproblem, providing high-quality warm starts for crossover and branch-and-bound procedures. Validation on 4224-, 6049-, and 6717-bus networks demonstrates significant reductions in computation time—up to  $2.88\times$  speed-up—while preserving solution quality, with mean scores near 0.9999 relative to CPU-based methods. These results highlight the potential of GPU-based optimization to enhance the efficiency and scalability of large-scale mixed-integer linear programming for power system operations, enabling faster and more reliable unit commitment in modern grids. As GPU architectures and sparse optimization libraries continue to advance, the performance and scalability of the proposed solver are expected to further improve.

#### ACKNOWLEDGMENT

Artificial intelligence tools were used to enhance the clarity and readability of the manuscript. We want to thank the support of US ARPA-E (#DE-AR0001646), and NSF (#2442420 and #2313768).

#### REFERENCES

- [1] G. B. Giannakis, V. Kekatos, N. Gatsis, S. J. Kim, H. Zhu, and B. F. Wollenberg, “Monitoring and optimization for power grids: a signal processing perspective,” *IEEE Signal Process. Mag.*, vol. 30, no. 5, pp. 107–128, 2013, doi: 10.1109/MSP.2013.2245726.
- [2] S. T. Elbert, J. T. Holzer, A. Veeramany, C. DeMarco, H. Mittelmann, and T. Overbye, “Supporting ARPA-E Power Grid Optimization: Final Scientific/Technical Report,” Pacific Northwest National Laboratory (PNNL), Richland, WA, PNNL-36158, 2024. Available: <https://www.osti.gov/servlets/purl/2404530>; last accessed Nov. 1, 2025.
- [3] M. Gao, J. Yu, Z. Yang, and J. Zhao, “A Physics-Guided Graph Convolution Neural Network for Optimal Power Flow,” *IEEE Trans. Power Syst.*, vol. 39, no. 1, pp. 380–390, 2024, doi: 10.1109/TPWRS.2023.3238377.
- [4] J. Mohammadi, G. Hug, and S. Kar, “Agent-based distributed security constrained optimal power flow,” *IEEE Trans. Smart Grid*, vol. 9, no. 2, pp. 1118–1130, 2018, doi: 10.1109/TSG.2016.2577684.
- [5] F. Capitanescu *et al.*, “State-of-the-art, challenges, and future trends in security constrained optimal power flow,” *Electr. Power Syst. Res.*, vol. 81, no. 8, pp. 1731–1741, 2011, doi: 10.1016/j.epsr.2011.04.003.
- [6] J. K. Skolfield and A. R. Escobedo, “Operations research in optimal power flow: A guide to recent and emerging methodologies and applications,” *Eur. J. Oper. Res.*, vol. 300, no. 2, pp. 387–404, 2022, doi: 10.1016/j.ejor.2021.10.003.
- [7] S. Chevalier, “A Parallelized, Adam-Based Solver for Reserve and Security Constrained AC Unit Commitment,” *Electric Power Syst. Res.*, vol. 235, no. 110685, 2024, doi: 10.1016/j.epsr.2024.110685.
- [8] R. Parker and C. Coffrin, “Managing Power Balance and Reserve Feasibility in the AC Unit Commitment Problem,” *Electric Power Syst. Res.*, vol. 234, no. 110670, 2024, doi: 10.1016/j.epsr.2024.110670.
- [9] D. Bienstock and M. Villagra, “Accurate and Warm-Startable Linear Cutting-Plane Relaxations for ACOPF,” *2024 IEEE 63rd Conference on Decision and Control (CDC)*, 2024, doi: 10.1109/CDC56724.2024.10886304.
- [10] J. T. Holzer, S. Elbert, H. Mittelmann, R. O’Neill, and H. Oh, “GO Competition Challenge 3: Problem, Solvers, and Solution Analysis,” *Energy Systems*, 2024, doi: 10.1007/s12667-024-00708-1.
- [11] H. Sharadga, J. Mohammadi, C. Crozier, and K. Baker, “Optimizing Multi-Timestep Security-Constrained Optimal Power Flow for Large Power Grids,” *2024 IEEE Texas Power Energy Conf.*, 2024, doi: 10.1109/TPEC60005.2024.1047229.
- [12] H. Sharadga, J. Mohammadi, C. Crozier and K. Baker, “Scalable Solutions for Security-Constrained Optimal Power Flow with Multiple Time Steps,” *IEEE Transactions on Industry Applications*, vol. 61, no. 3, pp. 4812–4821, 2025, doi: 10.1109/TIA.2025.3532927.
- [13] J. Holzer *et al.*, “Grid optimization challenge 3 problem formulation.” [https://data.openei.org/files/5997/Challenge3\\_Problem\\_Formulation\\_20240122.pdf](https://data.openei.org/files/5997/Challenge3_Problem_Formulation_20240122.pdf) (accessed Nov. 2, 2025).
- [14] ARPA-E, “Data: ARPA-E Grid Optimization (GO) Competition Challenge 3,” <https://catalog.data.gov/dataset/arpa-e-grid-optimization-go-challenge-3>; last accessed Nov. 1, 2025.
- [15] Gurobi Optimization. *Gurobi 13.0 (Beta) Documentation*, 2025. Available online: <https://docs.gurobi.com/13.0/>; last accessed October 31, 2025.
- [16] D. Applegate, M. Díaz, O. Hinder, H. Lu, M. Lubin, B. O’Donoghue, and W. Schudy, “Practical large-scale linear programming using primal-dual hybrid gradient,” *Proc. 35th Conf. Neural Information Processing Systems (NeurIPS 2021)*, 2021.
- [17] Chambolle, A. and Pock, T. “A first-order primal-dual algorithm for convex problems with applications to imaging,” *Journal of Mathematical Imaging and Vision*, vol. 40, pp. 120–145, 2011.

TABLE V: Distribution of scores for GPU objective. Score = Objective (GPU) / Objective (Threads). Out of 57 scenarios, 36 scenarios achieve a score of at least 1, while 21 scenarios have scores slightly below 1 but above 0.999, indicating that the GPU solver maintains solution quality.

Score Range	Number of Scenarios
$0.999 < \text{Score} < 1$	21
$\text{Score} \geq 1$	36
Average Score	$\approx 0.9999$

TABLE VI: Average computation times for Threads-based and GPU-based solvers and resulting speed-up, for the 6049-bus system. The Threads-based solver runtime was improved by addressing CPU spin/wait effects. Colors indicate whether GPU is faster (blue) or slower (red) than Threads.

Division (D)	Avg. Time [mins]		Avg. Speed-up
	Threads	GPU	
D1	5.97	4.56	1.31x
D2	19.7	10.19	1.93x
D3	16.06	9.64	1.67x

TABLE VII: Speedups achieved for the threads-based solver runtime by addressing CPU spin/wait effects for the 6049-bus system.

Division (D)	Avg. Speed-up
D1	1.86x
D2	1.49x
D3	1.36x