GPU-accelerated Gradient Descent Based Load flow analysis

Aashish *2021EEB1144,* Anisha Singh *2021EEB1151,* Divyanshu Watts *2021EEB1168,*
Himanshu Galav *2021EEB1177,* Shreya Maheshwari *2021EEB1212*

***Abstract*—In this study, we introduce a gradient descent approach to load flow analysis, a critical task in power system operation. The gradient descent method offers high parallelizability and the potential for extensive utilization of GPU programming resources. The mathematical framework provided is versatile and applicable across various types of systems. Additionally, we present a "Pseudo-Random" Gradient Descent implementation of load flow analysis, which significantly enhances convergence rates. We propose algorithms to effectively leverage insights from the pseudo-random walk, enabling the implementation of an adaptive gradient descent approach with a convergence rate comparable to the pseudo-random walk while maintaining the swift iteration runtime characteristic of conventional gradient descent methods. Furthermore, we showcase the remarkable effectiveness of GPU parallelization in our approach, mitigating the computational complexity associated with load flow analysis and addressing scalability challenges.**

***Index Terms*—Load Flow, Gradient Descent, GPU, Sparse**

Load flow analysis is crucial for the operation of a power system. The complexity arises from the decoupled nature of the power flow equations[1]. Conventionally, iterative methods such as Newton-Raphson [2] have been utilized to solve these equations. Although Newton-Raphson offers a high convergence rate, its reliance on inverse calculation, which is an O(n^3) process, becomes prohibitively expensive for larger systems, significantly increasing simulation time. While methods like the Gauss-Seidel, DC load flow [3] method exist, they often sacrifice convergence rate or solution accuracy.

Another challenge with inverse calculation in Newton-Raphson is its sequential nature. In recent decades, with the rise of GPU and TPU-based parallel processing driven by machine learning and cryptocurrency, there is a missed opportunity as these processing units are highly effective in matrix multiplication but not in inverse calculation.

In this work, we introduce the gradient descent approach to load flow analysis. The gradient descent approach is highly parallelizable and has the potential to leverage GPU programming to its fullest extent. In the methodology section, we provide the mathematical framework of the approach. In the subsequent section, we explore gradient descent for different power systems. To enhance convergence rate, we utilize the pseudo-random walk, significantly expediting convergence. We propose algorithms to effectively utilize observations from the pseudo-random walk to implement an adaptive gradient descent with a convergence rate comparable to the pseudo-random walk but with each iteration's runtime as fast as conventional gradient descent. In the next section, we demonstrate the incredible effectiveness of GPU parallelization in our approach. Finally, we discuss potential future prospects.

I. Methodology

In this section, we'll offer a comprehensive explanation of gradient descent for load flow analysis. To effectively implement any gradient descent method, it's crucial to define a scalar cost or error function. We can achieve this by utilizing the correction matrix to define the error function:

$$Er(V) = -\vec{\Delta S}^{\dagger} \cdot \vec{\Delta S} \tag{1}$$

Here, $\Delta S = S - S_{\text{ref}}$, where S represents the apparent power corresponding to voltage V, and S_{ref} is the scheduled apparent power. The negative sign is incorporated because we aim to minimize the function.For implementation purposes, we'll be working in rectangular coordinates. The gradient is then given as:

$$\nabla_j Er = 2Re(\vec{\Delta S} \cdot \frac{\partial \vec{S}^{\dagger}}{\partial V_j}) \tag{2}$$

$$\frac{\partial \vec{S}_i^*}{\partial V_j^{re}} = V_i^* Yij + \sum_k Y_{ik} V_k \delta_{ij} \tag{3a}$$

$$\frac{\partial \vec{S}_i^*}{\partial V_j^{img}} = 1j(V_i^* Yij - \sum_k Y_{ik} V_k \delta_{ij}) \tag{3b}$$

Here, V^{re} and V^{img} represent the real and imaginary parts of the voltage, respectively. $[Y]$ is the Y-matrix corresponding to the bus, and δ denotes Dirac's delta function used to construct the diagonal matrix. Based on the gradient evaluated from these equations, the voltage at the next iteration is given by:

$$\vec{V}_{n+1} = \vec{V}_n + \gamma \vec{\nabla} Er \tag{4}$$

The step size, denoted by γ, determines the rate of change in each iteration.

Visually, one can grasp gradient descent using a contour map where each contour denotes a constant error (Fig. 1). The gradient at any point is a vector perpendicular to the constant error contour, representing the direction of maximum change in the error function. Thus, we adjust the voltage in the direction of the gradient (4). To achieve the fastest

convergence, optimizing the step size is essential. Therefore, we search for the value of γ until the gradient becomes tangential to another constant error contours, signifying the optimal step size. Subsequently, we evaluate the gradient at the next point. Determining this optimal γ value can be done analytically or numerically, but both methods are resource-intensive. Hence, in section IV, we introduce the concept of a pseudo-random walk to efficiently calculate the optimal γ value.

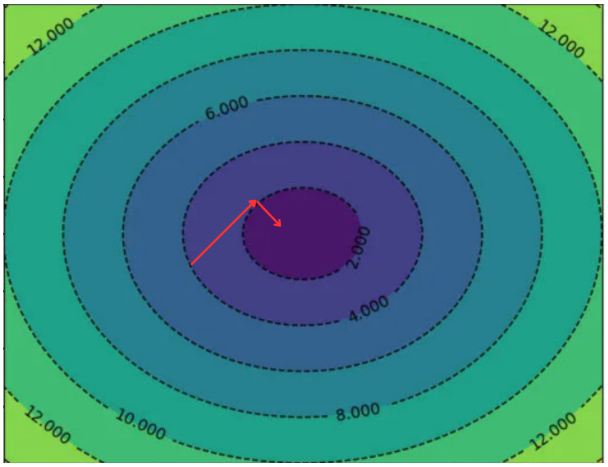

Fig. 1. 2D constant error contour representation of Gradient Descent

It's worth noting that consecutive gradients are observed to be orthogonal to each other. Leveraging this observation effectively enables our system to adapt and converge rapidly in scenarios where gradients don't change rapidly. However, this method is equally effective in handling peak loads or systems with irregular behavior. We adapt the bottom up approach in following section to explain our implementation and though process.

II. Modelling

To evaluate the admittance matrix (Y matrix), we've structured a network of buses. Each bus in this network connects to a smaller subset of buses than the total system, and we've randomly assigned impedance, hence admittance to each line. This strategy provides us with flexibility in modifying the bus size and influencing system behavior. To create a test bench, we've allocated voltage to each bus, which subsequently yields power flow at each bus through power flow equations.

III. Constant Step Gradient Descent

In this section we will analyse the impact of step size on convergence rate and residue error in case of different system size for constant step gradient descent. We begin our analysis by keeping the number of buses in the system fixed while varying the step size. A notable trend emerges: increasing the step size increases the system convergence. However, this acceleration comes with a trade-off, a large residue error is left behind (Fig. 2). This phenomenon occurs because a larger step size makes the algorithm more prone to overshooting the minimum point of the optimized function. Consequently,

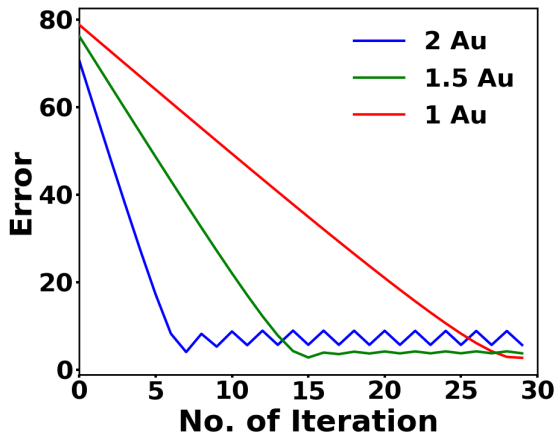

Fig. 2. Rate of convergence for fixed bus size and varying step size in Arb. Units(AU)

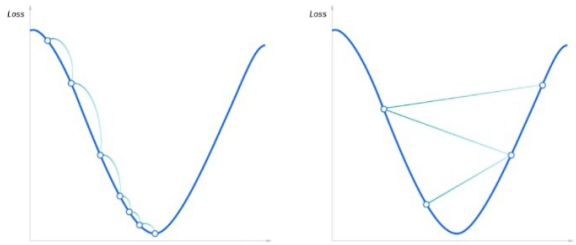

Fig. 3. 1D representation of Gradient decent for smaller step size(Left) and Larger step size(Right)

it's frequently observed that the algorithm jumps over the minima, resulting in a significant residue error. Conversely, reducing the step size decreases the residue error, but the rate of convergence slows considerably (Fig 3). This is because a smaller step size prompts smaller, more cautious steps toward the minimum point, thereby reducing the risk of overshooting. However, this cautious approach also necessitates more iterations to converge.

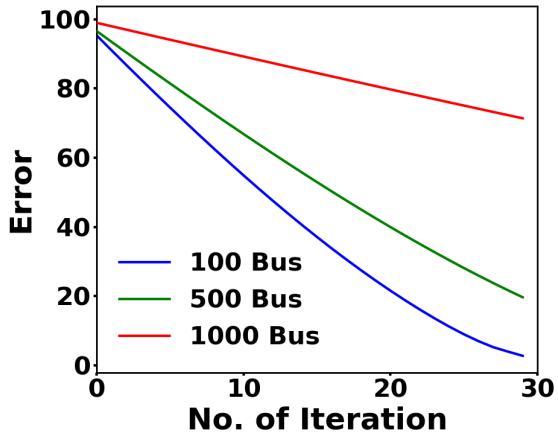

Fig. 4. Error versus Number of Iterations for different Systems at Fixed Step Size

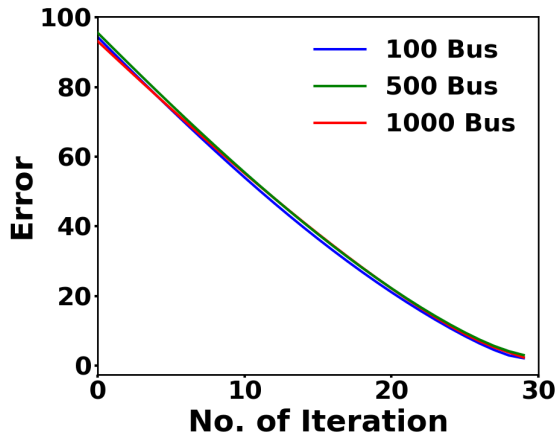

Fig. 5. Error Versus Number of Iterations for Different Systems in Optimal Case

When maintaining a fixed step size optimal for a particular bus size and varying the number of buses in the system, another intriguing observation surfaces. The step size optimal for one bus count may not necessarily be optimal for a different bus count in the system (Fig. 4).

The choice of step size plays a crucial role in balancing convergence rate and residue error, particularly regarding the number of buses in the system. To optimize convergence, an optimal step size must be determined for each specific number of buses. It is noted that selecting the most optimal step size for a particular number of buses ensures similar behaviors in convergence rate and remaining residue error across all cases (Fig. 5). Regardless of fluctuations in the number of buses, employing this optimal step size guarantees a balanced convergence process without compromising accuracy.

IV. Pseudo-Random Walk Gradient Descent

The traditional gradient descent methods are known for their power in optimizing differential function, yet they can be limited by the choice of step sizes and the challenges posed by high-dimensional and non-convex landscapes.The Pseudo Random Walk Gradient Descent approach (Fig 6) combines the strengths of gradient descent with an pseudo randomized approach to step selection. By incorporating gradient information and error functions through random perturbations, this algorithm provides an alternative for optimization problems. At each iteration x_k the gradient matrix $J(x_k)$ of the objective function is calculated. Random increments are introduced to the gradient within the estimated variation range, by choosing different step sizes randomly. The error functions are evaluated to identify potential directions of movement. The optimal step size that minimizes the error function is selected. If the error falls below a predefined threshold ϵ, the optimization process is terminated. Otherwise, the iteration continues based on the computed gradient and step size.

V. Adaptive Step Gradient Descent

In the pseudo-random approach, evaluating the error at multiple points can significantly increase each iteration runtime.

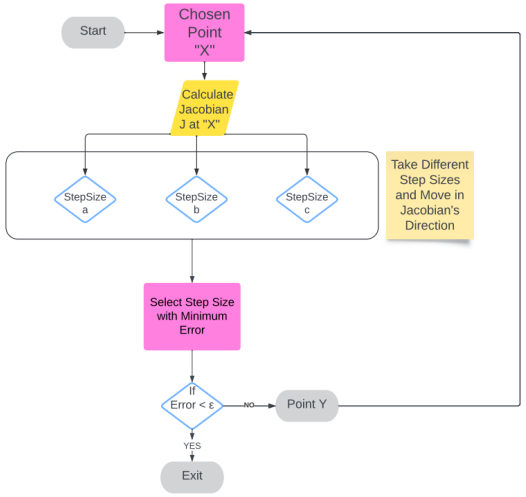

Fig. 6. Pseudo Random Walk Based Gradient Descent Algorithm Flow

To address this issue, we introduce the adaptive step gradient descent approach (Fig. 7) in this section. The step size is a function of our system, represented by the Y matrix, and the power/voltage in each iteration, knowledge of which can be represented by the error function. Thus, γ becomes a function of the Y matrix and error. Once a Y matrix is fixed, we just need to find the relation between the step size and error. Initially, we employ a random walk algorithm on the given system to gather data on the relationship between step size and error. Subsequently, this collected data is utilized to fit an analytical function that approximates the step size as a function of error (and Y bus). This function serves as the adaptive step size function in the algorithm. Once this relation is derived, it becomes uniquely determined for that specific Y bus.

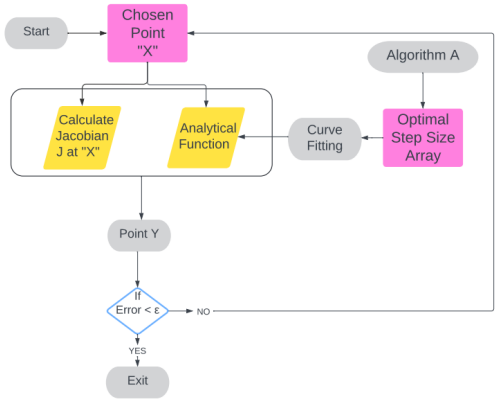

Fig. 7. Adaptive Size Gradient Descent Algorithm FLow

Then for any further simulation, we can utilize this data. During each iteration, the gradient matrix based on the current system state (voltages, power injections) is computed. Using the adaptive step size function (determined dynamically from the analytical function derived from empirical data), the system state is updated towards minimizing the error. We evaluate

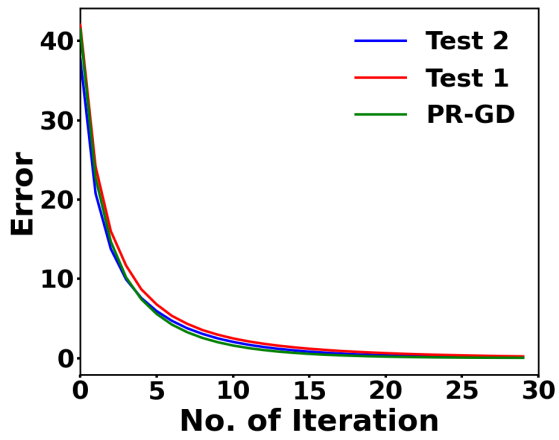

Fig. 8. The convergence of Adaptive Gradient Descent(Test-1,2) against the Pseudo random walk Gradient Descent(RP-GD)

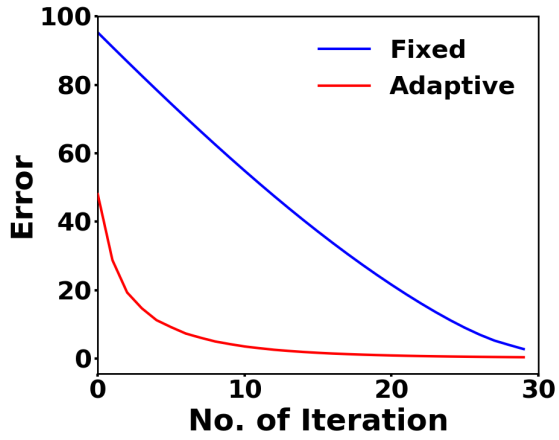

Fig. 9. Error versus Number of Iterations for 100 Bus System.

the error, and if it falls below a predefined threshold ϵ, we terminate the optimization process. Otherwise, we continue iterating using the adaptive step size.

As we can see in Fig 8, the adaptive gradient descent have convergence rate comparable to pseudo random walk approach.

VI. Second-Order Correction

The gradient represents the maximum rate of change locally and may not necessarily point towards the global minimum, potentially leading to suboptimal paths. One approach to tackle this issue is by introducing second-order corrections, such as the Hessian matrix, to incorporate the curvature of the error function. However, such methods often come with high time complexity (O(n^3) or higher), negating the advantages of gradient descent. Alternatively, methods like Nesterov's accelerated gradient descent algorithms attempt to dampen the spiral nature of the error, but it has been observed to be ineffective in our specific case.

Building on insights from the preceding section, we propose leveraging pseudo-random walks to integrate second-order correction effects into the gradient, enhancing convergence speed. The convexity of the error function near the minima ($S = S_{\text{ref}}$) is influenced by the Y matrix of the bus and S_{ref}. Utilizing the error function, we can effectively represent the information from S_{ref}. Hence, it should be feasible to analytically continue the second-order correction matrix near the vicinity of the minima via the Y bus matrix and error function. Although pseudo-random walks may increase simulation runtime, similar to the above section, we can utilize a pseudo-random walk once to determine the second-order correction to the gradient as a function of the fixed Y matrix and error function (Fig. 9).

VII. GPU-based Parallelisation

In this section, We transitioned to leveraging the power of GPU parallelization using the CuPy [4] library. By distributing diverse tasks across GPU cores, we harness the parallel processing capabilities of the GPU, significantly reducing execution time 10. While we primarily utilize first-order GPU parallelization with built-in functions, additional enhancements can be achieved through advanced manual GPU parallelization using libraries like Numba CUDA and CuSparse [5].

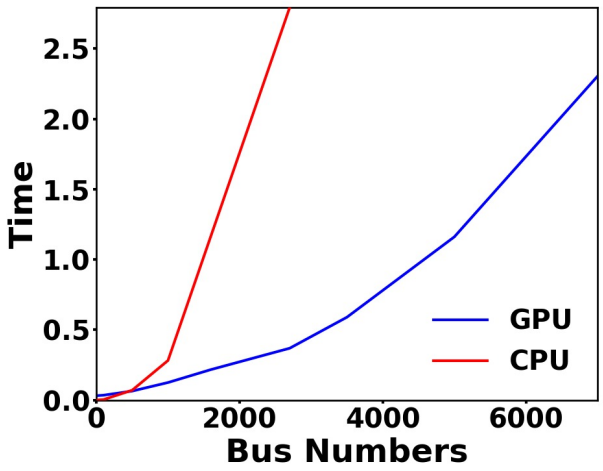

Fig. 10. Run time for 30 iteration for GPU and CPU programming

This shift to GPU-based processing has allowed us to capitalize on the potency of parallel computation, particularly beneficial for tasks involving matrix and array multiplication. Moreover, we've managed to mitigate the complexity of standardized algorithms, often of the order of n squared, through optimization techniques.

With this methodology, our GPU-based algorithm consistently surpasses standardized CPU algorithms, delivering expedited results and offering a notable advantage in computational efficiency. This enhanced performance is especially advantageous for computationally intensive tasks, enabling faster simulations and analyses.

VIII. Future Prospects

Looking into the future prospects of this research, there are several avenues for improvement and optimization like

Sparse Data structure-Addressing the issue of computing overhead due to the Sparsity of connected buses in the system holds promise. Leveraging the power of sparse data structures provided by CUDA in libraries like cuSparse can significantly reduce computational demands. By transitioning from computations dependent on the total number of buses to focusing solely on sparsely connected buses, unnecessary computing overhead can be mitigated, leading to more efficient simulations.

Customized GPU Programming- There is a need to optimize the codebase further. While current simulations rely on predefined functions and libraries like CuPy provided by CUDA, which may not be fully optimized for the specific requirements of the research, future efforts will involve developing custom functions and utilizing libraries tailored specifically for tasks such as gradient and error calculation in the context of power systems. This approach aims to streamline computations, enhance performance, and improve the overall efficiency of the research framework.

TABLE I
STEP SIZE AND RESIDUE ERROR

Step Size	Residue Error%
1	2.80
1.25	3.74
1.5	5.10
1.75	6.77
2	12.57
3	18.44

TABLE II
RATE OF CONVERGENCE FOR DIFFERENT ALGORITHMS

	Convergence Rate
Fixed	36
PR-GD	10
Test 1	11
Test 2	11

IX. CONCLUSION

In this paper, novel methodologies for load flow analysis in power systems, focusing on enhancing convergence rates and reducing simulation times, have been proposed. The utilization of the gradient descent approach has proven effective in load flow analysis, particularly when coupled with pseudo-random walk techniques and adaptive step sizing. Our observations indicate that this combined approach significantly accelerates convergence rates while maintaining solution accuracy. It underscores the critical importance of selecting step sizes tailored to specific system configurations, emphasizing the trade-off between convergence speed and residue error. Furthermore, the use of GPU-based parallelization in this algorithm represents a pivotal advancement, demonstrating substantial improvements in computational efficiency and speed. By leveraging the parallel processing power of GPUs, we have achieved remarkable reductions in simulation time, thereby enhancing the practicality and scalability of load flow analysis, especially for larger power systems. Looking ahead, future advancements in load flow analysis include implementing second-order correction techniques to improve convergence efficiency, leveraging sparse data structures with CUDA libraries like cuSparse to reduce computational overhead, and customizing GPU programming for optimized performance in power system analysis. These developments aim to refine methodologies, streamline computations, and contribute to more scalable and efficient solutions.

X. ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude to Professor Ranjana Sodhi for her invaluable guidance, mentorship, and unwavering support throughout our journey in the Power Systems course. Her expertise and encouragement have been instrumental in shaping our understanding and propelling us towards innovation. We extend our sincere appreciation to Mr. Manish, a dedicated Ph.D. scholar, whose expertise and assistance have been indispensable in our endeavor to propose a novel load flow method. Their insights, feedback, and encouragement have been invaluable in the development and presentation of our research. We are deeply thankful for their time, dedication, and contributions to our academic growth and success.

REFERENCES

[1] Ashirwad Dubey. Load flow analysis of power systems. *International Journal of Scientific & Engineering Research*, 7(5), 2016.

[2] M. Karimi, A. Shahriari, M.R. Aghamohammadi, H. Marzooghi, and V. Terzija. Application of newton-based load flow methods for determining steady-state condition of well and ill-conditioned power systems: A review. *International Journal of Electrical Power Energy Systems*, 113:298–309, 2019.

[3] Deepinder Kaur Mander and Supreet Kaur Saini. Load flow analysis: a review. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, 5(3):1254–1260, 2016.

[4] Ryosuke Okuta, Yuya Unno, Daisuke Nishino, Shohei Hido, and Crissman Loomis. Cupy: A numpy-compatible library for nvidia gpu calculations. In *Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Thirty-first Annual Conference on Neural Information Processing Systems (NIPS)*, 2017.

[5] Qihan Wang, Mingliang Li, Jianming Pang, and Di Zhu. Research on performance optimization for sparse matrix-vector multiplication in multi/many-core architecture. pages 350–362. IEEE, 12 2020.