GPU-Based Fast Decoupled Power Flow With Preconditioned Iterative Solver and Inexact Newton Method

Xue Li, *Member, IEEE*, Fangxing Li, *Fellow, IEEE*, Haoyu Yuan, *Student Member, IEEE*, Hantao Cui, *Student Member, IEEE*, and Qinran Hu, *Member, IEEE*

***Abstract*—Power flow is the most fundamental computation in power system analysis. Traditionally, the linear solution in power flow is solved by a direct method like LU decomposition on a CPU platform. However, the serial nature of the LU-based direct method is the main obstacle for parallelization and scalability. In contrast, iterative solvers, as alternatives to direct solvers, are generally more scalable with better parallelism. This study presents a fast decouple power flow (FDPF) algorithm with a graphic processing unit (GPU)-based preconditioned conjugate gradient iterative solver. In addition, the Inexact Newton method is integrated to further improve the GPU-based parallel computing performance for solving FDPF. The results show that the GPU-based FDPF maintains the same precision and convergence as the original CPU-based FDPF, while providing considerable performance improvement for several large-scale systems. The proposed GPU-based FDPF with the Inexact Newton method gives a speedup of 2.86 times for a system with over 10 000 buses if compared with traditional FDPF, both implemented based on MATLAB. This demonstrates the promising potential of the proposed FDPF computation using a preconditioned iterative solver under GPU architecture.**

***Index Terms*—Fast decoupled power flow (FDPF), graphic processing unit (GPU), inexact newton method, iterative solver, precondition, conjugate gradient (CG) method.**

I. Introduction

POWER flow is one of the most fundamental computations in power system analysis. Among various power flow algorithms, fast decoupled power flow (FDPF) [1] is a classic power flow algorithm with high efficiency and serves as the basis for many applications in practice. Meanwhile, with the development of advanced power system controls and the deployment of new sensors and meters, the interests of simulating large-scale power systems have been growing [2], [3]. Therefore, a computationally efficient and scalable power flow algorithm is of great importance to not only power flow itself but also many applications built on top of it.

Power flow is a nonlinear problem which is numerically calculated by iteratively solving a set of linear equations of $\mathbf{A}x = b$, such as in the classic Newton-Raphson power flow (NRPF) or fast decoupled power flow (FDPF). In FDPF, certain simplifications of the original NRPF are made to provide a simpler and faster solution [1]. While there are variations of the formulation, the basic FDPF can be written in Eqs. (1) (2), which shows the decoupled formulation for ΔP and ΔQ as well as the constant Jacobian matrices (B' or B''), so as to achieve fast performance. Thus, this power flow algorithm has the words "fast" and "decoupled" in its name.

$$\Delta P = [B'] \times \Delta\delta \tag{1}$$

$$\Delta Q = [B''] \times \Delta V \tag{2}$$

Direct solvers such as LU decomposition is typically used to solve the unknown vector x in $\mathbf{A}x = b$ (i.e., $\Delta\delta$ and ΔV in Eqs. (1) (2) for FDPF). However, in recent years, iterative solvers such as the Conjugate Gradient (CG) method for solving linear equations $\mathbf{A}x = b$, as an alternative to the LU-based direct solver, have gained more interest due to better scalability under the paradigm of parallel computing [4].

Fig. 1 shows a schematic comparison of the computing flowcharts of solving FDPF using a direct linear solver like LU decomposition and an iterative linear solver like CG in Figs. 1(a) and 1(b), respectively. Fig. 1(a) shows the flowchart of the original FDPF with direct solver. Fig. 1(b) shows a typical process with an iterative solver like CG, where we have two iterative loops in the overall FDPF procedure: the outer loop is the regular FDPF iteration which is a Newton iteration to solve the nonlinear power flow equation, while the inner loop is to solve linear equations in (1) (2) via an iterative CG method.

Thus, it should be noted that the word "iterative" or "iteration" in this paper can be referred to as: 1) the iterations within an iterative solution such as CG to solve $\mathbf{A}x = b$ (i.e., Eqs. (1) (2) in FDPF); or 2) the commonly known "iterations" for iteratively solving power flows. Therefore, to avoid confusion, the iteration within the CG solver is called the *inner iteration* or *CG iteration* in this paper, while the main Newton iteration in FDPF is called the *outer iteration*, *PQ iteration* or *Newton iteration* depending

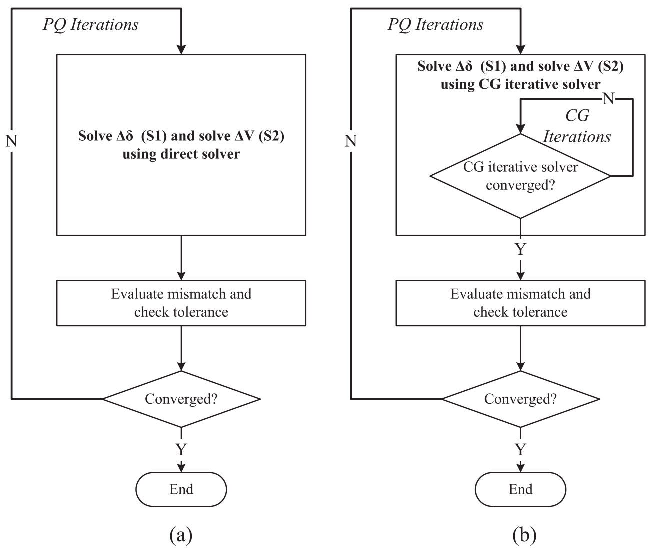
PQ Iterations
Solve Δδ (S1) and solve ΔV (S2) using direct solver
N
Evaluate mismatch and check tolerance
Converged?
Y
End
(a)
PQ Iterations
Solve Δδ (S1) and solve ΔV (S2) using CG iterative solver
N
CG Iterations
CG iterative solver converged?
Y
N
Evaluate mismatch and check tolerance
Converged?
Y
End
(b)

Fig. 1. Comparison of the flowcharts of fast decoupled power flow using direct solver (a) or iterative solver (b).

on the context. Fig. 1(b) illustrates the two levels of the iterative loops.

To improve the computing efficiency of the overall FDPF solution based on an iterative solver, preconditioning [4]–[7] and the Inexact Newton method [8], [9] can be applied. Both will be further elaborated in Section II. Also, the Graphic Processing Units (GPUs) will be the main parallel computing platform for this research work. GPU has been applied to many power system applications, such as power flow [10]–[16], optimal power flow [17], transient stability simulation [18]–[20], contingency analysis [21], because of its parallel computation capability and commodity server compatibility.

With the above motivation, this work will propose a fast decoupled power flow (FDPF) algorithm with a GPU-based preconditioned iterative CG solver. The Inexact Newton method will be plugged into the PQ iterations and the performance will be discussed. The proposed GPU-based FDPF is implemented with GPU libraries and integrated with MATPOWER, a Matlab-based software package, to test its validity and performance.

The rest of the paper is organized as follows: Section II briefly discusses iterative solvers with preconditioning and the Inexact Newton method; Section III presents the GPU-based FDPF; Section IV presents the computational results; and Section V concludes this research work.

II. Fundamentals of Iterative Solvers and the Inexact Newton Method

In this section, first, the direct solver and the iterative solver with preconditioning are discussed. Then, the Inexact Newton method to further improve iterative solvers is introduced.

A. Direct Versus Iterative Linear Solvers

It is well understood to the scientific computing community that an LU-based direct solver is the most efficient solution to solve $\mathbf{A}x = b$ in a sequential computing platform. The LU decomposition has been used in various power system applications including solving the key equations (1) (2) in FDPF. While appearing simple, a typical LU-based direct solver always needs reordering steps to reduce the potential fill-ins in the decomposition. It also needs more storage for the two triangular matrices L and U. This makes the LU-based direct solver less efficient or scalable when ported to parallel computing architecture. In contrast, iterative solvers for a linear system $\mathbf{A}x = b$ become more recognized as suitable for parallel computing.

The fundamentals and applications of iterative solvers in power flow computation under parallel computing have been extensively discussed in [4]–[6], [22]–[25]. In particular, for symmetric positive definite systems, the conjugate gradient (CG) method is usually the first choice among various iterative solvers because of its convergence rate and robustness [26], [27]. Since the B' and B'' in Eqs. (1) (2) for FDPF are symmetric positive definite, CG is applied in this work to solve FDPF under the GPU architecture.

Iterative solvers usually require preconditioners to reduce the number of iterations to improve the computational performance. A preconditioner is essentially a matrix, which will be left or right multiplied to the original matrix (i.e., the $\boldsymbol{A}$ matrix) and the right-hand side vector (i.e., the *b* vector) to precondition the linear system. Preconditioned CG has been applied to linearized power flow or fast decoupled power flow [24], [28]–[32]. This work uses the two-step preconditioner introduced in [16] for GPU-based FDPF.

Since this paper focuses on FDPF and the related matrices are symmetric positive definite, preconditioned CG will be utilized. For more general cases, the integration of a General Minimal Residual (GMRES) iterative solver with the Newton method and its preconditioners has been studied as well [5], [6], [30], [33]–[39]. Works discussing preconditioners [4], [7], [30], [40] can be applied with these iterative solvers to improve the convergence rate. Such preconditioned iterative solvers have also been used in many other power system applications, such as power system dynamics [41]–[45], state estimation [46], [47], and contingency screening [21], [24], [48].

B. Inexact Newton Method

The Inexact Newton (IN) method [8], [9] employed the concept that the inner linear solution of the Newton method does not need to be precise to maintain the convergence of the outer Newton iteration. Thus, the computation time of oversolving (i.e., reaching an unnecessary accuracy) the linear equations can be saved. This is another advantage for using iterative solvers. Flueck and Chiang [33] discussed using GMRES with the Inexact Newton method to solve power flow. De Leon and Semlyen [4] reported the performance of power flow based on the Inexact Newton method. Next, the Inexact Newton method is briefly introduced.

The classic Newton method is to solve a set of non-linear equations $F(x) = 0$ using (3), which is the generic form of Eqs. (1) and (2).

$$x_{k+1} = x_k + s_k \quad \text{where} \quad F'(x_k)\, s_k = -F(x_k) \qquad (3)$$

Therefore, each step in the Newton method involves a linear solution to reach s_k. With the direct method like LU decomposition, this is a non-problem because LU always gives the exact solution. The Inexact Newton method as showed in Eq. (4) below uses a forcing factor η to avoid the oversolving of s_k.

$$\begin{aligned} x_{k+1} &= x_k + s_k \\ \text{where} \quad F'(x_k)s_k &= -F(x_k) + e_k, \quad \|e_k\| / \|F(x_k)\| < \eta \end{aligned} \tag{4}$$

It has been proven in [8] that as long as the forcing factor η is smaller than 1, the Inexact Newton method can locally converge. The reason is that the goal of the inner iterative linear solution (e.g., the CG method) is not to obtain a precise solution for the inner CG iteration, but to help the outer Newton iteration to gradually approach the accurate solution. Even if the inner solution from the iterative CG method is not precise, the outer Newton iteration can still find a way to converge to the actual solution as long as the forcing factor is smaller than 1.

Since the inner iterative linear solver always progressively approaches the actual solution with many steps and each of the steps demands less computation, we can use the forcing factor η as the stop criterion for the inner iterative CG solver to save some computational efforts to avoid reaching unnecessarily high accuracy at the end of each outer Newton iteration. The ratio between the norm of error and right-hand side in (4) is called the relative residual that is further expanded to (5). Relative residual is a value to measure the precision of current solution s_k of the inner iterative solver. The relative residual from the inner iterative linear solver will be compared with the preset threshold, the forcing factor η, to decide whether to continue the inner iterative solution or not. The iterative solver stops when the relative residual is smaller than the preset forcing factor η.

$$rr_{k+1} = \frac{\|e_k\|}{\|F(x_k)\|} = \frac{\|F'(x_k)s_k + F(x_k)\|}{\|F(x_k)\|} \tag{5}$$

Since FDPF is a simplification of Newton-Raphson power flow and uses constant matrices B' or B'', instead of updated Jacobian matrix $F'(x_k)$ to drive to the solution, the relative residual in FDPF's inner CG iterative solution can be defined as follows:

$$rr_{k+1} = \frac{\|B \times s_k + F(x_k)\|}{\|F(x_k)\|} \tag{6}$$

where B, s_k & $F(x_k)$ can be either the combination of B', $\Delta\delta$ & ΔP or B'', ΔV & ΔQ, respectively.

The relative residual of the inner iterative method, such as CG, generally keeps a decreased trend along with its convergence. In other words, a smaller forcing factor leads to more inner iterations while a larger forcing factor is expected to use less inner iteration. Also, a smaller forcing factor usually guarantees that the outer loop has no delay to reach the solution for the outer non-linear equation, while a larger tolerance tends to demand more outer iterations to converge because of the lower precision of the inner loop. With such features, the forcing factor η, serving as the stop criterion, can be adjusted to well reach a tradeoff between the inner and outer iterations in order to improve the overall performance. Different forcing factors will be tested in this work.

C. Notes

Although both LU and CG can be implemented in sequential or parallel computing for solving a set of linear equations, $\mathbf{A}x = b$, LU decomposition is more suitable for sequential computing while a preconditioned iterative solver like CG is more suitable for parallel computing architecture. Thus, we have two main computing categories in the discussion and comparison:

1) CPU-based FDPF using LU decomposition; and
2) GPU-based FDPF and its variants based on the preconditioned CG method, with further integration of the Inexact Newton method to enhance the performance.

III. GPU-Based Fast Decoupled Power Flow

A. Basic Algorithm for GPU-Based FDPF

GPU, as an accelerator to CPU, has its advantages in linear computation because of its massive parallel computation resources. However, because it is a peripheral device, the communication between CPU and GPU is time-consuming. Therefore, an ideal GPU-based computation should satisfy the following conditions:

1) Potentials for speedup;
2) Chances for data reuse; and
3) Less communication between CPU and GPU.

Fast decoupled power flow (FDPF) with an iterative linear solver like CG has corresponding features: 1) the iterative linear solver is parallel friendly and GPU can bring a speedup [15], [49]; 2) matrices B' and B'' and their preconditioners do not need to be updated during the entire process for solving the FDPF; and 3) only vectors require updates in every PQ iteration, i.e. less data needs to be communicated. Therefore, FDPF fits the computational architecture of the GPU platform, once combined with the CG iterative linear solver.

The proposed GPU-based FDPF is implemented with Matlab on top of MATPOWER v4.1 [50] from the power engineering side. Also, from the computing side, the computational kernels in CG involve mainly sparse vector/matrix operations and are built on top of the library functions from CUBLAS [51] and CUSPARSE [52].

Because FDPF does not have to update the matrices B' and B'' every iteration, the proposed implementation will copy all matrices, including B' and B'' and their preconditioners, to GPU before entering the outer PQ iteration for reducing the overhead of GPU memory transfer. Thus, all matrices will stay on GPU during the entire FDPF computation. The inner linear computations (S1) and (S2) for solving Eq. (1) and (2), respectively, in Fig. 1 are carried out on GPU. The rest parts involve less linear computation and hence will still be finished by Matlab on CPU. The recurring communication overhead between GPU and CPU under such implementation will be minimized such that only the vectors $\Delta\delta$, ΔV, ΔP and ΔQ are needed to be updated while the copying process for matrices B' and B'' and their preconditioners will be one-time only.

The mismatches of ΔP and ΔQ will be calculated on CPU, and the active power ΔP will be sent to GPU such that GPU can solve the equation $B'\Delta\delta = \Delta P$ by the inner preconditioned CG solver to obtain $\Delta\delta$. The preconditioned system is solved by

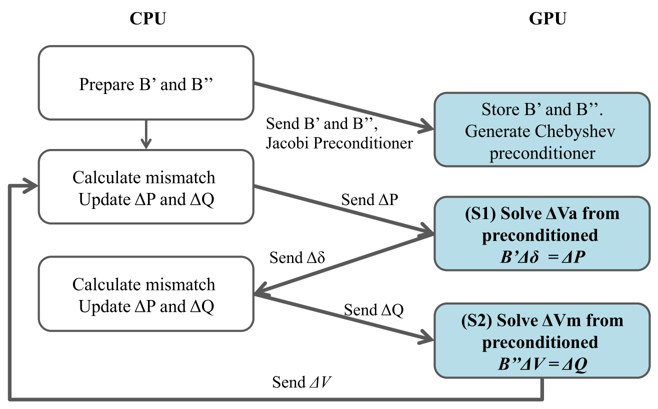
CPU
GPU
Prepare B' and B''
Send B' and B'', Jacobi Preconditioner
Store B' and B''. Generate Chebyshev preconditioner
Calculate mismatch Update ΔP and ΔQ
Send ΔP
(S1) Solve ΔVa from preconditioned B'Δδ = ΔP
Send Δδ
Calculate mismatch Update ΔP and ΔQ
Send ΔQ
(S2) Solve ΔVm from preconditioned B''ΔV = ΔQ
Send ΔV

Fig. 2. Diagram showing how GPU works with CPU to solve FDPF.

the two-step preconditioner discussed in [16] and the conjugate gradient iterative solver on GPU. Then, it will be sent back to CPU such that this updated $\Delta\delta$ can be used for CPU to calculate the nodal mismatches and obtain the new values of ΔP and ΔQ. Then, new reactive power ΔQ will be sent to GPU to obtain the solution of ΔV from the inner preconditioned $B''\Delta V = \Delta Q$ using the same preconditioning method and the inner iterative CG solver as the solution for $\Delta\delta$. With the new ΔQ value, updated ΔV will be sent back to CPU and a new PQ iteration will begin. The complete process is illustrated in Fig. 2.

With all the above discussion, it is clear that after the main algorithm starts, all the data transfer is limited to vector transferring only for the involved vectors, $\Delta\delta$, ΔV, ΔP and ΔQ. If compared with copying matrices back and forth, the vector copy is a light-weight operation. Because of the recurring usage of the copied matrices, the copy overhead is well offset by the performance improvement from the GPU computation. The elimination of unnecessary data copy and the abundant data reuse (matrices B' & B'' and their preconditioners) make such implementation promising in the computational performance improvement. This is also the reason that FDPF is particularly suitable for the designed FDPF implementation with GPU combined with the CG algorithm.

B. *GPU-Based Preconditioned Iterative Solver*

The preconditioner used in this work is a two-step preconditioner discussed in [16], which includes a diagonal Jacobi preconditioner, and a polynomial Chebyshev preconditioner. More details about the two-step preconditioner can be found in [16].

In this work, the Jacobi preconditioner is constructed with the diagonal elements from the original matrix. Its construction is less computationally intensive. Therefore, it is constructed on CPU, but the Jacobi preconditioning, along with the inner iterative CG solver, and the construction and preconditioning of the Chebyshev preconditioner are implemented with CUDA on GPU. All the GPU functions are further compiled to *mex* files for Matlab to make function calls in this work.

Note that for the inner iterative solver, although this specific preconditioned CG solver is used in this wok, the GPU-based FDPF architecture is not limited to this specific inner solver and can be integrated with other similar preconditioned iterative solvers such as GMRES as well.

TABLE I
TEST SYSTEMS BASED ON POLISH SYSTEM

System	Note
Case 2383wp	Test system from MATPOWER, Polish system
Case 3012wp	Test system from MATPOWER, Polish system
Case 10790	Case 3012wp $\times$ 2 and case 2383wp $\times$ 2
Case 13173	Case 3012wp $\times$ 2 and case 2383wp $\times$ 3

TABLE II
TEST SYSTEMS BASED ON PAN-EUROPEAN SYSTEM

System	Note
Case 1354pegase	Test system from MATPOWER, Pan-European system
Case 2869pegase	Test system from MATPOWER, Pan-European system
Case 5738	Case 2869pegase $\times$ 2
Case 9241pegase	Test system from MATPOWER, Pan-European system
Case 11624	Case 9241pegase and case 2383wp

C. *GPU-Based Preconditioned Iterative Solver with Inexact Newton Method*

As discussed in Section II-B, the Inexact Newton method with different relaxed stopping criterions (e.g., 0.01 and 0.1) for the preconditioned CG solver is applied in this work to demonstrate the further computational performance improvement of the proposed GPU-based method.

IV. COMPUTATIONAL TEST RESULTS

In this section, first, the computing experiment setup and the test matrices used in this work will be introduced. Then, the accuracy and performance of the GPU-based FDPF method will be discussed. Finally, the test results from the GPU-based FDPF with the Inexact Newton method will be presented.

A. *Computing Experiment Setup*

The computing experiments are carried out on a server equipped with NVIDIA Tesla M2070 GPU, which has 14 stream multiprocessors and each multiprocessor has 32 CUDA cores. The server has an 8-core Xeon E5607 2.27 GHz CPU and 24 GB memory. The CUDA driver version is 5.0 and the GCC version is 4.7.3. The operating system is Ubuntu 12.04. In order to keep the same precision as Matlab, all CUDA implementation is based on double-precision floating point operations. The stop criterion for the iterative solver is to have relative residual (rr) less than 10^{-3}. The degree for Chebyshev preconditioner is selected as 2 based on [7], [15].

The test systems are categorized into two groups as Table I and Table II show. The first group is based on the Polish system from MATPOWER as Table I shows. We choose case 2383wp and case 3012wp as the base cases, and use them to construct two synthetic systems, case 10790 and case 13173 as in Table I. The second group is based on the Pan-European grid as in Table II. Case 1354pegase, case 2869pegase and case

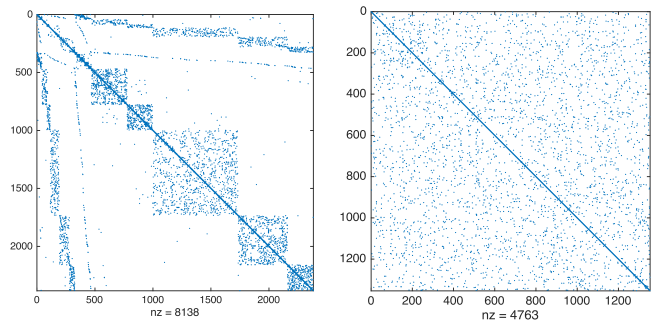

Fig. 3. Sparsity pattern of B' matrix of Case 2383wp and Case 1354pegase.

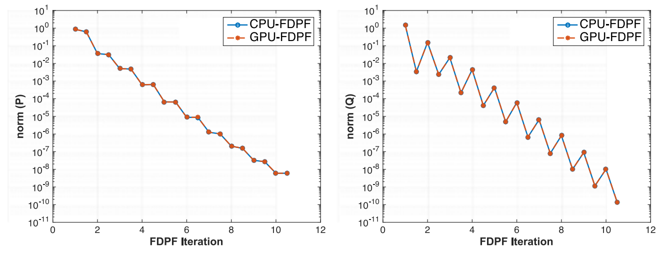

Fig. 4. Convergence comparison of P and Q between CPU-based FDPF and GPU-based PDPF for case 13173.

9241pegase are the original cases from MATPOWER. Another two more synthetic systems case 5738 and case 11624 are formed to expand the test cases as well. We choose an XB version of FDPF in MATPOWER as the basis to develop our own GPU-based FDPF package.

Fig. 3 shows the sparsity layout of B' of the typical cases case 2383wp and case 1354pegase from each category. It should be noted that the sparsity layout of B'' shows similar patterns as their corresponding B' matrices, but they are not shown here because of the space limit. Case 2383wp is one of the sample Polish systems which show several groups of locally connected subsystems, while the Pan-European system case 1354pegase gives a more coupled system. Their different connection patterns yield different computational results; therefore, we categorize them into two groups and will discuss them separately in the following subsections.

B. GPU-Based FDPF

This subsection will compare the accuracy of the original CPU-based FDPF algorithm with the GPU-based FDPF which has the preconditioned iterative solver.

Fig. 4 and Fig. 5 show the convergence process of the CPU-based and GPU-based FDPF based on MATPOWER with the largest test case from each of the two groups. Again, the CPU-FDPF is based on LU decomposition while GPU-FDPF is based on CG with preconditioning. For case 13173 in Fig. 4, GPU-FDPF based on a preconditioned iterative solver tracks the convergence of the original CPU-FDPF very well for both of the P and Q iterations. For case 11624 in Fig. 5, the P iteration of GPU-FDPF tracks CPU-FDPF as precisely as Fig. 4, while the Q iteration of GPU-FDPF has some tiny gaps if compared

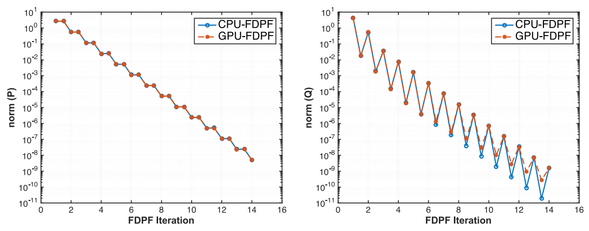

Fig. 5. Convergence comparison of P and Q between CPU-based FDPF and GPU-based PDPF for case 11624.

TABLE III
PERFORMANCE COMPARISON

System	CPU-FDPF(s)	GPU-FDPF(s)	Speedup
Case 2383wp	0.56	2.03	0.28
Case 3012wp	0.54	2.91	0.19
Case 10790	1.79	1.93	0.93
Case 13173	2.25	1.98	**1.14**
Case 1354pegase	0.14	0.38	0.38
Case 2869pegase	0.53	0.79	0.67
Case 5738	1.05	0.82	**1.29**
Case 9241pegase	8.79	8.47	**1.04**
Case 11624	9.37	8.37	**1.12**

with CPU-FDPF. However, the gap begins to happen only after norm(Q) is less than 10^{-8}. Besides, all of the test cases converge with the same number of PQ iterations as that of the CPU-FDPF. In other words, the inner linear iterative solver does not influence the convergence of the outer PQ iteration. It should be noted that the GPU-FDPF tracks CPU-FDPF very well for all the nine test systems. Not all the convergence comparisons are shown here because of space limitations. Therefore, it can be concluded that our GPU-FDPF has similar precision and convergence rate as the original CPU-based FDPF.

Table III compares the computing speed between CPU-FDPF and GPU-FDPF. It can be seen that for the test systems that are around 10000-bus scale, GPU-FDPF can provide better performance. For Pan-European systems, because their connection is more unordered, using a direct solver has no advantage here. As a result, the proposed GPU-FDPF with a preconditioned iterative solver performs better than the CPU-FDPF for the Pan-European system starting from systems larger than the scale of 5000 buses.

These results demonstrate that if we use the precise linear solution in the Newton iteration, the GPU-based inner preconditioned iterative solver can work as precisely as the original direct solvers, and it can provide performance improvement when the system is large enough to fully drive the parallel potential of GPU. Although the improvement seems to be marginal, it can be considerably enhanced with the Inexact Newton method that will be presented next.

C. GPU-Based FDPF with Inexact Newton Method

The results in the previous subsection show that the GPU-FDPF can provide similar computational results and better performance compared with the CPU-FDPF for large-scale systems. This subsection will discuss more noteworthy

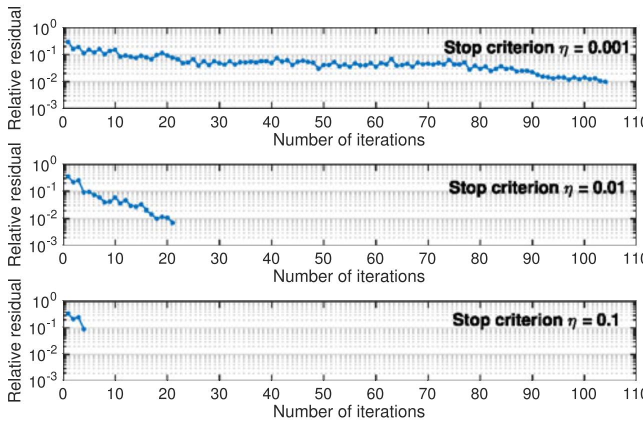

Fig. 6. Conjugate gradient convergence with stop criterion as 0.1, 0.01 and 0.001 for the solution of the first $\Delta\delta$ of FDPF for case 1354pegase.

performance improvement after the Inexact Newton (IN) method is applie to FDPF.

The inner iterative solvers use a threshold to determine dwhen to stop the iterative linear solution. It compares the relative residual (rr) defined in (6) with the preset stop criterion. If rr is larger than it, the iterative solver will keep improving the approximation of the unknowns, until rr is smaller than the stop criterion. If the Inexact Newton method is applied, the forcing factor η can be naturally used as the stop criterion.

The stop criterion is chosen as 0.01 and 0.1 to relax the precision of the inner iterative CG solver that stays inside the PQ iterations. A less precise stop criterion will always lead to fewer number of iterations for CG to converge, and hence cost less runtime. Fig. 6 shows the converge process for the first solution of $\Delta\delta$ with test system case 1354pegase. It clearly illustrates that when the stop criterion is set differently, the number of iterations changes significantly too. For the GPU-FDPF discussed in the previous subsection, the stop criterion for relative residual is set to 10^{-3}, and it takes over 100 iterations to converge to 10^{-3}. If we set the stop criterion to 0.01, as the subplot in the middle shows, the iterative solver will be converged around 20 iterations. The lower subplot even converges within 10 iterations when the stop criterion is set to 0.1. The x-axis of the three subplots in Fig. 6 is set to the same scale to better illustrate the iteration reduction. Obviously, the last subplot can provide more speedup for iterative solvers at the price of slightly reduced accuracy of the inner iteration, while the impact to outer iteration will be discussed next.

Here, we name the stop criterion for relative residual 0.1 and 0.01 with the Inexact Newton method as "IN (0.1)" and "IN (0.01)" in the following discussions for convenience. We will continue to call the original GPU-FDPF with the stopping criterion 10^{-3} in the previous subsection as GPU-FDPF for consistency.

In the following computing experiment, we still take the largest test cases from each category, case 13173 and case 11624, as examples. Fig. 7 and Fig. 8 show the convergence process in terms of every half PQ iteration with CPU-FDPF, GPU-FDPF, GPU-FDPF IN (0.1) and GPU-FDPF IN (0.01) for case 13173 and case 11624, respectively. GPU-FDPF can track the trace of

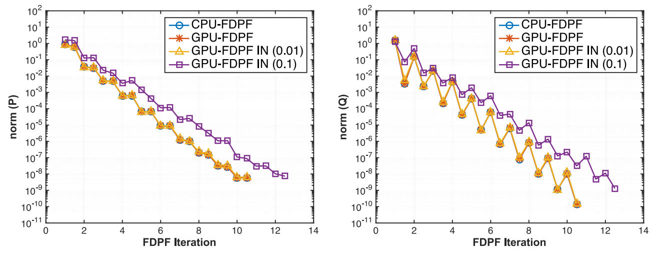

Fig. 7. Convergence comparison of P and Q between CPU-FDPF, GPU-PDPF, GPU-FDPF IN (0.01) and GPU-FDPF IN (0.1) for case 13173.

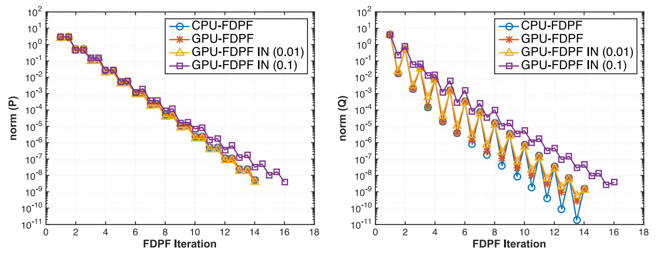

Fig. 8. Convergence comparison of P and Q between CPU-FDPF, GPU-PDPF, GPU-FDPF IN (0.01) and GPU-FDPF IN (0.1) for case 11624.

TABLE IV
COMPARISON OF PQ ITERATIONS

System	CPU-FDPF		GPU-FDPF		GPU-FDPF IN (0.01)		GPU-FDPF IN (0.1)	
	P	Q	P	Q	P	Q	P	Q
Case 2383wp	18	17	18	17	18	17	19	18
Case 3012wp	9	8	9	8	9	8	10	10
Case 10790	10	10	10	10	10	10	12	12
Case 13173	10	10	10	10	10	10	12	12
Case 1354pegase	8	7	8	7	9	8	11	11
Case 2869pegase	9	9	9	9	9	9	12	11
Case 5738	9	9	9	9	9	9	12	11
Case 9241pegase	14	13	14	13	14	13	15	14
Case 11624	14	13	14	13	14	13	16	15

CPU-FDPF precisely for most cases as discussed in the previous subsection. GPU-FDPF IN (0.01) keeps a relatively similar converging process as GPU-FDPF and CPU-FDPF, but there are some small differences between GPU-FDPF IN (0.01) and CPU-FDPF. It can still keep a similar number of PQ iterations as CPU-FDPF.

However, GPU-FDPF IN (0.1) is the one with the largest stopping criterion (0.1), i.e. the most imprecise one. The inner loop for solving linear equations generally requires fewer iterations to converge as Fig. 6 shows. However, the loss of precision in the inner loop will cause more outer iterations. For example, in Fig. 7, the original CPU-FDPF takes 14 outer PQ iteration to converge, while the GPU-FDPF IN (0.1) requires 16 outer PQ iteration. Besides, the convergence process of GPU-FDPF IN (0.1) has an obvious gap from CPU-FDPF and other implementations of GPU-FDPFs. However, GPU-FDPF IN (0.1) can always manage to converge to the preset tolerance of PQ iteration, and hence solve the overall FDPF accurately.

Table IV compares the number of PQ iterations of different implementations for all the test systems. CPU-FDPF is still the

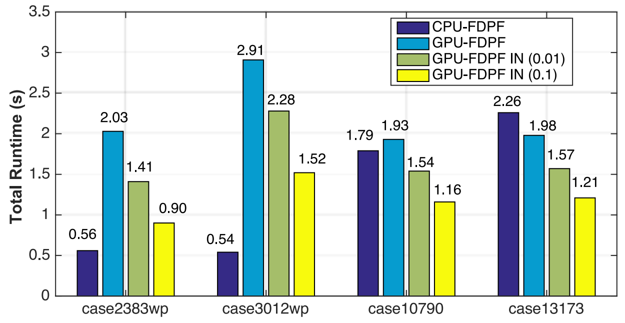

Fig. 9. Performance comparison between the original CPU-FDPF, GPU-PDPF, GPU-FDPF IN (0.01) and GPU-FDPF IN (0.1) for Polish systems.

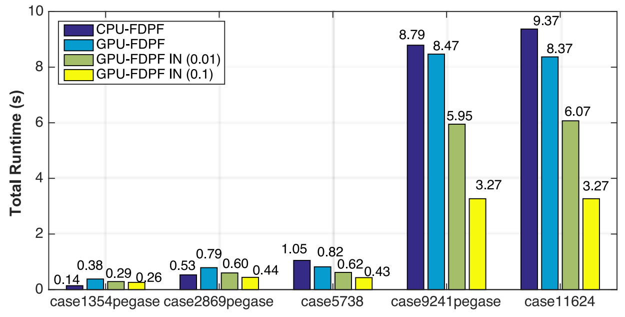

Fig. 10. Performance comparison between the original CPU-FDPF, GPU-PDPF, GPU-FDPF IN (0.01) and GPU-FDPF IN (0.1) for Pan-European systems.

TABLE V
SPEEDUP COMPARISON

System	CPU-FDPF(Baseline)	GPU-FDPF	GPU-FDPF IN (0.01)	GPU-FDPF IN (0.1)
Case 2383wp	1	0.28	0.40	0.62
Case 3012wp	1	0.19	0.24	0.36
Case 10790	1	0.93	**1.16**	**1.55**
Case 13173	1	**1.14**	**1.44**	**1.87**
Case 1354pegase	1	0.38	0.49	0.55
Case 2869pegase	1	0.67	0.89	**1.22**
Case 5738	1	**1.29**	**1.69**	**2.43**
Case 9241pegase	1	**1.04**	**1.48**	**2.68**
Case 11624	1	**1.12**	**1.54**	**2.86**

baseline. GPU-FDPF maintains exactly the same number of PQ iterations as CPU-FDPF. GPU-FDPF IN (0.01) keeps the same number of PQ iterations as CPU-FDPF and GPU-FDPF for almost all cases. However, GPU-FDPF IN (0.1) always takes two to three more PQ iterations than the other methods because of its lower inner precision.

Fig. 9 and Fig. 10 show the total runtime of the FDPF with two groups of test systems. As Table IV shows, GPU-FDPF IN (0.1) may incur a higher number of outer iterations if compared with other implementations. However, because of the significant reduction of the inner iteration, the overall performance of GPU-FDPF IN (0.1) is the best among all three GPU-FDPFs. It can also outperform the original CPU-FDPF for large systems starting around 3000 buses.

Table V provides a comparison of the speedup that different implementations can achieve based on the results from Fig. 9 and Fig. 10. It gives a more intuitive way to see that more speedup can be obtained for the inner iterative solver with less accuracy. GPU-FDPF IN (0.1) can always provide the most speedup compared with GPU-FDPF IN (0.01) and GPU-FDPF. The maximum speedup that GPU-FDPF IN (0.1) can reach is 2.86 for case 11624.

Considering the outer PQ iteration results in Table IV, we can conclude that more PQ iterations will not necessarily lead to more execution time. Although all the inexact iterative linear solutions using GPU-FDPF IN (0.1) or GPU-FDPF IN (0.01) demands one or several extra outer PQ iterations for FDPF, they can provide an overall performance improvement for systems that are sufficiently large.

V. CONCLUSION

Fast decoupled power flow (FDPF) is one of the most fundamental algorithms in power system analysis. It involves a progressive solution of a set of decoupled linear systems. The traditional approach to solve a set of linear equations, $\mathbf{A}x = b$, is LU decomposition based on the common CPU architecture. In this work, a novel parallel solution based on the GPU architecture is proposed using an iterative linear system solver (conjugate gradient method with preconditioning) combined with the Inexact Newton method, instead of LU decomposition, to solve $\mathbf{A}x = b$ and eventually the FDPF problem. The computing experiment results show that GPU-FDPF with a preconditioned iterative solver can provide the same precision as the original CPU-FDPF, and GPU-FDPF can provide performance improvement up to a speedup of 2.86 times for a case with more than 10000 buses.

The contributions of this work can be summarized as follows:

1) *Technical approach:* The proposed method in this paper best exploits the features of each individual technique: GPU application to power flow, FDPF with the Inexact Newton method, and a preconditioned iterative solver. This work discovers that these three aspects are tightly dependent on each other to yield an improved computational performance of FDPF, so it is logical to combine them to form the proposed algorithm using IN method and the preconditioned iterative solver (the inner solver).
2) *Actual implementation:* This work integrates MATPOWER and GPU implementation of the preconditioned iterative solver for the first time to the authors' best knowledge, and hence proves in practice that the use of GPU in solving the linear Jacobian equations is actually beneficial to the power flow computations.
3) *Results with archival value:* As mentioned previously, the fast decoupled power flow with the GPU-based preconditioned linear solver and Inexact Newton method has not been reported. This is the first work reporting such results. Hence, the results reported in this work can serve as a benchmark for other future related works.

This work demonstrates the great computational benefit of a GPU-based iterative solver for FDPF computation. If this approach is integrated with other power system applications that need to run power flow computations repeatedly such as contingency screening, a significant performance improvement

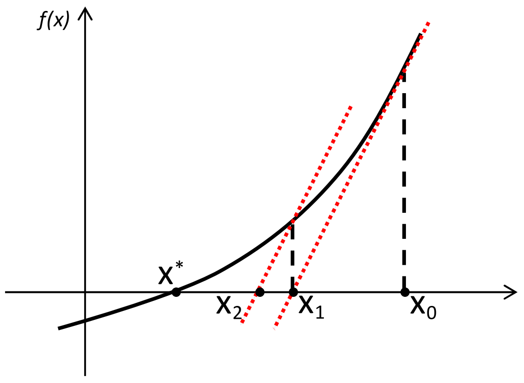

Fig. A1. An illustration of the original FDPF method.

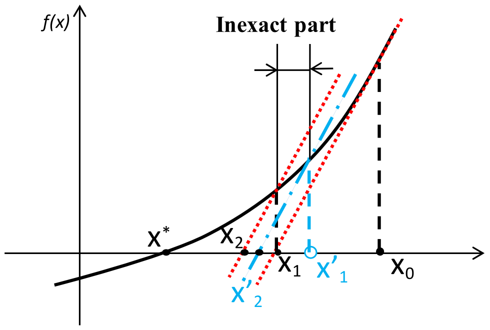

Fig. A2. An illustration of the FDPF with Inexact Newton method.

can be expected. Since power flow is the foundation of many power system applications in industrial practice as well as in research, the proposed method has a promising potential to the electric power and energy community. Further, such integration of a GPU-based preconditioned iterative solver and the Inexact Newton method can be also applied to other engineering fields using the Newton method to solve non-linear equations.

Since solving a linear system or solving a non-linear system via iteratively solving a linear system is crucial to many other disciplines, the proposed method may have a broad application in the future.

VI. Appendix

Figs. A1 and A2 give illustrative diagrams to show how the original FDPF and the Inexact Newton method with FDPF converge to the actual solution as long as the original FDPF converges. As shown in Fig. A1, the new solution after the first step is x_1, and after the second step it is x_2. Note, the same slope is applied here since FDPF uses the same Jacobian matrix. In Fig. A2, after the first step with the Inexact Newton method, the solution is x'_1 instead of x_1, and after the second step, it is x'_2 instead of x_2. This inaccuracy may demand more steps towards the final solution x^*; however, the whole process is generally expected to converge as long as the original FDPF converges. Note, a previous work [53] reported some limitations and assumptions of FDPF as opposed to Newton-Raphson power flow. More rigorous investigation on the numerical convergence performance and limitations of the Inexact Newton method can be conducted in the future.

References

[1] B. Stott and O. Alsac, "Fast decoupled load flow," *IEEE Trans. Power App. Syst.*, vol. PAS-93, no. 3, pp. 859–869, May 1974.

[2] Z. Li, V. D. Donde, J. C. Tournier, and F. Yang, "On limitations of traditional multi-core and potential of many-core processing architectures for sparse linear solvers used in large-scale power system applications," in *Proc. IEEE Power & Energy Soc. General Meeting*, San Diego, CA, USA, 2011, pp. 1–8.

[3] Y. Chen, Z. Huang, Y. Liu, M. J. Rice, and S. Jin, "Computational challenges for power system operation," in *Proc. 45th Hawaii Int. Conf. Syst. Sci.*, 2012, pp. 2141–2150.

[4] F. de Leon and A. Semlyen, "Iterative solvers in the Newton power flow problem: Preconditioners, inexact solutions and partial Jacobian updates," *IEE Proc. Gener., Transm. Distrib.*, vol. 149, pp. 479–484, Jul. 2002.

[5] H. Dag and F. Alvarado, "Direct methods versus GMRES and PCG for power flow problems," in *Proc. North Amer. Power Symp.*, 1993, pp. 274–278.

[6] A. Semlyen, "Fundamental concepts of a Krylov subspace power flow methodology," *IEEE Trans. Power Syst.*, vol. 11, no. 3, pp. 1528–1537, Aug. 1996.

[7] H. Dag and A. Semlyen, "A new preconditioned conjugate gradient power flow," *IEEE Trans. Power Syst.*, vol. 18, no. 4, pp. 1248–1255, Nov. 2003.

[8] R. S. Dembo, S. C. Eisenstat, and T. Steihaug, "Inexact Newton methods," *SIAM J. Numer. Anal.*, vol. 19, pp. 400–408, 1982.

[9] S. C. Eisenstat and H. F. Walker, "Choosing the forcing terms in an inexact Newton method," *SIAM J. Sci. Comput.*, vol. 17, pp. 16–32, 1996.

[10] N. Garcia, "Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: A GPU-based approach," in *Proc. IEEE Power & Energy Soc. General Meeting*, Minneapolis, MN, USA, 2010, pp. 1–4.

[11] D. Ablakovic, I. Dzafic, and S. Kecici, "Parallelization of radial three-phase distribution power flow using GPU," in *Proc. 3rd IEEE PES Int. Conf. Exhib. Innovative Smart Grid Technol.*, 2012, pp. 1–7.

[12] C. Guo, B. Jiang, H. Yuan, Z. Yang, L. Wang, and S. Ren, "Performance comparisons of parallel power flow solvers on GPU system," in *Proc. IEEE 18th Int. Conf. Embedded Real-Time Comput. Syst. Appl.*, 2012, pp. 232–239.

[13] X. Li, F. Li, and J. M. Clark, "Exploration of multifrontal method with GPU in power flow computation," in *Proc. IEEE Power & Energy Soc. General Meeting*, Vancouver, BC, Canada, 2013, pp. 1–6.

[14] X.-X. Liu, H. Wang, and S. X. D. Tan, "Parallel power grid analysis using preconditioned GMRES solver on CPU-GPU platforms," in *Proc. 2013 IEEE/ACM Int. Conf. Comput.-Aided Des.*, 2013, pp. 561–568.

[15] X. Li and F. Li, "GPU-based power flow analysis with Chebyshev preconditioner and conjugate gradient method," *Elect. Power Syst. Res.*, vol. 116, pp. 87–93, 2014.

[16] X. Li and F. Li, "GPU-based two-step preconditioning for conjugate gradient method in power flow," in *Proc. IEEE Power & Energy Soc. General Meeting*, Denver, CO, USA, 2015, pp. 1–5.

[17] L. Rakai and W. Rosehart, "GPU-accelerated solutions to optimal power flow problems," in *Proc. 47th Hawaii Int. Conf. Syst. Sci.*, 2014, pp. 2511–2516.

[18] V. Jalili-Marandi, Z. Zhou, and V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 7, pp. 1255–1266, Jul. 2012.

[19] V. Jalili-Marandi, Z. Zhou, and V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel GPUs," in *Proc. 2012 IEEE Power & Energy Soc. General Meeting*, 2012, pp. 1–11.

[20] Z. Yu, H. Shaowe, L. Shi, and Y. Chen, "GPU-based JFNG method for power system transient dynamic simulation," in *Proc. Int. Conf. Power Syst. Technol.*, 2014, pp. 969–975.

[21] A. Gopal, D. Niebur, and S. Venkatasubramanian, "DC power flow based contingency analysis using graphics processing units," in *Proc. IEEE Power Tech*, Lausanne, Switzerland, 2007, pp. 731–736.

[22] F. D. Galiana, H. Javidi, and S. McFee, "On the application of a preconditioned conjugate gradient algorithm to power network analysis," *IEEE Trans. Power Syst.*, vol. 9, no. 2, pp. 629–636, May 1994.

[23] M. A. Pai and H. Dag, "Iterative solver techniques in large scale power system computation," in *Proc. 36th IEEE Conf. Decision Control*, San Diego, CA, USA, 1997, vol. 4, pp. 3861–3866.

[24] A. B. Alves, E. N. Asada, and A. Monticelli, "Critical evaluation of direct and iterative methods for solving Ax=b systems in power flow calculations and contingency analysis," in *Proc. 21st IEEE Int. Conf. Power Ind. Comput. Appl.*, 1999, pp. 15–21.

[25] T. Cui and F. Franchetti, "Power system probabilistic and security analysis on commodity high performance computing systems," in *Proc. 3rd Int. Workshop High Perform. Comput., Netw. Anal. Power Grid*, 2013, pp. 1–10.

[26] J. R. Shewchuk, "An introduction to the conjugate gradient method without the agonizing pain," School Comput. Sci., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-94-125, 1994.

[27] J. Nocedal and S. J. Wright, *Conjugate Gradient Methods*. New York, NY, USA: Springer, 2006.

[28] H. Mori, H. Tanaka, and J. Kanno, "A preconditioned fast decoupled power flow method for contingency screening," *IEEE Trans. Power Syst.*, vol. 11, no. 1, pp. 357–363, Feb. 1996.

[29] Y. Wallach, "Gradient methods for load-flow problems," *IEEE Trans. Power App. Syst.*, vol. PAS-87, no. 5, pp. 1314–1318, May 1968.

[30] H. Dag and F. L. Alvarado, "Computation-free preconditioners for the parallel solution of power system problems," *IEEE Trans. Power Syst.*, vol. 12, no. 2, pp. 585–591, May 1997.

[31] F. D. Galiana, H. Javidi, and S. McFee, "On the application of a preconditioned conjugate gradient algorithm to power network analysis," *IEEE Trans. Power Syst.*, vol. 9, no. 2, pp. 629–636, May 1994.

[32] H. Dag and F. L. Alvarado, "Toward improved uses of the conjugate gradient method for power system applications," *IEEE Trans. Power Syst.*, vol. 12, no. 3, pp. 1306–1314, Aug. 1997.

[33] A. J. Flueck and H.-D. Chiang, "Solving the nonlinear power flow equations with an inexact Newton method using GMRES," *IEEE Trans. Power Syst.*, vol. 13, no. 2, pp. 267–273, May 1998.

[34] T. Feng and A. J. Flueck, "A message-passing distributed-memory Newton-GMRES parallel power flow algorithm," in *Proc. IEEE Power Eng. Soc. Summer Meeting*, 2002,vol. 3, pp. 1477–1482.

[35] Y. Chen and C. Shen, "A Jacobian-free Newton-GMRES(m) method with adaptive preconditioner and its application for power flow calculations," *IEEE Trans. Power Syst.*, vol. 21, no. 3, pp. 1096–1103, Aug. 2006.

[36] W. Xu and W. Li, "Efficient preconditioners for Newton-GMRES method with application to power flow study," *IEEE Trans. Power Syst.*, vol. 28, no. 4, pp. 4173–4180, Nov. 2013.

[37] Y.-S. Zhang and H.-D. Chiang, "Fast Newton-FGMRES solver for large-scale power flow study," *IEEE Trans. Power Syst.*, vol. 25, no. 2, pp. 769–776, May 2010.

[38] Y. Chen, C. Shen, and J. Wang, "Distributed transient stability simulation of power systems based on a Jacobian-free Newton-GMRES method," *IEEE Trans. Power Syst.*, vol. 24, no. 1, pp. 146–156, Feb. 2009.

[39] S. Abhyankar and A. J. Flueck, "Real-time power system dynamics simulation using a parallel block-Jacobi preconditioned Newton-GMRES scheme," in *Proc. High Perform. Comput., Netw., Storage Anal.*, 2012, pp. 299–305.

[40] X. Li and F. Li, "Estimation of the largest eigenvalue in Chebyshev preconditioner for parallel conjugate gradient method-based power flow computation," *IET Gener., Transm. Distrib.*, vol. 10, pp. 123–130, 2016.

[41] A. Y. Kulkarni, M. A. Pai, and P. W. Sauer, "Iterative solver techniques in fast dynamic calculations of power systems," *Int. J. Elect. Power Energy Syst.*, vol. 23, pp. 237–244, 2001.

[42] M. A. Pai, P. W. Sauer, and A. Y. Kulkarni, "A preconditioned iterative solver for dynamic simulation of power systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, Seattle, WA, USA, 1995, vol. 2, pp. 1279–1282.

[43] D. Chaniotis and M. A. Pai, "Iterative solver techniques in the dynamic simulation of power systems," in *Proc. Power Eng. Soc. Summer Meeting*, Seattle, WA, USA, 2000, vol. 1, pp. 609–613.

[44] M. A. Pai, P. W. Sauer, and A. Y. Kulkarni, "Conjugate gradient approach to parallel processing in dynamic simulation of power systems," in *Proc. Amer. Control Conf.*, 1992, pp. 1644–1647.

[45] I. C. Decker, D. M. Falcao, and E. Kaszkurewicz, "Conjugate gradient methods for power system dynamic simulation on parallel computers," *IEEE Trans. Power Syst.*, vol. 11, no. 3, pp. 1218–1227, Aug. 1996.

[46] H. Karimipour and V. Dinavahi, "Accelerated parallel WLS state estimation for large-scale power systems on GPU," in *Proc. North Amer. Power Symp.*, 2013, pp. 1–6.

[47] Y. Chen, S. Jin, M. Rice, and Z. Huang, "Parallel state estimation assessment with practical data," in *Proc. IEEE Power & Energy Soc. General Meeting*, 2013, pp. 1–5.

[48] H. Mori, H. Tanaka, and J. Kanno, "A preconditioned fast decoupled power flow method for contingency screening," in *Proc. IEEE Power Ind. Comput. Appl. Conf.*, Salt Lake City, UT, USA, 1995, pp. 262–268.

[49] A. Kamiabad and J. Tate, "Polynomial preconditioning of power system matrices with graphics processing units," in *High Performance Computing in Power and Energy Systems*, S. K. Khaitan and A. Gupta, Eds. Berlin, Germany: Springer, 2013, pp. 229–246.

[50] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.

[51] NVIDIA. (2012). *CUBLAS | NVIDIA Developer Zone*. [Online]. Available: https://developer.nvidia.com/cublas

[52] NVIDIA. (2012). *CUSPARSE LIBRARY*. [Online]. Available: http://docs.nvidia.com/cuda/pdf/CUDA_CUSPARSE_Users_Guide.pdf

[53] A. Monticelli, A. Garcia, and O. Saavedra, "Fast decoupled load flow: Hypothesis, derivations, and testing," *IEEE Trans. Power Syst.*, vol. 5, no. 4, pp. 1425–1431, Nov. 1990.