energies

MDPI

Review

Parallel Power Flow Computation Trends and Applications: A Review Focusing on GPU

Dong-Hee Yoon [1] and Youngsun Han [2,*]

[1] Department of Railway, Kyungil University, Gyeongsan 38428, Korea; dhyoon@kiu.kr
[2] Department of Computer Engineering, Pukyong National University, Pusan 48513, Korea
* Correspondence: youngsun@pknu.ac.kr; Tel.: +82-51-629-6250

Received: 30 March 2020; Accepted: 21 April 2020; Published: 1 May 2020

Abstract: A power flow study aims to analyze a power system by obtaining the voltage and phase angle of buses inside the power system. Power flow computation basically uses a numerical method to solve a nonlinear system, which takes a certain amount of time because it may take many iterations to find the final solution. In addition, as the size and complexity of power systems increase, further computational power is required for power system study. Therefore, there have been many attempts to conduct power flow computation with large amounts of data using parallel computing to reduce the computation time. Furthermore, with recent system developments, attempts have been made to increase the speed of parallel computing using graphics processing units (GPU). In this review paper, we summarize issues related to parallel processing in power flow studies and analyze research into the performance of fast power flow computations using parallel computing methods with GPU.

Keywords: power flow computation; high performance computing (HPC); parallelism; parallel computation; LU decomposition

1. Introduction

Power flow (PF) analysis is popularly employed to analyze electrical power systems by estimating the voltage and phase angle of buses inside the power system. Power flow computation is designed to repeatedly apply several numerical methods for solving nonlinear equations of such electric power systems; the calculations thus require a considerable amount of computational execution time. Furthermore, due to a general increase in the demand for electric power, the introduction of emerging renewable energy sources, and the spread of electric vehicles (EVs), modern power systems are continually expanding and display rapidly increasing complexity. The adoption of smart grids and advanced electronic devices creates a need for more electrical power of higher quality, and the spread of EVs is expected to lead to increased power consumption in the near future. Hence, recently, there has been greater demand for more accurate power flow analysis. In addition, because power flow analysis involves a large amount of contingency analysis for all probable accident conditions in power systems, the computational resources and time required for such analyses have increased dramatically.

Numerous studies have been undertaken to significantly reduce the computation time of complex numerical analysis by applying parallel and distributed processing with massive computing resources. In particular, the use of high-performance computing (HPC) machines to reduce computational times in power flow computation has long been studied [1–5]. After D. M. Falcao et al. [1] and V. Ramesh et al. [2] introduced the application of HPC to power system problems, several studies have been conducted in earnest into the use of HPC for accelerating power system analysis. R. Baldick et al. [3] and F. Li et al. [4] proposed rapid power flow computation using distributed computing, which uses multiple computers connected via an Ethernet network. Currently, because HPC technologies such as multi-core processors, clusters, and GPUs have made significant strides, many more studies have

been undertaken into accelerating power system analysis by using parallel and cloud computing and adopting HPC technologies [5].

Attempts to reduce the analysis time of power systems by using GPU-based parallel computing have recently increased. Of course, the use of GPU is not confined to power flow studies. One example is the visualization part of power systems. Some studies into the visualization of power systems have been undertaken [6]. Real-time visualization of power systems is necessary for the simulation, state estimation, and prediction of power systems, and many studies have been performed investigating the speedup using HPC [7]. The parallel processing is also valuable in the electric railway system because it requires a lot of computation and data processing in various fields such as scheduling optimizations, signal processing, finite element analysis, train dynamics, and power network simulation [8]. A variety of parallel processing techniques can be applied to the electric railways, and some studies using GPU have been conducted in this area [9–11]

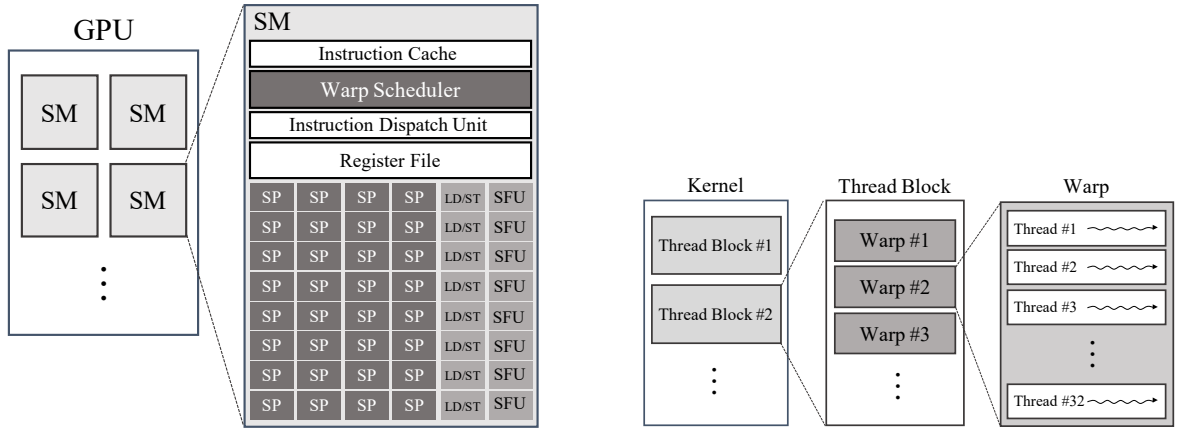
The rest of the paper is organized as follows: In Section 2, we describe the background of our review, such as GPU, power flow computation, and matrix handling methods, briefly. In Section 3, we present GPU application trends in power flow computation. For the sake of understanding, we introduce the previous PF studies with parallel processing and describe the recent trend of GPU-based power flow studies. In Section 4, we provide a detailed description of some representative studies that apply GPU-based parallel computing to power flow analysis and summarize their performance using a comparative analysis table. Finally, the conclusion is made in Section 5.

2. Background

2.1. GPU

GPUs have been widely adopted to accelerate graphical processing algorithms for image processing and computer graphics that handle large amounts of data using their highly parallel architecture [12–16]. In particular, modern GPUs have been developed in the form of general-purpose computing on graphics processing units (GPGPUs) as a substitute for CPUs in HPC for various scientific algorithms, such as deep learning, genome mapping, and power flow analysis. This is because GPUs significantly outperform CPUs in terms of the cost-effectiveness of floating-point computational throughput.

Figure 1 briefly outlines the overall hardware architecture of a GPU and its programming model. The GPU is specialized to dramatically accelerate computation-intensive tasks due to its massive parallel architecture, which employs a large number of streaming multiprocessors (SMs) consisting of 32 scalar processors (SPs) that operate in a lockstep manner. The GPU supports parallel execution of a computational kernel consisting of multiple thread blocks, and each thread block is divided into warps, i.e., groups of 32 threads. All threads in each warp are executed simultaneously on a single SM. A warp scheduler selects one eligible warp at a time and concurrently executes all threads of the warp on the processing cores, i.e., the SPs, of the SM using lockstep synchronization. Therefore, we can exploit extremely high thread-level parallelism (TLP) by interleaving a large number of threads to the processing cores. Programmers can implement parallel programming models on GPUs using open computing language (OpenCL) and compute unified device architecture (CUDA).



(**a**) Hardware Architecture (**b**) Programming Model

Figure 1. GPU hardware architecture and programming model.

2.2. Numerical Analysis of Power Systems

2.2.1. Power Flow Analysis

A power flow analysis is a representative numerical method that estimates the electric power flow of an interconnected power system. It is designed to repeatedly perform power flow computations to obtain the voltage magnitudes and angles of all the buses in a power system and calculate the real and reactive powers of peripheral equipment connected to the buses. Because the power flow computation mainly consists of nonlinear algebraic equations, various approaches, including Gauss–Seidel (GS), Newton–Raphson (NR) [17], and fast-decoupled power flow (FDPF) [18] methods, have been developed to accelerate the solution of the equations.

The GS method is often used because it is the first practical approach proposed for estimating power flow in large-scale power systems. Because an admittance matrix of a power system contains many zero coefficients, power flow analysis is faster using the GS method than using the NR method for a single iteration. However, the GS method cannot easily be widely adopted in power flow analysis due to its poor convergence characteristics. Therefore, though the GS method is popular in power flow analysis, it is mentioned here only briefly because only a few components can benefit from parallel processing with GPUs.

The NR method is currently the most prevalent for power flow analysis. A number of studies have discussed acceleration of the NR method by applying GPU-based parallel processing techniques. In general, we can write a nodal equation of a power system network as follows:

$$I = Y_{bus} \cdot V \tag{1}$$

where Y_{bus}, I, and V denote an admittance matrix of the network, a current vector, and a voltage vector, respectively. I_i indicates the total power flow at the i-th bus as in Equation (2).

$$I_i = \sum_{k=1}^{n} Y_{ik} V_k \tag{2}$$

A complex power equation can be expressed separately with real and imaginary parts, P and Q, as follows:

$$S_i = V_i (\sum_{k=1}^{n} Y_{ik} V_k)^* = V_i \sum_{k=1}^{n} Y_{ik}^* V_k^* = P_i + jQ_i \tag{3}$$

$$P_i = \sum_{k=1}^{n} |V_i| \cdot |V_k| \cdot (G_{ik} \cos\theta_{ik} + B_{ik} \sin\theta_{ik}) \tag{4}$$

$$Q_i = \sum_{k=1}^{n} |V_i| \cdot |V_k| \cdot (G_{ik}\sin\theta_{ik} - B_{ik}\cos\theta_{ik}) \tag{5}$$

Equations (4) and (5) show the power balance equations of the i-th bus in polar form. The NR method employs an iterative technique to solve the two nonlinear power balance equations. The set of resulting linear equations can be formulated in a matrix form as follows:

$$\Delta f = J \cdot \Delta x \tag{6}$$

where J is a Jacobian coefficient matrix. Equation (7) shows the complete formulation to obtain the power mismatch, i.e., ΔP and ΔQ, through Equations (4) and (5) using the NR method.

$$\begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} = \underbrace{\begin{bmatrix} \frac{\partial P}{\partial \theta} & \frac{\partial P}{\partial V} \\ \frac{\partial Q}{\partial \theta} & \frac{\partial Q}{\partial V} \end{bmatrix}}_{\text{Jacobian}} \begin{bmatrix} \Delta \theta \\ \Delta V \end{bmatrix} \tag{7}$$

For the sake of clarity, Figure 2 provides a succinct illustration of the iterative process of calculating a power flow solution using the NR method. The method starts with arbitrary initial values of V and θ, and then calculates the power mismatch using Equation (7). Also, the method determines whether the mismatch is converged to complete the iteration. If it is not converged, the method computes a Jacobian matrix of the power flow system with updated P and Q first, and derives the mismatches of voltage and phase angles, i.e., ΔV and $\Delta\theta$, from Equation (7). Then, the method updates the voltage and phase angles with the mismatches and performs the overall procedures iteratively.

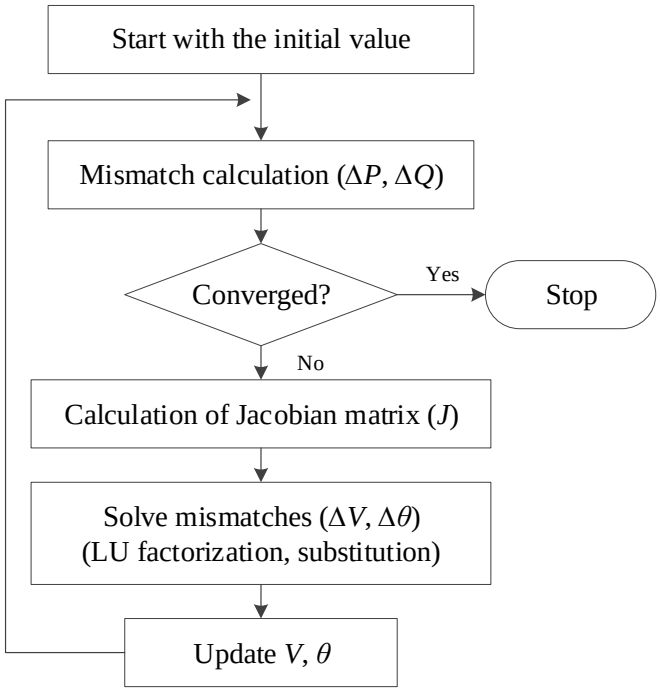


Figure 2. Brief algorithm of the Newton–Raphson (NR) method.

As previously mentioned, we need to carry out heavy matrix calculations, such as an LU decomposition for updating both the Jacobian coefficients and the mismatch of voltage and phase angles. Nevertheless, the NR method performs the power flow computation much faster than the GS method by converging to the solution with fewer iterations; thus, the method is more suitable for large-scale power systems [19]. Also, the Jacobian matrix produced by the NR method can provide

an index for sensitivity analysis or some other control problems. Hence, in this paper, we discuss several previous studies focused on accelerating the NR method using GPU-based parallel computing in detail.

Finally, the FDPF is a method of approximating a solution based on the NR method, which required significantly less computational effort [17]. Although the NR method is the most popular method for PF studies, calculation is slow because it requires calculation of many inversions of the Jacobian matrix for each iteration. To resolve this drawback, some methods, including FDPF, have been proposed to apply approximation approaches instead of using the Jacobian coefficient matrix in its full form. In addition, the FDPF method exploits P-Q decoupling, which means the active and reactive powers are not significantly affected by the magnitude and phase angle change of the bus voltage, respectively. The method does not update its Jacobian coefficient matrix during the convergence iterations for reducing the overhead of numerical computations, so the Jacobian matrix is inverted only once during calculation of the solution. As a result, the FDPF method can formulate the NR method more simply; however, this comes with the disadvantage that more iterations are required to reach the solution and the method more often fails to eventually converge. Hence, the method is usually adopted in specific situations where fast calculation is compulsory for the power flow calculation.

2.2.2. LU Decomposition

For computing the power flow calculation by utilizing either the NR or FDPF method, the solution of a linear system of matrix equations is required, as in Equation (1). Notably, the Y_{bus} matrix reflects the admittance information of a power system, so it presents a sparse quality in containing many zeros. As shown in Figure 2, we need to update V and θ with ΔV and $\Delta\theta$ obtained from the power mismatch, i.e., ΔP and ΔQ, using Equation (7). Hence, we apply a mathematical technique such as sparsity oriented LU decomposition to calculate the inverse of the Jacobian coefficient matrix based on the admittance information [17].

$$A \cdot X = (L \cdot U) \cdot X = L \cdot (U \cdot X) = b \tag{8}$$

$$L \cdot y = b \tag{9}$$

$$y = U \cdot X \tag{10}$$

LU decomposition is also known as triangular factorization or LU factorization. To obtain X by solving a linear equation, i.e., $A \cdot X = b$, the LU decomposition first divides the coefficient matrix A into two lower and upper triangular matrices, L and U, as in Equation (8). Then, y is calculated by performing a forward substitution in Equation (9), and X is finally obtained by a backward substitution in Equation (10).

Numerical computation techniques based on such sparsity are popularly employed in procedures to solve the network equations of large power systems [20]. The time required for the matrix calculations requires a high proportion of the total computation time; for instance, it was found to take up to about 80% of the total computation time in [21]. Hence, many related studies have already been performed [22–26], as we can significantly reduce the total execution time of PF analysis by applying parallel processing to the matrix calculations.

2.2.3. QR Decomposition

LU decomposition is more commonly used to solve power flow problems, but it is also possible to adopt QR decomposition as an alternative for leveraging GPU-based parallel processing. This is because LU decomposition without pivoting may suffer from significant numerical instability, as described in [27]. Although QR decomposition has greater computational complexity compared to

LU decomposition, some studies have found that it is more suitable for GPU-based parallel computing due to its high numerical stability even without pivoting [28].

In linear algebra, a QR decomposition of a complex $m \times n$ matrix A is represented as follows:

$$A = Q \cdot R \tag{11}$$

where Q and R indicate an $m \times m$ unitary matrix and an $m \times n$ upper triangular matrix, respectively. Several methods for implementing the QR decomposition are available, such as the Gram–Schmidt process, Householder transformations, and Givens rotations [29].

2.2.4. Conjugate Gradient Method

The conjugate gradient (CG) method is designed to find numerical solutions of linear system equations in which the matrix is symmetric and positive definite. The method is typically implemented as an iterative algorithm for analyzing large sparse power systems that are too large to be handled by direct methods, such as Cholesky decomposition.

The CG method can be employed instead of LU decomposition in a PF study. The use of the CG method in PF studies has been investigated extensively. In [30], a PF study based on the CG method was introduced. In [31], a PF study using the CG method was performed based on GPUs. In [32], a GPU-based PF study was conducted using a combination of preconditioners and a CG solver. In general, preconditioners are widely used for matrix handling. Since it is difficult to secure parallelism in a general preconditioner, a Chebyshev preconditioner is used to secure the parallelism of the preconditioner in [32].

3. GPU Application Trend in Power Flow Computation

3.1. Conventional Parallel Processing in PF Studies

Recent studies applying GPUs to PF studies have usually been conducted based on existing parallel processing research. Therefore, in terms of PF studies, a review of current parallel processing research is necessary to understand the application of GPUs. As mentioned in Section 2.2.1, PF computation using the NR method performs matrix handling during computation. In this process, it is usually necessary to solve a sparse linear system for power system analysis such as static security and transient stability analysis. Most of the power system applications have conventionally employed direct LU decomposition methods to solve such linear equations. Since this decomposition process takes large amounts of time, numerous studies have examined how to perform this operation more quickly. Considerable computational resources are required for fast computation, and parallel computing is an excellent way to secure such resources. Hence, many attempts to conduct large amounts of power flow computation quickly have been made using parallel computing. Parallel computing requires resources for computation, and some attempts have been made to perform power flow computation by connecting multiple computational resources, such as computers with the goal of fast PF computation. Theoretical attempts at parallel processing in PF studies are described in [33,34]. In the 1990s, researches into the parallel processing of PF computation began in earnest. The fundamental concepts of applying the parallel processing to PF computations are precisely presented in [22,26], and a number of further studies have been performed since; most of the studies focused on matrix handling to divide computation for each computational resource [23–25,35,36]. G. A. Ezhilarasi et al. [37] presented an effective method to perform a parallel contingency analysis using high-performance computing. Also, Z. Huang [38] proposed a dynamic computational load balancing method that utilizes multiple machines of high-performance computing for delivering a large number of contingency computations.

It is also essential to employ a massively large number of cores for accelerating the computation. There have been many endeavors to enhance the performance of parallel computing by adopting multiple processing machines in PF computation. In [39], a commercial solver was utilized in a massively parallel computing environment to carry out the contingency analysis of a power system.

To decrease the calculation time significantly, 4127 of the contingencies were spread to 4127 cores and computed simultaneously. I. Konstantelos et al. [40] presented a computational platform, as a part of the iTesla project, for evaluating the dynamic security of an extensive power system using a commercial analysis software on 10,000 cores of Intel Xeon E5-2680. This project utilizes a large number of CPUs, i.e., 80,640 of cores, which are mutually interconnected, but the maximum number of CPU cores used simultaneously is up to 10,000.

3.2. Advantages of GPU-Based PF Computation

Analysis of a power system requires a large number of PF computations, and one method for reducing the computational time is parallel processing. Although the purpose is to calculate quickly by using many computational resources, this approach has obvious disadvantages in terms of time and cost in establishing a parallel processing environment by connection of a large number of machines. Therefore, the use of GPUs that can process many computations at once is an attractive proposal for PF computation, as it overcomes the shortcomings related to physical connections between machines [41].

The LU decomposition mentioned in Section 2.2.2 is a very time-consuming task in PF studies; thus, it is a good candidate for the application of parallel processing with GPUs. LU decomposition with GPUs has been studied outside the field of PF studies, as it is used in various fields that require a large amount of computation. For example, a parallel process for solving the sparse matrix for circuit simulation has been studied in [42–44], and L. Ren et al. [45] presented results that could overcome the difficulty of super-node formation for extremely sparse matrix objects.

Many studies have recently been conducted to reduce the computation time required for LU decomposition through the use of GPUs. Basically, while matrix handling related operations have been physically assigned to multiple machines in the past, recent studies have focused on how to assign the operations to GPUs for efficient calculation. Compared to using physically separated computational sources, the use of a GPU inside the same machine as a computational resource can yield very high-speed delivery of information, which has a significant advantage in terms of reducing computational time compared to past parallel processing studies. The following chapters summarize the trend of GPU-applied PF studies.

3.3. GPU Applications Trends in PF Studies

GPU-based acceleration techniques are highly productive methods to enhance the performance of power flow computations significantly. We can dramatically diminish the computational time of a sparse linear system (SLS) solution by utilizing GPUs. Researches broadly range from investigating conventional direct LU decomposition to using computational solvers other than LU. In [31], the basics of GPU-based parallel computing were explicitly presented, and an efficient method of using GPUs was proposed in [32]. There are some other attempts to GPU-accelerated SLS solution using LU decomposition in [46–48]. Also, in [49,50], the implementation of parallel PF computation being conducted by adopting GPUs were described in detail. Although the methods of using GPUs in the PF study are different, there is a commonality that they tried to use the GPU for a large amount of computation for the analysis of large-scale systems [51–53]. S. Hung et al. [54] showed a typical use of GPU to solve the PF problem. They described a GPU-accelerated FDPF method and a direct linear solver using LU decomposition in detail, and simulation results using Matlab were summarized. G. Zhou et al. [55] described an LU decomposition solver that packaged a large number of LU decomposition tasks. The speed of their proposed solver was up to 76 times greater compared to the KLU library. KLU library is a solver for a sparse linear system [56].

In [32], a GPU-based Chebyshev preconditioner integrated with a GPU-based conjugate gradient solver was proposed to solve SLS for speedup. Also, X. Li et al. [57] combines a GPU with a multifrontal method to solve sparse linear equations in power system analysis, and the multifrontal method requires an inversion of the factorization of a sparse matrix to a series of a dense matrix. Since the

inversion process requires significant computation, a reduction of the computation time using GPUs was proposed.

In addition to the typical approach based on the NR method, which requires LU decomposition, in some cases, GPUs have been combined with other methods. In [50], PF computation was studied by combining GPUs and the FDPF method.

In the paper, the authors proposed a new approach using Matpower, i.e., one of Matlab's toolbox, to solve the FDPF problem by combining an iterative linear solver of a preconditioned conjugate gradient method and an Inexact Newton method, alternatively of typical LU decomposition [58,59].

The proposed GPU-based FDPF method in [50] exhibited a speedup performance improvement of up to 2.86 times over 10,000 buses compared to the CPU-based FDPF method. S. Hung et al. [60] proposed a fast batched solution for real-time optimal power flow (RTOPF) considering renewable energy penetration. A primal-dual interior point method including three kernels was used, and several test power systems were simulated on four platforms (regular CPU, parallel CPU, regular GPU, batched GPU). The batched GPU method obtained good results, with a speedup of up to 56.23 compared to a regular CPU. Moreover, an unconventional study used GPUs for computation of probabilistic power flow (PPF) rather than general PF [61]. PPF can be obtained by using Monte Carlo simulation with simple random sampling (MCS-SRS). In [61], a GPU-accelerated algorithm was proposed that could handle largescale MCS-SRS for PFF computation. There are some studies focusing on the CPU-GPU hybrid system [62,63]. In [62], an advanced PF computation technique was proposed. This technique is based on the CPU-GPU hybrid system, vectorization parallelization, and sparse techniques. In [63], the method for performing simultaneous parallel PF computation using a hybrid CPU-GPU system was proposed.

4. Typical GPU-Based PF Studies

4.1. A Study Based on LU Decomposition

In this section, we describe some representative studies that apply GPU-based parallel computing to PF analysis. The most time-consuming part of PF computation using the NR method is the LU decomposition component for matrix handling. Therefore, many studies have been conducted on fast parallel processing of LU decomposition parts using GPUs; this chapter summarizes the LU decomposition parallel processing elements of several related papers.

A GPU-based sparse solver that does not use dense kernels was proposed in [64]. The primary purpose of this study was the application of the parallel processing approach for circuit simulation, so it was not closely related to PF research. However, we mention this study here to explain the parallel processing of LU decomposition using a GPU.

Basically, LU decomposition is a popular method using pivoting. There is partial pivoting in the LU decomposition with proper permutation. In the case of partial pivoting, one of the rows and columns is permutated, and row permutation is generally performed. This can be expressed using the following formula.

$$P \cdot A = L \cdot U \tag{12}$$

In the case of full pivoting, both row and column pivoting are performed, as expressed by the following formula.

$$P \cdot A \cdot Q = L \cdot U \tag{13}$$

where P and Q represent permutation matrices, and L and U mean the lower and upper triangular matrices, respectively. Also, A is an input matrix to be decomposed.

However, this study found that LU decomposition without pivoting is more favorable for parallel processing with GPUs. Hence, a GPU-based sparse solver based on LU decomposition without pivoting was proposed. The proposed GPU solver does not use dense kernels, unlike conventional direct solvers, so it can be used for fast matrix calculation. The framework of the proposed sparse

solver is shown in Figure 3. The proposed method is designed to distribute tasks appropriately to CPUs and GPUs, and the row/column reordering performed for the pivoting is replaced by the execution of one pre-processing step on a CPU to reduce the computation time. Most of the iterations for actual factorization take place on a GPU, which receives the CPU's initial calculations. A GPU-based left-looking algorithm is used to factorize a matrix in this method, which is summarized in Algorithm 1. The result obtained through iteration on the GPU is read back from the CPU, and triangular equations are executed. The proposed approach achieves, on average, 7.02 × speedup compared with sequential PARDISO. The simulation results were obtained using a total of 16 cores of Intel E5-2690 CPU and a single NVIDIA GTX580 GPU, respectively.

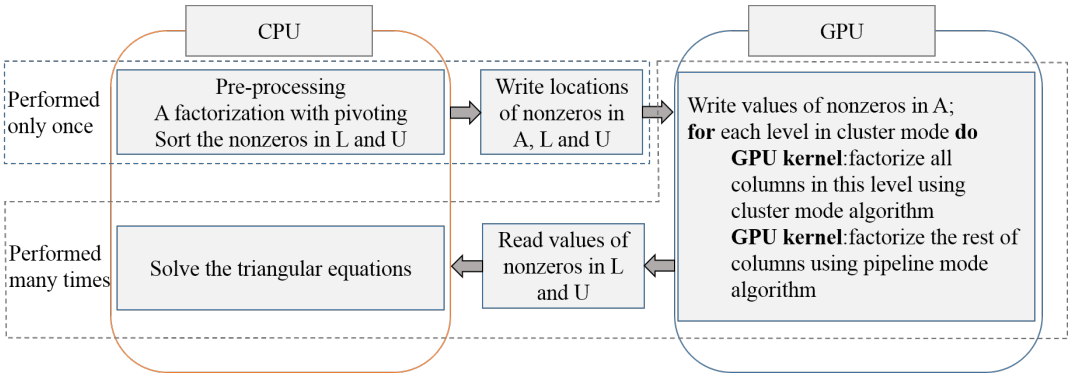


Figure 3. Framework of the sparse solver in [64].

Algorithm 1 left-looking LU factorization (GPU-based) [45,55,64].

INPUT A: an $n \times n$ input matrix to be LU factorized

OUTPUT L, U: lower and upper triangular matrices factorized from A

1: **while** there still exist executable columns **do in parallel**
   // e.g., the GPU thread j of the j-th column.
2: $\&c = A(:, j)$;
   // Renew the j-th column with all its dependent columns on the left-hand side.
3: **for** $i = 1 : j - 1$ **where** $U(i, j)$ is not equal to zero **do**
4: $c(i + 1 : n) - = c(i) \times L(i + 1 : n, i)$;
5: **end for**
6: Normalize the j-th column;
7: **end while**

In [49], a GPU-accelerated GS and NR methods for PF study were analyzed. As discussed earlier, the primary purpose of this review paper is a focus on the acceleration of matrix handling with GPU, so the use of the GS method is beyond the scope of this work. However, this is valuable because it shows the time required for each computational step of the NR method. Hence, the LU decomposition of the NR method is briefly summarized. Also, some preliminary works were presented at the level of data, software composition, and matrix representation for parallel processing using GPU; however, this is not discussed in detail here.

Table 1 summarizes the four computational steps that consume the most execution time in NR implementation. Both building an admittance matrix and obtaining apparent power and line losses are unrelated to matrix handling, and the use of the GS method does not introduce any differences here. The part to be examined carefully is the solution of a sparse linear system that produces and solves the Jacobian matrix. This study demonstrated how some of the iterative solvers provided by CUDA are beneficial for single calculations but have no significant advantages in parallel processing for large

scale PF studies. Hence, the Eigen library, an open-source C++ library, is proposed as a solution [56]. In the proposed method, each linear system is solved sequentially, but parallel processing is performed in which multiple systems are distributed to multiple OpenMP threads.

Table 1. Implementation of the NR method with average time and speedup in [49].

Major Procedures (>30 ms)	Time (ms)		Speedup
	CPU	GPU	
Build admittance matrix	154.285	2.820	54.7×
Build Jacobian matrix	61.308	2.925	21.0×
Solve linear system	972.043	55.104	17.6×
Compute S and losses on all lines	69.094	0.971	71.2×

G. Zhou et al. [55] proposed a batch-LU solution method to calculate multiple SLS in the massive-PFs problem (MPFP). The crux of the proposed method is a simultaneous solution of multiple linear systems consisting of $\mathbf{A}_i \cdot x_i = b_i$ (i = 1, 2, ..., N). This method is described in Algorithm 2 and its main features are as follows.

Algorithm 2 GPU-based batch LU factorization [55].

INPUT A: an $n \times n$ input matrix to be LU factorized

OUTPUT L, U: lower and upper triangular matrices factorized from A

1: **while** there still exist executable columns **do in parallel**
   // Thread block j for the j-th column of batch tasks of LU factorization
2: $j \leftarrow$ block ID, $t \leftarrow$ thread ID in the thread block;
3: $\&c_t = A_t(:, j)$;
   // Renew the j-th column of all the other subtasks of LU factorization.
4: **for** $i = 1 : j - 1$ **where** $U_t(i, j)$ is not equal to zero **do**
5: $c_t(i + 1 : n) - = c_t(i) \times L_t(i + 1 : n, i)$;
6: **end for**
7: Normalize the j-th column of A_t ;
8: **end while**

- Unifying sparsity pattern: All SLSs must have a uniform sparsity pattern.
- Performing reordering and symbolic analysis only once: $\mathbf{A}_i = \mathbf{L}_i\mathbf{U}_i$ can be performed only once.
- Achieving extra inter-SLS parallelism: We have achieved N (N = batch size) times parallelism automatically by packaging a large number of LU decomposition subtasks into a new large-scale batch computation task. First, the decompositions on the same indexed columns, which belong to different subtasks of LU decomposition, are designated to a single thread block. In the algorithm, A_t, L_t, and U_t mean the input matrix, output lower and upper triangular matrices, respectively, that are accessed by the t-th thread block. Second, the data of the same indexed columns are located in the adjacent address of a GPU device memory.
- Preventing thread divergence: Lines four to six in Algorithm 2 (updated warp thread operation) will execute the same branch and it does not diverge due to the identical sparsity pattern and aforementioned thread-allocation type.

This method can be used for both forward and backward substitutions of LU decomposition. In the case study using the proposed method, the maximum improvement is a 76-fold speedup compared with the KLU library.

4.2. A Study Based on QR Decomposition

In the case of a system of N elements, an $N-1$ static security analysis (SSA) is required to perform N alternating-current power flows (ACPF), so that usually incurs a tremendously high computational expense for a large-scale power system. Hence, in [28], a GPU-based batch-ACPF method using a QR solver was proposed to reduce the computation time for solving a large number of PFs of $N-1$ SSA. For the sake of clarity, the basic concept of QR decomposition is described in the background Section 2.2.3, and the overall framework of the proposed batch-ACPF solution is shown in Figure 4.

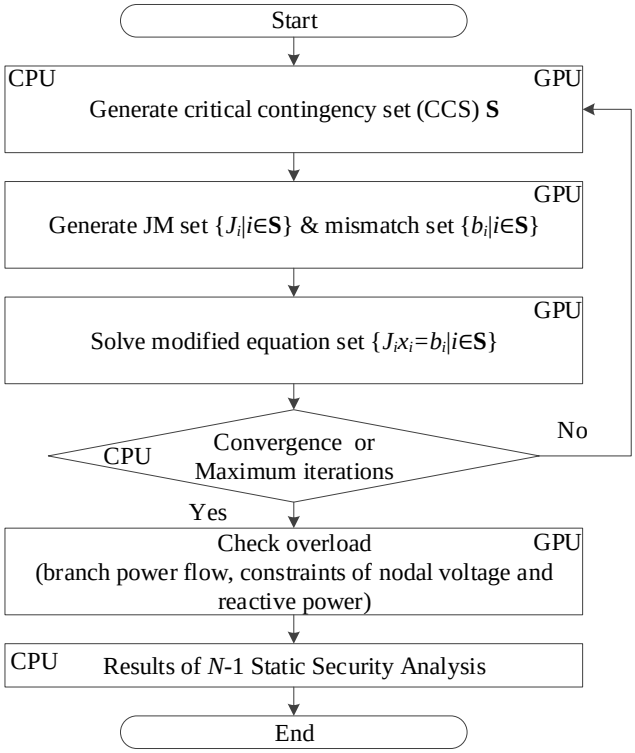


Figure 4. Framework of the proposed method in [28].

Reference [28] proposed a GPU-accelerated contingency screening algorithm to determine the critical contingency set (CCS) required for analysis. The proposed algorithm, which is described in detail in chapter VI of [28], includes dense vector operation in the execution of the algorithm.

Two GPU-based algorithms are used to solve the CCS **S** generated by the screening algorithm. First, the batch Jacobian-Matrix (batch JM) generation is employed to produce the JM set $\{J_i|i\in\mathbf{S}\}$ and power mismatch set $\{b_i|i\in\mathbf{S}\}$. The JM generation can be identified from the modified equations of ACPF analysis in Equation (14).

$$\begin{bmatrix}\Delta P\\ \Delta Q\end{bmatrix} = -J\begin{bmatrix}\Delta\theta\\ \Delta U/U\end{bmatrix} = -\begin{bmatrix}HN\\ ML\end{bmatrix}\begin{bmatrix}\Delta\theta\\ \Delta U/U\end{bmatrix} \tag{14}$$

where ΔP denotes a mismatch vector of nodal active power injection, ΔQ represents a mismatch vector of nodal reactive power injection, $\Delta\theta$ and ΔU are modification vectors of voltage angle and magnitude, respectively. Also, U means a vector of nodal voltage magnitude, H, N, M, and L indicate the sub-matrices of JM J. JM generation is necessary to calculate four sub-matrices. Each submatrix has independence by node, so the larger the power system is, the greater the parallelism.

Second, the batch-QR solver is necessary to accelerate the solution of the modified equation set $\{J_i x_i = b_{out}|i\in\mathbf{S}\}$. Reference [28] showed that a conventional QR solver for solving single SLS was not suitable for the GPU-accelerated algorithm, so a batch-QR solver for a massive number of SLSs is

proposed to overcome the disadvantages. The purpose of the batch-QR solver is to simultaneously solve the linear system $A_i x_i = b_i$ (i = 1, 2, ..., N). The proposed solver splits the entire calculation into a single QR subtask, which takes the form of a parallel computation on a GPU to achieve a higher degree of parallelism. This includes the parts for coalesced memory accesses and reordering algorithms for computation.

There are two stop criteria for batch-ACPF. One is that 80% of ACPFs have converged. The other is that the batch-ACPF solver has reached the maximum number of iterations. In [28], the maximum number of iterations was set at 10. Here, the GPU kernels in double-precision format are used to check the overload. One is used to check the branch power flow, and the other is used to check the nodal voltage and reactive power. In Table 2, we summarize the simulation results for a specific case. The first solution was designated as a baseline for performance comparison, and the speedup result for each SSA solution method is indicated to prove the validity of the proposed method.

Table 2. Performance comparison in [28]. All the other speedups were calculated against the analysis time of No. 1.

No.	SSA Solution	Analysis Time (s)	Speedup
1	CPU SSA with a single-threaded UMFPACK [65]	144.8	-
2	CPU SSA with 12-threaded PARDISO [66]	33.6	4.3×
3	CPU SSA with 12-threaded KLU [56]	9.9	14.6×
4	GPU SSA with Batch-ACPF solver [28]	2.5	57.6×

4.3. Computation Time Reduction of GPU-Based Parallel Processing

Research on GPUs to PF study can be categorized into studies aimed at reducing the execution time of matrix handling in a single computation, and studies focused on parallel processing of a large number of computations simultaneously. Many studies focused on parallel processing have suggested methods for efficiently separating matrix operations. Since most of the operations are concentrated in this area, efficiency can be increased if the time required for matrix handling can be reduced significantly [21]. The previous studies introduced in Sections 4.1 and 4.2 are typical matrix handling studies. On the other hand, some studies that attempt parallel processing for a large number of computations suggest a method of reducing the execution time by focusing on parallel processing beyond the division of matrix handling. In [63], parallel power flow using a hybrid CPU-GPU approach was studied. The NR method was used here for PF computation, and some libraries were used for CUDA implementation. This study considered the time needed for the allocation of memory or disadvantageous aspects of GPUs related to CPUs, as well as the PF computation time. The hybrid CPU-GPU approach for the proposed simultaneous parallel power flow computation is shown in Figure 5.

In [63], several systems with 14, 30, 57, 118, 300, 1354, 2869, and 9241 buses were used to obtain results under various conditions, respectively. Single and parallel calculations of 10, 50, 100, and 200 PF computations were performed simultaneously for each system, and the speedup result of simulations with the proposed method was recorded. The results show that if the power system is small, the overhead of memory allocation to the GPU is relatively high for multiple simultaneous PF computations. In contrast, when a large number of PF computations are performed, the memory allocation becomes relatively negligible as the power system grows. Furthermore, focusing the allocation on the beginning of the program reduces the bottleneck effect caused by the nature of GPU behavior. The overall results of the study show that the use of GPUs is efficient in PF studies and that

this can be applied to real-time operational planning or control actions in control centers that require large amounts of PF computation.

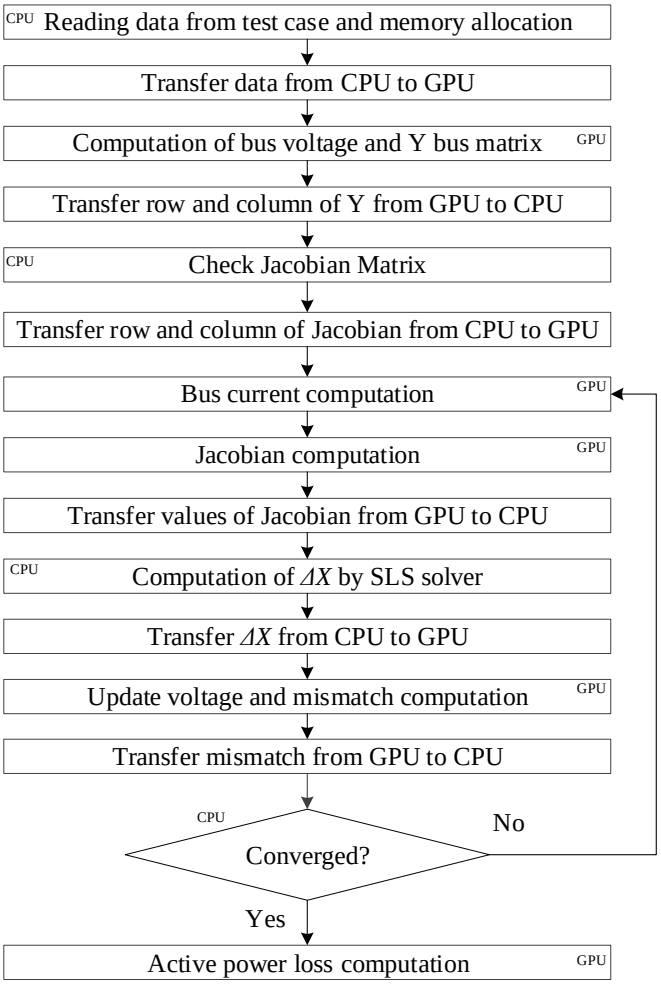


Figure 5. Hybrid CPU-GPU approach in [63].

4.4. Results and Comparison

Studies performing parallel processing using GPUs for PF are summarized in Table 3. In addition to the papers described in detail above, a number of additional meaningful studies are included.

Table 3 shows the solvers or the specific methods used for each study and the speedup results. The NR method is the most popular, and the time-consuming part of this method is matrix handling. Therefore, parallel processing studies using GPUs are also associated with the matrix handling part for speedup of computation. Many studies have been conducted on parallel processing of LU decomposition, which is frequently used in the NR method, and there have been some studies using QR decomposition associated with matrix handling. Table 3 summarizes the speedup results presented in each paper, including what reference they were compared with. The table also shows the hardware specifications of CPU and GPU, which are employed for the speedup evaluation. In some cases, the speedup is measured in comparison to the existing commercial library, and in other cases, the performance is evaluated compared to the CPU alone. Since the speedup reference is differently represented in each paper, it is difficult to understand the inherent speedup value in each case; however, it is clear that GPUs aid speedup when performing PF computations.

Table 3. Comparison of main features and speedup.

References	Main Features (Solver or Method)	Speedup (Comparison Target)	Year	Hardware Specification
[46]	LU decomposition	10.4× (in-house code)	2012	CPU: Phenom 9850 (4 cores) GPU: Tesla S1070
[45]		7.90× (1-core CPU) 1.49× (8-core CPU)	2012	CPU: Xeon X5680 (24 cores) GPU: GTX 580
[54]		4.16× (Matlab counterpart)	2017	CPU: Xeon E5-2620 (6 cores) GPU: GeForce Titan Black
[55]		76.0× (KLU library)	2017	CPU : Xeon E5-2620 (24 cores) GPU : Tesla K40
[63]	LU decomposition, hybrid CPU-GPU	Only graphs (MATPOWER)	2019	CPU : Xeon E5-2620 (6 cores) GPU : Tesla K20c
[32]	CG method	4.7× (Matlab CG)	2014	CPU: Xeon E5607 (8 cores) GPU: Tesla M2070
[28]	Batch-QR	64× (UMFPACK)	2014	CPU: Xeon E5-2620 (24 cores) GPU: Tesla K20c
[50]	GPU-based FDPF with Inexact Newton method	2.86× (traditional FDPF)	2017	CPU: Xeon E5607 (8 cores) GPU: Tesla M2070
[49]	Parallel GS, NR	GS : 45.2×, NR : 17.8× (MATPOWER)	2017	CPU: Xeon E5-2650 (32 cores) GPU: Telsa K20c
[67]	Continuous NR	11.13× (CPU)	2017	CPU: Xeon E5-2620 (6 cores) GPU: Tesla K20Xm

5. Conclusions

In this paper, we present an overview of the current state of knowledge regarding parallel power flow computation using GPU technology. The utilization of GPUs in the PF computation contributes to computational performance improvement and confirms its suitability for parallel processing. In summary, the technical trends and applications of GPUs for the PF computation are reviewed as follows:

- The basic concepts of PF computation methods and matrix handling are described.
- Conventional approaches of parallel processing technology in PF studies are summarized.
- Representative studies are reviewed to describe how to achieve parallel processing using GPUs in PF studies.
- The main features and speedup results are summarized for each study as a table form.

The parallel processing using GPUs is expected to be accelerated significantly further as their hardware performance improves more and more. We would be highly recommended to consider the utilization of GPUs in the near future for electrical power system analysis with a large amount of computations. In particular, it is expected that the GPU-based parallel processing will be very useful in the case of on-line operational planning that reflects the real time changes of the system, or performing simulations considering massive contingencies. However, the studies of performing PF analysis using the GPU techniques are still in a developing phase. Although significant speedup achievements were sufficiently convinced in the studies, but a standard technique has not been yet established completely at this moment. Also, we can expect the employment of cloud computing in the power flow analysis since the power systems are scaling up incrementally as the use of renewable energy is getting increased [68]. By exploiting the coarse and fine-grained parallelism of the PF study with the hybrid of cloud computing and GPU technologies, we can resolve the extremely massive numerical computations to evaluate the future power system properly. We have a plan to study the hybrid further soon.

Author Contributions: All authors contributed to this work. Writing—original draft preparation, D.-H.Y.; writing—review and editing, D.-H.Y. and Y.H.; supervision, Y.H. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Korea Electric Power Corporation (Grant number: R17XA05-48) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (Grant number: 2019R1C1C1008789).

Conflicts of Interest: The authors declare no conflict of interest.

Nomenclature

PF	Power flow
NR	Newton–Raphson
GS	Gauss–Seidel
EV	Electric vehicle
FDPF	Fast decoupled power flow
CG	Conjugate gradient
RTOPF	Real-time optimal power flow
SLS	Sparse linear system
CUDA	Compute Unified Device Architecture
OpenCL	Open Computing Language
HPC	High performance computing
CPU	Central processing unit
GPU	Graphics Processing Units
GPGPU	General-Purpose computing on Graphics Processing Units
SM	Streaming multiprocessor
TLP	Thread-level parallelism
PPF	Probabilistic power flow

References

1. Falcão, D.M. High performance computing in power system applications. In *International Conference on Vector and Parallel Processing*; Springer: Berlin, Germany, 1996; pp. 1–23.
2. Ramesh, V. On distributed computing for on-line power system applications. *Int. J. Electr. Power Energy Syst.* **1996**, *18*, 527–533. [CrossRef]
3. Baldick, R.; Kim, B.H.; Chase, C.; Luo, Y. A fast distributed implementation of optimal power flow. *IEEE Trans. Power Syst.* **1999**, *14*, 858–864. [CrossRef]
4. Li, F.; Broadwater, R.P. Distributed algorithms with theoretic scalability analysis of radial and looped load flows for power distribution systems. *Electr. Power Syst. Res.* **2003**, *65*, 169–177. [CrossRef]
5. Green, R.C.; Wang, L.; Alam, M. High performance computing for electric power systems: Applications and trends. In Proceedings of the 2011 IEEE Power and Energy Society General Meeting, Detroit, MI, USA, 24–28 July 2011; pp. 1–8.
6. Tournier, J.C.; Donde, V.; Li, Z. Potential of general purpose graphic processing unit for energy management system. In Proceedings of the 2011 Sixth International Symposium on Parallel Computing in Electrical Engineering, Luton, UK, 3–7 April 2011; pp. 50–55.
7. Huang, Z.; Nieplocha, J. Transforming power grid operations via high performance computing. In Proceedings of the 2008 IEEE Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, Pittsburgh, PA, USA, 20–24 July 2008; pp. 1–8.
8. Wu, Q.; Spiryagin, M.; Cole, C.; McSweeney, T. Parallel computing in railway research. *Int. J. Rail Trans.* **2018**, 1–24. [CrossRef]
9. Santur, Y.; Karaköse, M.; Akin, E. An adaptive fault diagnosis approach using pipeline implementation for railway inspection. *Turk. J. Electr. Eng. Comput. Sci.* **2018**, *26*, 987–998. [CrossRef]
10. Nitisiri, K.; Gen, M.; Ohwada, H. A parallel multi-objective genetic algorithm with learning based mutation for railway scheduling. *Comput. Ind. Eng.* **2019**, *130*, 381–394. [CrossRef]
11. Zawidzki, M.; Szklarski, J. Effective multi-objective discrete optimization of Truss-Z layouts using a GPU. *Appl. Soft Comput.* **2018**, *70*, 501–512. [CrossRef]

12. Li, Y.; Liu, Z.; Xu, K.; Yu, H.; Ren, F. A GPU-Outperforming FPGA Accelerator Architecture for Binary Convolutional Neural Networks. Available online: https://dl.acm.org/doi/abs/10.1145/3154839 (accessed on 10 February 2020).
13. Choi, K.H.; Kim, S.W. Study of Cache Performance on GPGPU. *IEIE Trans. Smart Process. Comput.* **2015**, *4*, 78–82. [CrossRef]
14. Oh, C.; Yi, Y. CPU-GPU^2 Trigeneous Computing for Iterative Reconstruction in Computed Tomography. *IEIE Trans. Smart Process. Comput.* **2016**, *5*, 294–301. [CrossRef]
15. Burgess, J. RTX on—The NVIDIA Turing GPU. *IEEE Micro* **2020**, *40*, 36–44. [CrossRef]
16. Wittenbrink, C.M.; Kilgariff, E.; Prabhu, A. Fermi GF100 GPU Architecture. *IEEE Micro* **2011**, *31*, 50–59. [CrossRef]
17. Kundur, P.; Balu, N.J.; Lauby, M.G. *Power System Stability and Control*; McGraw-hill: New York, NY, USA, 1994; Volume 7.
18. Stott, B.; Alsac, O. Fast decoupled load flow. *IEEE Trans. Power Appar. Syst.* **1974**, *PAS-93*, 859–869. [CrossRef]
19. Glover, J.D.; Sarma, M.S.; Overbye, T. Power System Analysis & Design, SI Version. Available online: https://books.google.co.kr/books?hl=ko&lr=&id=XScJAAAAQBAJ&oi=fnd&pg=PR7&dq=Power+System+Analysis+%26+Design,+SI+Version&ots=QEYUnLwXnN&sig=fofumZoNynEPX0YuRaKiO2glnA8#v=onepage&q=Power%20System%20Analysis%20%26%20Design%2C%20SI%20Version&f=false (accessed on 16 February 2020).
20. Alvarado, F.L.; Tinney, W.F.; Enns, M.K. Sparsity in large-scale network computation. *Adv. Electr. Power Energy Convers. Syst. Dyn.* **1991**, *41*, 207–272.
21. Zhou, G.; Zhang, X.; Lang, Y.; Bo, R.; Jia, Y.; Lin, J.; Feng, Y. A novel GPU-accelerated strategy for contingency screening of static security analysis. *Int. J. Electr. Power Energy Syst.* **2016**, *83*, 33–39. [CrossRef]
22. Wu, J.Q.; Bose, A. Parallel solution of large sparse matrix equations and parallel power flow. *IEEE Trans. Power Syst.* **1995**, *10*, 1343–1349.
23. Lau, K.; Tylavsky, D.J.; Bose, A. Coarse grain scheduling in parallel triangular factorization and solution of power system matrices. *IEEE Trans. Power Syst.* **1991**, *6*, 708–714. [CrossRef]
24. Amano, M.; Zecevic, A.; Siljak, D. An improved block-parallel Newton method via epsilon decompositions for load-flow calculations. *IEEE Trans. Power Syst.* **1996**, *11*, 1519–1527. [CrossRef]
25. El-Keib, A.; Ding, H.; Maratukulam, D. A parallel load flow algorithm. *Electr. Power Syst. Res.* **1994**, *30*, 203–208. [CrossRef]
26. Fukuyama, Y.; Nakanishi, Y.; Chiang, H.D. Parallel power flow calculation in electric distribution networks. In Proceedings of the 1996 IEEE International Symposium on Circuits and Systems, Circuits and Systems Connecting the World (ISCAS 96), Atlanta, GA, USA, 15 May 1996; Volume 1, pp. 669–672.
27. Davis, T.A. Direct Methods for Sparse Linear Systems. Available online: https://books.google.co.kr/books?hl=ko&lr=&id=oovDyJrnr6UC&oi=fnd&pg=PR1&dq=Direct+Methods+for+Sparse+Linear+Systems&ots=rQcLPxz3GE&sig=jYD7MDkkW_KGkK6rU6AWOstk3j8#v=onepage&q=Direct%20Methods%20for%20Sparse%20Linear%20Systems&f=false (accessed on 20 February 2020).
28. Zhou, G.; Feng, Y.; Bo, R.; Chien, L.; Zhang, X.; Lang, Y.; Jia, Y.; Chen, Z. GPU-accelerated batch-ACPF solution for N-1 static security analysis. *IEEE Trans. Smart Grid* **2017**, *8*, 1406–1416. [CrossRef]
29. Golub, G.H.; Van Loan, C.F. Matrix Computations. Available online: https://books.google.co.kr/books?hl=ko&lr=&id=5U-l8U3P-VUC&oi=fnd&pg=PT10&dq=Matrix+Computations&ots=7_JDJm_Lfp&sig=FP_n3ws8CRIxseHwoh97znxmpmE#v=onepage&q=Matrix%20Computations&f=false (accessed on 1 March 2020).
30. Dag, H.; Semlyen, A. A new preconditioned conjugate gradient power flow. *IEEE Trans. Power Syst.* **2003**, *18*, 1248–1255. [CrossRef]
31. Garcia, N. Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: A GPU-based approach. In Proceedings of the Power and Energy Society General Meeting, Providence, RI, USA, 25–29 July 2010; pp. 1–4.
32. Li, X.; Li, F. GPU-based power flow analysis with Chebyshev preconditioner and conjugate gradient method. *Electr. Power Syst. Res.* **2014**, *116*, 87–93. [CrossRef]
33. Rafian, M.; Sterling, M.; Irving, M. Decomposed Load-Flow Algorithm Suitable for Parallel Processor implementation. In IEE Proceedings C (Generation, Transmission and Distribution). Available online: https://digital-library.theiet.org/content/journals/10.1049/ip-c.1985.0047 (accessed on 3 March 2020).

34. Wang, L.; Xiang, N.; Wang, S.; Huang, M. Parallel reduced gradient optimal power flow solution. *Electr. Power Syst. Res.* **1989**, *17*, 229–237. [CrossRef]
35. Enns, M.K.; Tinney, W.F.; Alvarado, F.L. Sparse matrix inverse factors (power systems). *IEEE Trans. Power Syst.* **1990**, *5*, 466–473. [CrossRef]
36. Huang, G.; Ongsakul, W. Managing the bottlenecks in parallel Gauss-Seidel type algorithms for power flow analysis. *IEEE Trans. Power Syst.* **1994**, *9*, 677–684. [CrossRef]
37. Ezhilarasi, G.A.; Swarup, K.S. Parallel contingency analysis in a high performance computing environment. In Proceedings of the 2009 International Conference on Power Systems, Kharagpur, India, 27–29 December 2009; pp. 1–6. doi:10.1109/ICPWS.2009.5442650. [CrossRef]
38. Huang, Z.; Chen, Y.; Nieplocha, J. Massive contingency analysis with high performance computing. In Proceedings of the 2009 IEEE Power & Energy Society General Meeting, Calgary, AB, Canada, 26–30 July 2009; pp. 1–8.
39. Smith, S.; Van Zandt, D.; Thomas, B.; Mahmood, S.; Woodward, C. *HPC4Energy Final Report: GE Energy*; Technical Report; Lawrence Livermore National Laboratory (LLNL): Livermore, CA, USA, 2014.
40. Konstantelos, I.; Jamgotchian, G.; Tindemans, S.H.; Duchesne, P.; Cole, S.; Merckx, C.; Strbac, G.; Panciatici, P. Implementation of a massively parallel dynamic security assessment platform for large-scale grids. *IEEE Trans. Smart Grid* **2017**, *8*, 1417–1426. [CrossRef]
41. Guo, C.; Jiang, B.; Yuan, H.; Yang, Z.; Wang, L.; Ren, S. Performance comparisons of parallel power flow solvers on GPU system. In Proceedings of the 2012 IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Seoul, Korea, 19–22 August 2012; pp. 232–239.
42. Demmel, J.W.; Eisenstat, S.C.; Gilbert, J.R.; Li, X.S.; Liu, J.W. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Anal. Appl.* **1999**, *20*, 720–755. [CrossRef]
43. Schenk, O.; Gärtner, K. Solving unsymmetric sparse systems of linear equations with PARDISO. *Future Gener. Comput. Syst.* **2004**, *20*, 475–487. [CrossRef]
44. Christen, M.; Schenk, O.; Burkhart, H. General-Purpose Sparse Matrix Building Blocks Using the NVIDIA CUDA Technology Platform. In First Workshop on General Purpose Processing on Graphics Processing Units. Available online: https://scholar.google.co.kr/scholar?hl=ko&as_sdt=0%2C5&q=General-purpose+sparse+matrix+building+blocks+using+the+NVIDIA+CUDA+technology+platform&btnG= (accessed on 10 March 2020).
45. Ren, L.; Chen, X.; Wang, Y.; Zhang, C.; Yang, H. Sparse LU factorization for parallel circuit simulation on GPU. In Proceedings of the 49th Annual Design Automation Conference, San Francisco, CA, USA, 3–7 June 2012; ACM: New York, NY, USA, 2012; pp. 1125–1130.
46. Jalili-Marandi, V.; Zhou, Z.; Dinavahi, V. Large-scale transient stability simulation of electrical power systems on parallel GPUs. In Proceedings of the 2012 IEEE Power and Energy Society General Meeting, San Diego, CA, USA, 22–26 July 2012; pp. 1–11.
47. Chen, D.; Li, Y.; Jiang, H.; Xu, D. A parallel power flow algorithm for large-scale grid based on stratified path trees and its implementation on GPU. *Autom. Electr. Power Syst.* **2014**, *38*, 63–69.
48. Gnanavignesh, R.; Shenoy, U.J. GPU-Accelerated Sparse LU Factorization for Power System Simulation. In Proceedings of the 2019 IEEE PES Innovative Smart Grid Technologies Europe (ISGT-Europe), Bucharest, Romania, 29 September–2 October 2019; pp. 1–5.
49. Roberge, V.; Tarbouchi, M.; Okou, F. Parallel Power Flow on Graphics Processing Units for Concurrent Evaluation of Many Networks. *IEEE Trans. Smart Grid* **2017**, *8*, 1639–1648. [CrossRef]
50. Li, X.; Li, F.; Yuan, H.; Cui, H.; Hu, Q. GPU-based fast decoupled power flow with preconditioned iterative solver and inexact newton method. *IEEE Trans. Power Syst.* **2017**, *32*, 2695–2703. [CrossRef]
51. Jiang, H.; Chen, D.; Li, Y.; Zheng, R. A Fine-Grained Parallel Power Flow Method for Large Scale Grid Based on Lightweight GPU Threads. In Proceedings of the 2016 IEEE 22nd International Conference on Parallel and Distributed Systems (ICPADS), Wuhan, China, 13–16 December 2016; pp. 785–790.
52. Liu, Z.; Song, Y.; Chen, Y.; Huang, S.; Wang, M. Batched Fast Decoupled Load Flow for Large-Scale Power System on GPU. In Proceedings of the 2018 International Conference on Power System Technology (POWERCON), Guangzhou, China, 6–8 November 2018; pp. 1775–1780.
53. Zhou, G.; Feng, Y.; Bo, R.; Zhang, T. GPU-accelerated sparse matrices parallel inversion algorithm for large-scale power systems. *Int. J. Electr. Power Energy Syst.* **2019**, *111*, 34–43. [CrossRef]

54. Huang, S.; Dinavahi, V. Performance analysis of GPU-accelerated fast decoupled power flow using direct linear solver. In Proceedings of the Electrical Power and Energy Conference (EPEC), Saskatoon, SK, Canada, 22–25 October 2017; pp. 1–6.
55. Zhou, G.; Bo, R.; Chien, L.; Zhang, X.; Shi, F.; Xu, C.; Feng, Y. GPU-based batch LU factorization solver for concurrent analysis of massive power flows. *IEEE Trans. Power Syst.* **2017**, *32*, 4975–4977. [CrossRef]
56. Eigen. Eigen 3 Documentation. Available online: http://eigen.tuxfamily.org (accessed on 12 April 2020).
57. Li, X.; Li, F.; Clark, J.M. Exploration of multifrontal method with GPU in power flow computation. In Proceedings of the Power and Energy Society General Meeting (PES), Vancouver, BC, Canada, 21–25 July 2013; pp. 1–5.
58. MATPOWER. MATPOWER User's Manual. Available online: http://matpower.org (accessed on 3 February 2020).
59. MathWorks Inc. MATLAB. Available online: http://www.mathworks.com (accessed on 15 March 2020).
60. Huang, S.; Dinavahi, V. Fast batched solution for real-time optimal power flow with penetration of renewable energy. *IEEE Access* **2018**, *6*, 13898–13910. [CrossRef]
61. Zhou, G.; Bo, R.; Chien, L.; Zhang, X.; Yang, S.; Su, D. GPU-accelerated algorithm for online probabilistic power flow. *IEEE Trans. Power Syst.* **2018**, *33*, 1132–1135. [CrossRef]
62. Su, X.; He, C.; Liu, T.; Wu, L. Full Parallel Power Flow Solution: A GPU-CPU Based Vectorization Parallelization and Sparse Techniques for Newton-Raphson Implementation. *IEEE Trans. Smart Grid* **2019**, *11*, 1833–1844. [CrossRef]
63. Araújo, I.; Tadaiesky, V.; Cardoso, D.; Fukuyama, Y.; Santana, Á Simultaneous parallel power flow calculations using hybrid CPU-GPU approach. *Int. J. Electr. Power Energy Syst.* **2019**, *105*, 229–236. [CrossRef]
64. Chen, X.; Ren, L.; Wang, Y.; Yang, H. GPU-accelerated sparse LU factorization for circuit simulation with performance modeling. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 786–795. [CrossRef]
65. Davis, T. SUITESPARSE: A Suite OF Sparse Matrix Software. Available online: http://faculty.cse.tamu.edu/davis/suitesparse.html (accessed on 1 April 2020).
66. Schenk, O.; Gärtner, K. User Guide. Available online: https://www.pardiso-project.org/ (accessed on 15 April 2020).
67. Wang, M.; Xia, Y.; Chen, Y.; Huang, S. GPU-based power flow analysis with continuous Newton's method. In Proceedings of the 2017 IEEE Conference on Energy Internet and Energy System Integration (EI2), Beijing, China, 26–28 November 2017; pp. 1–5.
68. Yoon, D.H.; Kang, S.K.; Kim, M.; Han, Y. Exploiting Coarse-Grained Parallelism Using Cloud Computing in Massive Power Flow Computation. *Energies* **2018**, *11*, 2268. [CrossRef]

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).