Fine-Grained Fully Parallel Power Flow Calculation by Incorporating BBDF Method into A Multi-Step NR Algorithm

Xueneng, Su, *Student Member*, *IEEE*, Tianqi Liu, *Senior Member*, *IEEE*, and Lei Wu, *Senior Member*, *IEEE*

***Abstract*—In recognizing urgent needs in fast calculation of AC power flow (PF) problems, PF computation has been explored under different parallel computing platforms. Specifically, block bordered diagonal form (BBDF) method has been widely studied to permute linear equations in PF calculations into BBDF form for facilitating parallel computation. However, determining an optimal network segmentation scheme that leads to the best speed-up ratio of BBDF based parallel PF is challenging. As a first contribution, this paper proposes a node-tearing based approach to determine optimal network segmentation scheme, which leverages sizes of subnetworks and the coordination network to achieve the best speedup ratio of BBDF based parallel PF calculation. In addition, a fine-grained fully parallel PF approach is proposed to further enhance parallel performance, in which all three key steps of the Newton-Raphson (NR) based PF calculation are implemented in parallel. Studies illustrate effectiveness of the proposed network segmentation method and fully parallel PF approach.**

***Index Terms*—Block bordered diagonal form, Newton-Raphson algorithm, parallel computation, power flow.**

I. Introduction

AC power flow (PF) analysis, which calculates steady-state status of the entire power grid with respect to given generation/load settings and network parameters, plays an essential role in many power system applications. Indeed, it is routinely utilized by system operators to analyze steady-state impacts prior to applying new settings and/or control strategies [1]. In addition, PF can evaluate impacts of variations of loads and network structures on system security, providing valuable information for designing future expansions of power systems [2]. Moreover, PF is also a key building block in short-circuit analysis and transient stability study [1]-[3], to name a few.

Mathematically, PF analysis is essentially to solve a system of nonlinear equations with the same number of variables [1]-[6]. Gauss-Seidel (GS) [1], [7], Newton-Raphson (NR) [1]-[2], [7]-[8], and fast decoupled power flow (FDPF) [4], [7] are among three widely used techniques in solving PF problems.

(i) GS method iteratively calculates phasor voltages of PQ and PV buses using nodal admittance/impedance matrix and voltage values estimated from previous iteration, until a converged phasor voltage solution is achieved. Compared to nodal admittance matrix, using nodal impedance matrix could achieve better convergence performance, at the cost of a higher memory requirement due to its full-rank matrix nature [7].

(ii) NR method iteratively calculates a system of linear equations $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$, which is derived via Taylor series expansion as an approximation of original non-linear PF equations, until a converged phasor voltage solution is achieved. Jacobian matrix $\mathbf{A}$ and vector $\boldsymbol{b}$ are iteratively updated according to solutions of previous NR iteration. $\mathbf{A}$ is usually nonsingular and sparse [1].

(iii) FDPF further simplifies Jacobian matrix calculation of NR method by taking advantage of relationship between real/ reactive power and voltage magnitude/angle. It can reduce per-iteration computing time and memory usage of PF analysis [4].

It is noteworthy that since the three algorithms use same nodal real/reactive power mismatch equations, their final PF solutions are expected to be the same. On the other hand, FDPF can provide approximate PF solutions much faster and in turn is of great benefit for contingency analysis, while NR presents a more stable convergence characteristic with fewer iterations. This paper focuses on the NR method.

With the advanced development and wide deployment of monitor and control technologies in power systems, interests in fast simulation of large-scale power systems have been stimulated [1], [3]-[4], [7], [9]-[10]. Indeed, a scalable and computationally efficient PF algorithm is of significant importance to many applications that are built on top of it. For instance, online static security assessment involves extensive PF analysis with respect to individual contingency scenarios, while fast PF calculation is key to meeting the stringent time requirement of this type of time-consuming online applications [3]. Under this environment, parallel computation is considered as an important technique for improving computational efficiency of PF analysis, with special focuses on identifying inherent parallelism of PF (especially parallelly solving $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ in each NR iteration) and exploring different parallel computing platforms. Specifically, iterative approach and block bordered diagonal form (BBDF) method are among commonly used parallel techniques for calculating a system of linear equations $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ in PF analysis.

(i) Iterative approach starts with an initial guess of state variables $\boldsymbol{x}$, and then performs a series of updates to gradually improve approximation accuracy. Iterative approach includes preconditioned conjugate gradient (PCG), bi-conjugate gradient stabilized (BiCGStab), and generalized minimal residual (GMREs) methods [4], [11], among others. A salient feature of iterative methods is that solution to $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ can be obtained without calculating the inverse of high-dimension Jacobian matrix $\mathbf{A}$. Indeed, its solution is derived by multiplication and inner product operations of relevant matrices and vectors, which can be implemented in parallel to further enhance the entire computational performance [4]. However, convergence of iterative methods highly depends on spectrum or condition number of $\mathbf{A}$ matrix [11].

(ii) BBDF method [7] is designed to split an original network into one coordination network and multiple independent subnetworks that can be calculated in parallel to accelerate computation. In BBDF, state variables of individual subnetworks are independent from each, and only coupled with the coordination network. Thus, once state variables of the

This work was supported in part by the U.S. National Science Foundation grants PFI:BIC-1534035 and CNS-1647135. X. Su and T. Liu are with the School of Electrical Engineering and Information, Sichuan University, Chengdu, 610065, China (e-mails: xsu@clarkson.edu, tqliu@scu.edu.cn). L. Wu is with the ECE Department, Clarkson University, Potsdam, NY, 13699, USA (e-mail: lwu@clarkson.edu).

coordination network are calculated, state variables of individual subnetworks can be subsequently conducted in parallel. Based on how subnetworks are created, BBDF can be categorized into node-tearing (i.e., creating subnetworks by removing certain nodes), branch-cutting (i.e., generating subnetworks by removing certain branches), and hybrid-dividing methods (i.e., considering both nodes and branches) [7]-[8], [12]-[13]. Specifically, different from node-tearing approach, Jacobian matrices of subnetworks derived from branch-cutting method could be singular. Moreover, branch-cutting method usually takes longer computational time because of relatively higher-dimension Jacobian matrices of subnetworks, and could be ineffective if nodal power injections instead of nodal current injections are given. In addition, hybrid approach is attractive when only certain nodal phasor voltages and branch power flows are of concern.

Both methods have been applied in literature [2], [4], [12]-[14] to calculate PF problems in parallel. In [2], GPU-based parallel implementations of GS, NR, and FDPF approaches combined with Gauss elimination were compared, with speedup ratios of 0.05x, 1.74x, and 1.30x, respectively. Reference [14] discussed a GPU-based parallel BiCGStab algorithm, with a speedup ratio of 2.1x. In [4], a PCG based parallel FDPF approach was implemented with GPUs, with speedup ratio of up to 2.86x. Moreover, reference [12] discussed a node-tearing network partitioning strategy, which could derive a relatively small coordination network while the other important prerequisite of similar-sized subnetworks has been neglected. Reference [13] further extended [12] to analyze impacts of different network division patterns on acceleration efficiency of parallel PF calculation.

In addition, iterative method usually requires hundreds of cores to achieve a noticeable computational improvement. However, its performance may be unstable and additional time-consuming procedures, e.g., Cholesky preconditioner and incomplete-LU, have to be applied [3], [15], especially when Jacobian matrix is ill-conditioned. In comparison, BBDF based parallel PF calculation is relatively robust. Moreover, among alternative approaches within BBDF, node-tearing method usually presents a better computational performance and has been widely implemented in many applications such as parallel PF calculation [12] and parallel transient analysis [16]. For these reasons, this paper studies a node-tearing based BBDF method for calculating PF problem in parallel.

Indeed, computational performance of BBDF based NR method for parallel PF calculation is affected by several major factors. Specifically, a key to enhancing BBDF performance is a small coordination network and multiple equally-sized subnetworks, which are viewed as two criteria for assessing quality of network segmentation schemes in BBDF calculation. However, to our best knowledge, studies that simultaneously pursue these two criteria are rather rare, while [12]-[13] and [16] mainly focus on identifying a relatively minimal coordination network but the other important prerequisite of equally-sized subnetworks has been neglected. Moreover, most BBDF based parallel PF studies in literatures [12]-[13] focus on computing $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ in parallel, while potentials for calculating other key steps of PF analysis in parallel, namely forming nodal power mismatch vectors and updating Jacobian matrix $\mathbf{A}$, have been neglected. In fact, as shown in Table I, these two steps spend considerable efforts, about 20% and 38% of the total PF calculation time as compared to 39% for solving $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$. Detailed information of these five test systems can be referred to in [17]-[18]. As a result, further exploring parallel implementations of these two important steps would be beneficial in improving acceleration performance of the entire PF calculation. It is noteworthy that for the sake of fair comparison between the proposed approach and conventional parallel PF methods in literature [12]-[13] while also focusing on fully exploiting BBDF's power in all three key steps of NR based PF calculation, computational times listed in Table I are retrieved as a base case via our own codes, in which all steps of PF are implemented in sequential while no advanced technologies (i.e., sparse technique and vectorization parallelization) are utilized. Indeed, this is also the reason why third-party APIs or libraries (e.g., Intel MKL and ArrayFire3.5.1) are not used to solve linear equations.

TABLE I COMPUTATIONAL TIME OF DIFFERENT STEPS IN PF (UNIT: MS/%)

System	Admittance Matrix	Nodal Injection Vector	Jacobi Matrix	$\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ Solution	Others
1354pegase	100/0.53%	4220/22.26%	7000/36.92%	7560/39.88%	79/0.42%
2383wp	408/0.79%	10900/21.01%	19547/37.67%	21010/40.49%	22/0.04%
2736sp	726/1.06%	13972/20.38%	26990/39.38%	26797/39.10%	58/0.08%
2869pegase	708/0.82%	19846/22.96%	33239/38.46%	32059/37.10%	572/0.66
3012wp	787/0.94%	16985/20.39%	32065/38.50%	32628/39.17%	825/0.99

This paper proposes a fine-grained fully parallel PF calculation to enhance its computational performance, by incorporating node-tearing based BBDF into NR method while also implementing three major PF calculation steps in a parallel manner. Specifically, a node-tearing based network division approach is explored to derive a minimal coordination network as well as multiple independent subnetworks whose sizes are relatively close. Moreover, in order to further boost up parallel efficiency, the proposed method explores inherent parallelism of three major steps of PF calculation, including the formation of nodal power mismatch vector, the update of Jacobian matrix, and the computation of system of linear equations.

Major contributions of the paper include:

(i) A node-tearing based approach is proposed to determine optimal network segmentation scheme, which leverages sizes of subnetworks and the coordination network to achieve the best speedup ratio of parallel PF calculation.

(ii) A fully parallel PF calculation approach is explored, which considers parallel implementation of the three key steps in BBDF based NR method. Specifically, two alternative ways for conducting parallel computation of nodal power mismatch vectors and Jacobian matrices with different granularities are investigated. In comparison, parallel calculation potentials of these two key steps have been largely neglected in literature.

(iii) A fine-grained parallel implementation for computing state variables of the coordination network is studied. Specifically, different from conventional BBDF based approach that first computes state variables of the coordination network in a single thread and then calculates state variables of individual subnetworks in parallel [7], the idea of BBDF on parallelly calculating subnetworks is further extended to compute state variables of the coordination network in parallel. This would further significantly enhance computational performance.

The remainder of this paper is organized as follows. Section II describes PF problem and conventional BBDF based parallel PF computation. Node-tearing based network segmentation

approach and details of the proposed fine-grained fully parallel PF are presented in Section III. Section IV discusses numerical studies, and Section V concludes the paper.

II. PF Problem and BBDF-Based Parallel Computation

A. Power Flow Problem

For an N-bus system, PF calculation is represented as in (1).

$$\widehat{\boldsymbol{P}}_m-\boldsymbol{V}_m\sum_{n=1}^{N}\boldsymbol{V}_n(\mathbf{G}_{mn}\cos\boldsymbol{\delta}_{mn}+\mathbf{B}_{mn}\sin\boldsymbol{\delta}_{mn})=0,\ m=1,\cdots,N\text{-}1 \quad (1a)$$

$$\widehat{\boldsymbol{Q}}_m-\boldsymbol{V}_m\sum_{n=1}^{N}\boldsymbol{V}_n(\boldsymbol{G}_{mn}\sin\boldsymbol{\delta}_{mn}-\mathbf{B}_{mn}\cos\boldsymbol{\delta}_{mn})=0,\ m=1,\cdots,Z \quad (1b)$$

where nodes $\{1,\cdots, Z\}$ are PQ buses, nodes $\{Z+1,\cdots,N\text{-}1\}$ are PV buses, and node N is the reference bus; $\widehat{\boldsymbol{P}}_m/\widehat{\boldsymbol{Q}}_m$ is given active/reactive power injection of node m; $\boldsymbol{V}_m/\boldsymbol{V}_n$ is voltage magnitude of node m/n; $\boldsymbol{\delta}_{mn}$ is voltage phase angle difference between nodes m and n; $(\mathbf{G}_{mn}+\mathrm{j}\mathbf{B}_{mn})$ is the $(m, n)^{\text{th}}$ element of nodal admittance matrix in rectangular coordinates.

Nonlinear PF equations (1) can be solved via NR algorithm, which iteratively computes (2)-(4) until real and reactive power mismatches $\Delta\boldsymbol{P}_m^k$ and $\Delta\boldsymbol{Q}_m^k$ are smaller than a predefined threshold (i.e., 10^{-6}p.u. is used in this paper). Specifically, (2a)/(2b) calculates nodal active/reactive power mismatch with respect to state variable solutions in iteration k, and (3)-(4) update state variables by solving Taylor series expansion-based linearization form of the original PF equations. Elements of Jacobian matrix $\mathbf{J}^k$ in iteration k are defined as in (5)-(6).

$$\Delta\boldsymbol{P}_m^k=\widehat{\boldsymbol{P}}_m-\boldsymbol{V}_m^k\sum_{n=1}^{N}\boldsymbol{V}_n^k(\mathbf{G}_{mn}\cos\boldsymbol{\delta}_{mn}^k+\mathbf{B}_{mn}\sin\boldsymbol{\delta}_{mn}^k),\quad m=1,\cdots,N\text{-}1 \quad (2a)$$

$$\Delta\boldsymbol{Q}_m^k=\widehat{\boldsymbol{Q}}_m-\boldsymbol{V}_m^k\sum_{n=1}^{N}\boldsymbol{V}_n^k(\mathbf{G}_{mn}\sin\boldsymbol{\delta}_{mn}^k-\mathbf{B}_{mn}\cos\boldsymbol{\delta}_{mn}^k),m=1,\cdots,Z \quad (2b)$$

$$\Delta\boldsymbol{S}^k=\begin{bmatrix}\Delta\boldsymbol{P}^k\\ \Delta\boldsymbol{Q}^k\end{bmatrix}=\begin{bmatrix}\mathbf{H}^k & \mathbf{M}^k\\ \mathbf{K}^k & \mathbf{L}^k\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{\delta}^k\\ \Delta\boldsymbol{V}^k/\boldsymbol{V}^k\end{bmatrix}=\mathbf{J}^k\Delta\boldsymbol{X}^k \quad (3)$$

$$\boldsymbol{X}^{k+1}=\begin{bmatrix}\boldsymbol{\delta}^{k+1}\\ \boldsymbol{V}^{k+1}\end{bmatrix}=\begin{bmatrix}\boldsymbol{\delta}^k\\ \boldsymbol{V}^k\end{bmatrix}+\begin{bmatrix}\Delta\boldsymbol{\delta}^k\\ \Delta\boldsymbol{V}^k\end{bmatrix}=\boldsymbol{X}^k+\Delta\boldsymbol{X}^k \quad (4)$$

$$\mathbf{H}_{mn}^k=\partial\Delta\boldsymbol{P}_m^k/\partial\boldsymbol{\delta}_n^k=-\boldsymbol{V}_m^k\boldsymbol{V}_n^k(\mathbf{G}_{mn}\sin\boldsymbol{\delta}_{mn}^k-\mathbf{B}_{mn}\cos\boldsymbol{\delta}_{mn}^k),\quad m\neq n \quad (5a)$$

$$\mathbf{M}_{mn}^k=\boldsymbol{V}_n^k\partial\Delta\boldsymbol{P}_m^k/\partial\boldsymbol{V}_n^k=-\boldsymbol{V}_m^k\boldsymbol{V}_n^k(\mathbf{G}_{mn}\cos\boldsymbol{\delta}_{mn}^k+\mathbf{B}_{mn}\cos\boldsymbol{\delta}_{mn}^k),\ m\neq n \quad (5b)$$

$$\mathbf{K}_{mn}^k=\partial\Delta\boldsymbol{Q}_m^k/\partial\boldsymbol{\delta}_n^k=\boldsymbol{V}_m^k\boldsymbol{V}_n^k(\mathbf{G}_{mn}\cos\boldsymbol{\delta}_{mn}^k+\mathbf{B}_{mn}\sin\boldsymbol{\delta}_{mn}^k),\quad m\neq n \quad (5c)$$

$$\mathbf{L}_{mn}^k=\boldsymbol{V}_n^k\partial\Delta\boldsymbol{Q}_m^k/\partial\boldsymbol{V}_n^k=-\boldsymbol{V}_m^k\boldsymbol{V}_n^k(\mathbf{G}_{mn}\sin\boldsymbol{\delta}_{mn}^k-\mathbf{B}_{mn}\cos\boldsymbol{\delta}_{mn}^k),\ m\neq n \quad (5d)$$

$$\mathbf{H}_{mm}^k=\partial\Delta\boldsymbol{P}_m^k/\partial\boldsymbol{\delta}_m^k=\boldsymbol{V}_m^k\sum_{n=1,n\neq m}^{N}\boldsymbol{V}_n^k(\mathbf{G}_{mn}\sin\boldsymbol{\delta}_{mn}^k-\mathbf{B}_{mn}\cos\boldsymbol{\delta}_{mn}^k) \quad (6a)$$

$$\mathbf{M}_{mm}^k=\boldsymbol{V}_m^k\partial\Delta\boldsymbol{P}_m^k/\partial\boldsymbol{V}_m^k=-\boldsymbol{V}_m^k\sum_{n=1,n\neq m}^{N}\boldsymbol{V}_n^k(\mathbf{G}_{mn}\cos\boldsymbol{\delta}_{mn}^k+\mathbf{B}_{mn}\sin\boldsymbol{\delta}_{mn}^k)-2\mathbf{G}_{mm}\boldsymbol{V}_m^k\boldsymbol{V}_m^k, \quad (6b)$$

$$\mathbf{K}_{mm}^k=\partial\Delta\boldsymbol{Q}_m^k/\partial\boldsymbol{\delta}_m^k=-\boldsymbol{V}_m^k\sum_{n=1,n\neq m}^{N}\boldsymbol{V}_n^k(\mathbf{G}_{mn}\cos\boldsymbol{\delta}_{mn}^k+\mathbf{B}_{mn}\sin\boldsymbol{\delta}_{mn}^k) \quad (6c)$$

$$\mathbf{L}_{mm}^k=\boldsymbol{V}_m^k\partial\Delta\boldsymbol{Q}_m^k/\partial\boldsymbol{V}_m^k=-\boldsymbol{V}_m^k\sum_{n=1,n\neq m}^{N}\boldsymbol{V}_n^k(\mathbf{G}_{mn}\sin\boldsymbol{\delta}_{mn}^k-\mathbf{B}_{mn}\cos\boldsymbol{\delta}_{mn}^k)+2\mathbf{B}_{mm}\boldsymbol{V}_m^k\boldsymbol{V}_m^k, \quad (6d)$$

B. BBDF-Based Parallel Computation of PF Problems

According to [7], [12]-[13], a straightforward way to implement parallel PF calculation is to solve linear equations (3) in a parallel manner. Specifically, (3) can be represented in a BBDF form (7) by dividing the original network into W subnetworks and one coordination network t, where $\Delta\boldsymbol{X}_w=[\Delta\boldsymbol{\delta}_w{}^T\ \Delta\boldsymbol{V}_w{}^T]^T$ is state variable vector of the w^{th} subnetwork.

$$\begin{bmatrix}\Delta\boldsymbol{S}_1\\ \vdots\\ \Delta\boldsymbol{S}_W\\ \Delta\boldsymbol{S}_t\end{bmatrix}=\begin{bmatrix}\mathbf{J}_{11} & \mathbf{0} & \mathbf{0} & \mathbf{J}_{1t}\\ \mathbf{0} & \ddots & \mathbf{0} & \vdots\\ \mathbf{0} & \mathbf{0} & \mathbf{J}_{WW} & \mathbf{J}_{Wt}\\ \mathbf{J}_{t1} & \cdots & \mathbf{J}_{tW} & \mathbf{J}_{tt}\end{bmatrix}\begin{bmatrix}\Delta\boldsymbol{X}_1\\ \vdots\\ \Delta\boldsymbol{X}_W\\ \Delta\boldsymbol{X}_t\end{bmatrix} \quad (7)$$

By observing that individual subnetworks are independent from each other and only coupled with the coordination network, (7) can be calculated by computing smaller-scale linear equations (8) and (9) in a queue. Equation (8) calculates $\Delta\boldsymbol{X}_t$ of the coordination network, then $\Delta\boldsymbol{X}_w$ of each subnetwork can be computed via (9) in parallel using solution of $\Delta\boldsymbol{X}_t$ from (8). Moreover, (8) involves the inverse of a large diagonal block matrix, which is equivalent to inversing individual small block submatrices. This can further convert (8) into a simplified equivalent form (10).

$$\left\{\mathbf{J}_{tt}-\begin{bmatrix}\mathbf{J}_{t1}\\ \vdots\\ \mathbf{J}_{tW}\end{bmatrix}^{\mathrm{T}}\begin{bmatrix}\mathbf{J}_{11} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \ddots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{J}_{WW}\end{bmatrix}^{-1}\begin{bmatrix}\mathbf{J}_{1t}\\ \vdots\\ \mathbf{J}_{Wt}\end{bmatrix}\right\}\Delta\boldsymbol{X}_t=\Delta\boldsymbol{S}_t-\begin{bmatrix}\mathbf{J}_{t1}\\ \vdots\\ \mathbf{J}_{tW}\end{bmatrix}\begin{bmatrix}\mathbf{J}_{11} & \mathbf{0} & \mathbf{0}\\ \mathbf{0} & \ddots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \mathbf{J}_{WW}\end{bmatrix}^{-1}\begin{bmatrix}\Delta\boldsymbol{S}_1\\ \vdots\\ \Delta\boldsymbol{S}_W\end{bmatrix} \quad (8)$$

$$\Delta\boldsymbol{X}_w=\mathbf{J}_{ww}^{-1}\Delta\boldsymbol{S}_w-\mathbf{J}_{ww}^{-1}\mathbf{J}_{wt}\Delta\boldsymbol{X}_t,\quad w=1,\cdots,W \quad (9)$$

$$\left(\mathbf{J}_{tt}-\sum_{w=1}^{W}\mathbf{J}_{tw}\mathbf{J}_{ww}^{-1}\mathbf{J}_{wt}\right)\Delta\boldsymbol{X}_t=\Delta\boldsymbol{S}_t-\sum_{w=1}^{W}\mathbf{J}_{tw}\mathbf{J}_{ww}^{-1}\Delta\boldsymbol{S}_w \quad (10)$$

III. The Proposed Fine-Grained Fully Parallel PF

This section first discusses a node-tearing based approach to identify optimal network segmentation scheme, followed by parallel implementations of the three key steps of BBDF based NR method for PF calculation.

A. Node-Tearing Based Network Segmentation Approach

In this section, the fast-splitting method [19]-[20], a community-identification approach in complex network theory, is first adopted to identify a limited number of weak edges that can divide an original network into one coordination network and several independent subnetworks following the BBDF form. However, sizes of derived subnetworks may not be necessarily close. Thus, we propose a node-tearing based network segmentation approach by including two additional steps: transfer to node-tearing based segmentation while using weak edges, and further combining certain subnetworks to balance sizes of subnetworks. The goal is to leverage sizes of subnetworks and the coordination network to achieve the best acceleration performance of parallel PF calculation.

A.1 Rank Edges According to Edge-Clustering Coefficient

Fast-splitting approach uses edge-clustering coefficient to identify weak edges among communities [20]. Specifically, edge-clustering coefficient $C_{ij,g}$ of an edge (i,j) connecting communities i and j is defined in [20] as the number of g-sided polygons $Z_{ij.g}$ over the total number of possible g-sided polygons containing this edge [20]. The total number of possible g-sided polygons containing edge (i,j) can be calculated as $\min[d_i\text{-}1,d_j\text{-}1]$, where d_i/d_j is degree of node i/j.

However, a potential issue with this definition is that when $Z_{ij.g}$ is zero, $C_{ij,g}$ is always zero regardless of d_i and d_j. To solve this issue, the modified edge-clustering coefficient (11) is adopted in this paper [20]. In addition, quadrilateral is usually considered to be the most appropriate and effective option for calculating this coefficient [19], i.e., g=4.

$$C_{ij.g}^{'}=(Z_{ij.g}+1)/(\min[d_i\text{-}1,d_j\text{-}1]) \quad (11)$$

Consequently, all edges in a network can be ranked in a list based on their edge-clustering coefficients as follows, while those with relatively small $C_{ij.g}^{'}$ are regarded as weak edges.

(i) Initialize the list of edges as null;

(ii) Calculate edge-clustering coefficients of all edges in the

current network topology;
(iii) Add the edge with the smallest edge-clustering coefficient into the list, and remove it from the current network topology;
(iv) Repeat above Steps (ii)-(iii) until all edges of the original network are removed.

A.2 Transfer to Node-Tearing Based Segmentation

Edges ranked via edge-clustering coefficients could be used successively to divide an original network into one coordination network and several independent subnetworks following the BBDF form. This network segmentation approach is referred to as branch-cutting based method [7]. However, it is recognized that Jacobian matrices of subnetworks generated via this method might be singular. Thus, we further discuss a procedure to transfer branch-cutting based splitting to node-tearing based segmentation.

Fig. 1 is used to illustrate the idea. Adopting fast-splitting method to the orignal network Fig. 1a, edges (1,5) and (2,5) are identified as the top two weak edges. Figs. 1b and 1c show two possible node-tearing based transformations using these two weak edges. By contrast, Fig. 1c derives a smaller coordination network (i.e., 1 node versus 2 nodes in Fig. 1b) and subnetworks of more consistent sizes (i.e., 4/4 versus 2/5 in Fig. 1b).

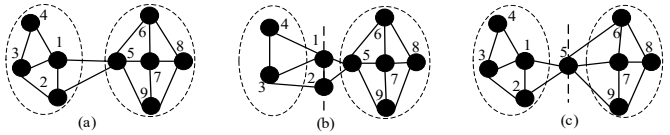

Fig. 1 Transfer from cutting-branches to torn-nodes.

Based on above observations, the following procedure is used to guide tranformation from branch-cutting based splitting to node-tearing based segmentation:
(i) Set initial torn-node collection (denoted as *coList*) as null and initial edge list as the one obtained from Section III.A.1. Initialize index k=1.
(ii) Select the k^{th} edge from edge list (denote its two nodes as i and j) to conduct node-tearing based segmentation. That is, we remove this edge from the current network topology, and calculate partial node adjacency degrees of the two nodes as $\left(\mathbf{1}_k(i)+\mathbf{1}_{k+1}(i)\right)$ and $\left(\mathbf{1}_k(j)+\mathbf{1}_{k+1}(j)\right)$ [13]. $\mathbf{1}_k(i)$ is an indicator function, which is equal to 1 if node i is a terminal node of edge k and 0 otherwise. Then, the following processes are executed:
  (ii.1) If node adjacency degrees of the two nodes are not equal, the node with a larger adjacency degree is regarded as a potential torn node, set k=k+2, and go to Step (iii);
  (ii.2) Otherwise, check whether removing node i/j from the current network will introduce new islanding nodes,
  - If neither of the two nodes introduces new islanding nodes, randomly select one as a potential torn node, set k=k+1, and go to Step (iii);
  - If one introduces new islanding nodes, regard the other as a potential torn node, set k=k+1, and go to Step (iii);
  - If both introduce new isolate nodes, set k=k+1, and go back to Step (ii);

(iii) If the torn node identfied in Step (ii) is already contained in *coList*, go back to Step (ii);
(iv) Use depth-first algorithm (DFS) [21] to derive preliminary network division pattern (denoted as $\boldsymbol{Nt}$=[$\boldsymbol{Nt}_1$ $\boldsymbol{Nt}_2$ … $\boldsymbol{Nt}_S$], where S is the number of subnetworks);
(v) If size of the largest subnetwork is no larger than a prespecified threshold (denoted as $N\rho$, where N is number of nodes in the original network and ρ represents a ratio in size of the largest subnetwork to the orginal network), all torn-nodes and the corresponding network segmentation scheme are identified; Otherwise, go back to Step (ii).

A.3 An Optimization Based Approach to Further Balance Sizes of Subnetworks

It is expected that a preferable network segmentation scheme would induce a relatively minimal coordination network together with a set of equally-sized subnetworks [7]. The former is granted via the approach discussed in Sections III.A.1-III.A.2, while the latter has been largely neglected. That is, sizes of subnetworks $\boldsymbol{Nt}$ derived from Section III.A.2 may be diverse rather than close to the expected value $N\rho$.

In this section, based on the preliminary division pattern $\boldsymbol{Nt}$ derived in Section III.A.2, we intend to further combine certain smaller subnetworks to generate S'=round($(N-n_t)/N\rho$) new subnetworks so that difference between sizes of new subnetworks and the expected value $N\rho$ is minimized. This can be modelled as a nonlinear integer problem (12)-(13). Equation (12), similar to standard deviation, quantitatively characterizes dispersed degree among sizes of new subnetworks and the expected value $N\rho$.

$$F=\min\left\{\sqrt{\sum_{j=1}^{S'}\left(\sum_{i=1}^{S}\boldsymbol{x}_{i,j}\cdot\boldsymbol{Nt}_i-N\rho\right)^2/(S'\text{-}1)}\right\} \tag{12}$$

$$\sum_{j=1}^{S'}\mathbf{x}_{i,j}=1,\quad \forall\mathbf{x}_{i,j}\in\{0,1\} \tag{13}$$

Where binary variable $\mathbf{x}_{i,j}$ indicates if an original subnetwork i is contained in a new subnetwork j, for i=1,⋯,S and j=1,⋯,S'.

The nonlinear integer problem (12)-(13) can be solved by various algorithms, such as branch-and-bound [22], particle swarm optimization (PSO) [23]-[24], and genetic algorithm (GA) [24]. Here, PSO with random inertial weight (RIW-PSO) is applied. Detailed setting on parameters used in RIW-PSO of this paper can be referred to in [24]. Indeed, the family of inertial weight based PSO methods includes three major branches: linearly-decreasing inertial weight based PSO (LS-PSO), self-adaptive inertial weight based PSO (SA-PSO), and RIW-PSO. As compared to the other two, RIW-PSO algorithm presents better convergence statistics, i.e., smaller number of iterations and lower probability of trapping into local optimum [24].

B. A Fine-Grained Fully Parallel PF Calculation

Using the optimal network segmentation scheme derived from Section III.A, this section focuses on parallel implementations of three key steps of the BBDF based NR method for PF calculation. More details can be found in [25].

B.1 Parallel Formation of Nodal Power Mismatch Vectors

For a node-tearing based network segmentation scheme, nodal power mismatch vectors of individual subnetworks can be formed independently. That is, for the w^{th} subnetwork, its nodal power mismatch vector $\Delta\boldsymbol{S}_w$=$\left[\Delta\boldsymbol{P}_w;\ \Delta\boldsymbol{Q}_w\right]$ = [$\Delta\boldsymbol{P}_{w,1}$; ⋯; $\Delta\boldsymbol{P}_{w,np_w}$;$\Delta\boldsymbol{Q}_{w,1}$;⋯ ;$\Delta\boldsymbol{Q}_{w,npq_w}$] only relies on state variables of this subnetwork and the coordination network, where np_w is total number of PQ & PV nodes and npq_w is number of PQ nodes in the w^{th} subnetwork. Thus, once state variables of the coordination network are calculated, multiple threads can be executed in parallel to calculate nodal power mismatch vectors

of individual networks.

Indeed, there are two alternative ways to conduct parallel calculation of nodal power mismatch vectors. The first one is to create (W+1) threads and calculate $\Delta \boldsymbol{S}_w$ w=1,⋯,W of individual subnetworks and $\Delta \boldsymbol{S}_t$ of the coordination network in parallel, which is referred to as PI-NPM-1. A more fine-grained one is to create 2(W+1) threads and calculate $\Delta \boldsymbol{P}_w$ and $\Delta \boldsymbol{Q}_w$ w=1,⋯,W of individual subnetworks as well as $\Delta \boldsymbol{P}_t$ and $\Delta \boldsymbol{Q}_t$ of the coordination network in parallel, which is referred to as PI-NPM-2. Case studies will evaluate whether exploring additional independency of $\Delta \boldsymbol{P}_w$ and $\Delta \boldsymbol{Q}_w$ could further enhance computational performance.

B.2 Parallel Computation of Jacobian Matrices

In each NR iteration, the coordination network and idividual subnetworks recalculate corresdponing Jacobian matrices included in (9)-(10) using updated state variable solutions. Specifically, for the w^{th} subnetwork, two Jacobian matrices are calculated, i.e., $\mathbf{J}_{ww}$=[$\partial \boldsymbol{P}_w/\partial \boldsymbol{\delta}_w$ $\partial \boldsymbol{P}_w/\partial \boldsymbol{V}_w$; $\partial \boldsymbol{Q}_w/\partial \boldsymbol{\delta}_w$ $\partial \boldsymbol{Q}_w/\partial \boldsymbol{V}_w$] and $\mathbf{J}_{wt}$=[$\partial \boldsymbol{P}_w/\partial \boldsymbol{\delta}_t$ $\partial \boldsymbol{P}_w/\partial \boldsymbol{V}_t$; $\partial \boldsymbol{Q}_w/\partial \boldsymbol{\delta}_t$ $\partial \boldsymbol{Q}_w/\partial \boldsymbol{V}_t$]. In addition, the coordination network calculates one Jacobian matrix $\mathbf{J}_{tt}$=[$\partial \boldsymbol{P}_t/\partial \boldsymbol{\delta}_t$ $\partial \boldsymbol{P}_t/\partial \boldsymbol{V}_t$; $\partial \boldsymbol{Q}_t/\partial \boldsymbol{\delta}_t$ $\partial \boldsymbol{Q}_t/\partial \boldsymbol{V}_t$] and W Jacobian matrices $\mathbf{J}_{tw}$=[$\partial \boldsymbol{P}_t/\partial \boldsymbol{\delta}_w$ $\partial \boldsymbol{P}_t/\partial \boldsymbol{V}_w$; $\partial \boldsymbol{Q}_t/\partial \boldsymbol{\delta}_w$ $\partial \boldsymbol{Q}_t/\partial \boldsymbol{V}_w$] for w=1,⋯,W.

Jacobian matrices of individual subnetworks and the coordination network can also be computated in parallel. Likewise, there are two ways to calculate Jacobian matrices in a parallel manner. A coarse-grained scheme is to create (W+1) threads to calculate {$\mathbf{J}_{ww}$, $\mathbf{J}_{wt}$} w=1,⋯,W of individual subnetworks and {$\mathbf{J}_{t1}$, … ,$\mathbf{J}_{tW}$, $\mathbf{J}_{tt}$} for the coordination network. On the other hand, it is noteworthy that there are W $\mathbf{J}_{ww}$ matrices, W $\mathbf{J}_{wt}$ matrices, W $\mathbf{J}_{tw}$ matrices, and one $\mathbf{J}_{tt}$. Thus, a more fine-grained parallel way is to create (3W+1) threads and assign individual Jacobian matrices $\mathbf{J}_{ww}$, $\mathbf{J}_{wt}$, $\mathbf{J}_{tw}$, and $\mathbf{J}_{tt}$ onto distinct processing units, which could further enhance utilization of available computing resources and accelerate Jacobian matrix formulation at smaller granularity. These two schemes, respectively termed as PI-JM-1 and PI-JM-2, will be quantitatively analyzed in numerical cast studies.

B.3 Parallel Calculation of Linear Equations

It is noticed that a major computational burden of (9) for each subnetwork comes from $\mathbf{J}_{ww}^{-1}\mathbf{J}_{wt}$, while (10) calculates summation of $\mathbf{J}_{tw}\mathbf{J}_{ww}^{-1}\mathbf{J}_{wt}$ for all subnetworks. Thus, computational complexity of (10) is analogous to, if not more than, total complexity of (9) for all W subnetworks. However, when applying BBDF to calculate PF in parallel, a widely used approach is to first compute (10) for the coordination network and then calculate (9) of individual subnetworks in parallel [12]. That is, although (10) for the coordination network is computational expensive, its potential parallel computation capability has not been fully explored.

This section further explores parallel computation of (10) for the coordination network. Specifically, W threads are first created to parallelly calculate $\mathbf{J}_{tw}\mathbf{J}_{ww}^{-1}\mathbf{J}_{wt}$ and $\mathbf{J}_{tw}\mathbf{J}_{ww}^{-1}\Delta \boldsymbol{S}_w$ of individual subnetworks, and then (10) is solved for updating coordination network's state vector $\Delta \boldsymbol{X}_t$. Moreover, for each of the W threads, intermedia result $\mathbf{J}_{tw}\mathbf{J}_{ww}^{-1}$ is first computed, which can be used to calculate both $\mathbf{J}_{tw}\mathbf{J}_{ww}^{-1}\mathbf{J}_{wt}$ and $\mathbf{J}_{tw}\mathbf{J}_{ww}^{-1}\Delta \boldsymbol{S}_w$. After that, (9) of individual subnetworks are computed in parallel. Computational performance of the above parallel approach, referred to as PA-2-BBDF, will be compared with conventional sequential method without BBDF (CSM-NBBDF), sequential method applied in conjunction with BBDF (SM-BBDF), and BBDF based parallel approach in [12] (PA-1-BBDF) in the case study section.

Indeed, linear equations associated with individual subnetworks can be directly solved via mainstream libraries such as Intel MKL, Eigen, ArrayFire, and KLU from SuiteSparse. However, for the purpose of fully exploiting BBDF's power in all three key steps of the NR based PF calculation and further conducting a fair comparison with other BBDF based parallel approach in literature [12], BBDF, instead of those libraries, is adopted to solve linear equations associated with each network.

C. Procedure of the Proposed Parallel PF Implementation

Detailed procedure of the proposed fully parallel PF implementation, along with the determination of optimal threshold ρ, are summarized as following:

(i) *Initialization*

Initialize range of the network segmentation threshold $\rho \in [\rho_l, \rho_u]$ and a step λ for discretizing the range, i.e., different values of ρ in $\{\rho_l, \rho_l+\lambda, \cdots, \rho_u\}$ will be evaluated. Initialize PF convergence criteria *tol*.

(ii) For each ρ value, conduct the following studies:

a) Network Segmentation

(a.1) Rank all edges using the fast-splitting algorithm;

(a.2) Transfer from branch-cutting based splitting to node-tearing based segmentation;

(a.3) Combine certain subnetworks to further balance sizes of subnetworks.

b) Full Parallel PF Calculation

(b.1) Initialize voltage magnitiutes of PQ nodes, and voltage phase angles of PQ and PV nodes;

(b.2) Form nodal power mismatch vectors in parallel, and record the maximum power mismatch value $\Delta PQ^{\max}$;

- If $\Delta PQ^{\max}$ is less than *tol*, final PF solution is obtained. Record current ρ as well as related calculation time;
- Otherwise, update Jacobian matrices in parallel, calculate the solution to $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ in parallel to compute state vectors, and go back to Step (b.2).

(iii) *Retrieve Optimal Threshold*

Compare calculation time associated with individual ρ values to identify the optimal threshold that corresponds to the best speedup ratio.

D. Computational Complexity of the Proposed PF Approach

Reference [10] revealed that computational burdens of CSM-NBBDF and CM-BBDF mainly come from calculating solution to $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$, and their computational complexities are $O(N^3)$ and $O(N_{w\max}^3)$ where N and $N_{w\max}$ are sizes of Jacobian matrices associated with the original network and the largest subnetwork involved in the BBDF method.

Moreover, as the major difference between PA-1-BBDF and the proposed PA-2-BBDF lies in whether state vector of the coordination network is updated in parallel, total computational requirements of PA-1-BBDF and PA-2-BBDF are termed as $\sum_{w=1}^{W}(N_w^3+2N_tN_w^2-N_tN_w+2N_wN_t^2-N_t^2)+N_t^3+(N_s^3+2N_tN_s^2-N_tN_s)$ and

$N_s^3+2N_tN_s^2-N_tN_s+2N_sN_t^2-N_t^2+N_t^3+N_s^3+2N_tN_s^2-N_tN_s$ where $N_s = \max\{N_1,\ldots,N_W\}$. That is, computational complexities of PA-1-BBDF and PA-2-BBDF are both $O(N_{w\max}^3)$. However, although computational complexities of the three approaches SM-BBDF, PA-1-BBDF, and PA-2-BBDF theoretically are the same, their actual computational needs differ significantly by considering actual numbers of subnetworks and low-order terms, i.e., $\sum_{w=1}^{W}(2N_w^3+4N_tN_w^2+2N_wN_t^2)+N_t^3$, $\sum_{w=1}^{W}(N_w^3+2N_tN_w^2-N_tN_w+2N_wN_t^2-N_t^2)+N_t^3+(N_s^3+2N_tN_s^2-N_tN_s)$, and $N_s^3+2N_tN_s^2-N_tN_s+2N_sN_t^2-N_t^2+N_t^3+N_s^3+2N_tN_s^2-N_tN_s$. Consequently, as will be observed in numerical studies, acceleration performance of PA-2-BBDF would apparently outperform the other two.

IV. Numerical Studies

This section uses the five power systems listed in Table I, together with several other benchmark systems, to illustrate effect of the proposed fine-grained fully parallel PF approach. Impacts of different network segmentation schemes and full parallel of all three key steps in the BBDF based NR method on speedup ratio and parallel efficiency of PF calculation are quantitatively analyzed. All case studies are carried out on a Windows 10 64-bits server with two 8-core Intel Xeon 2.1Ghz CPUs and 64 GB memory, and the program is implemented in C# language through IDE VS 2015 Enterprise and Task Parallel Library (TPL). If not mentioned otherwise, all 32 logical cores are used for parallel PF calculation.

A. The 1354pegase System

A.1 Associated BBDF Form of the 1354pegase System

Using the threshold of ρ=0.08 (impact of ρ on efficiency of PF calculation will be analyzed in Section IV.A.3, while Fig. 4 in Section IV.A.3 shows that 0.08 derives the best speedup ratio for this system), BBDF-form Jacobian matrix of this system is depicted in Fig. 2. Specifically, the original network is divided into 11 subnetworks and 1 coordination network (denoted as "11+1"), in which the largest/smallest subnetwork contains 123/108 nodes and the coordination network includes 126 nodes, all less than 10% of nodes in the original network. That is, sizes of all 11 subnetworks are close and the number of torn-nodes is also relatively small, which are two important factors for assessing quality of network segmentation schemes.

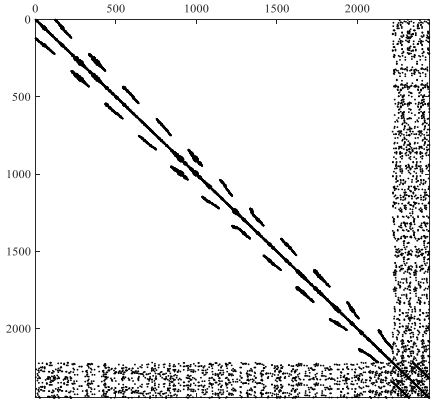

Fig. 2 BBDF-Form Jacobian matrix of the 1354pegase system.

Fig. 3 shows convergence performance of the RIW-PSO based optimal network segmentation with threshold of ρ=0.08. The optimal network segmentation scheme is obtained after 106 iterations in about 997s. This computational time is significantly longer than PF calculations, which is mainly caused by repeated time-consuming DFS process in the node-tearing based segmentation approach to search for division patterns after each new torn-node is identified. Indeed, time and space complexities of DFS are $O(V+E)$ and $O(V^2)$ for a graph with V nodes and E edges [21]. It is worth mentioning that for a given network topology, because optimal network segmentation only needs to be executed offline once while online PF calculations are usually executed routinely. Consequently, as will be observed in following case studies, such a time-consuming offline optimal network segmentation process is worthwhile as it could significantly enhance acceleration performance of parallel PF calculations.

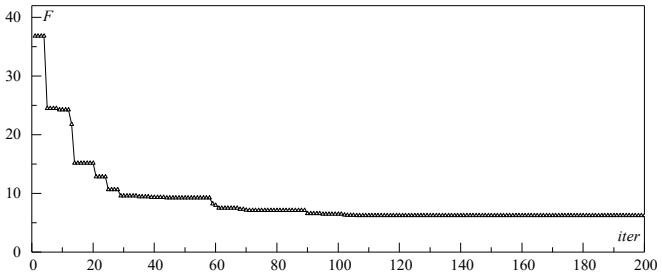

Fig. 3 Convergence performance of RIW-PSO based optimal network segmentation with ρ=0.08.

A.2 Speedup Ratio Improvement of the Three Key Steps in PF

The optimal network segmentation scheme presented in above Section IV.A.1 is used in this section. Table II shows computational performance of sequential and two parallel strategies to form nodal power mismatch vectors and update Jacobian matrices. It is noteworthy that PF of this system converges in 6 NR iterations, and final $\Delta PQ^{\max}$ of individual iterations for both sequential and parallel approaches are the same as [34.25 5.63 0.25 $0.65e10^{-2}$ $0.79e10^{-4}$ $0.13e10^{-9}$], which verifies solution accuracy of the proposed approaches. In addition, times reported in Table II are total time of all iterations. Results show that parallel calculation could reduce computation time of forming nodal power mismatch vectors by about one order of magnitude, while about 3-5 times for Jacobian matrix calculation. Specifically, with the fine-grained parallel computation strategies PI-NPM-2 and PI-JM-2, speedup ratios of the two steps are about 11.05 and 4.83.

Table II Speedup Ratios of the First Two Steps in PF Calculation

Approach	Process	Time (ms)	Speedup Ratio
Sequential	Form nodal power mismatch vector	4220	-
	Update Jacobian matrices	7000	-
PI-NPM-1	Form nodal power mismatch vector	527	8.01
PI-JM-1	Update Jacobian matrices	1970	3.55
PI-NPM-2	Form nodal power mismatch vector	382	11.05
PI-JM-2	Update Jacobian matrices	1450	4.83

Table III further shows computational time of various approaches for calculating $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$. Although both CSM-NBBDF and SM-BBDF are implemented in sequential, SM-BBDF takes about 1611ms shorter than CSM-NBBDF. The reason is that computational complexity of conventional NR and BBDF are respectively $O(N^3)$ and $O(N_{w\max}^3)$ [10]. This indeed is a salient feature of BBDF over conventional NR for speeding up calculation. Moreover, PA-2-BBDF is about 2 times faster than

PA-1-BBDF, which reveals that parallel computation of (10) for the coordination network could further enhance acceleration performance of PF computation.

TABLE III SPEEDUP RATIOS FOR SOLVING LINEAR EQUATIONS

Approach	CSM-NBBDF	SM-BBDF	PA-1-BBDF	PA-2-BBDF
Total Time (ms)	7560	5949	3242	1384
Speedup Ratio	-	1.27	2.33	5.46

A.3 The Determination of Optimal Threshold

This section further explores impact of different ρ values, and consequently different network segmentation schemes, on computational performance of the proposed parallel PF. Optimal threshold refers to the one with the best speedup ratio. It is noteworthy that optimal threshold is system specific, while once the network segmentation pattern is fixed, optimal threshold remains the same with respect to different load levels. Reference [7] suggests that a proper threshold could be within the range of [0.02, 0.4]. Fig. 4 shows speedup ratios of the entire PF calculation against different ρ values in [0.02, 0.4] with a step of 0.02. Two extreme ρ values of 0.01 and 0.95 are also studied to evaluate impact of extreme settings on parallel PF. As for the RIW-PSO algorithm, number of swarms is set to 50 and maximum number of iterations is 200 [24].

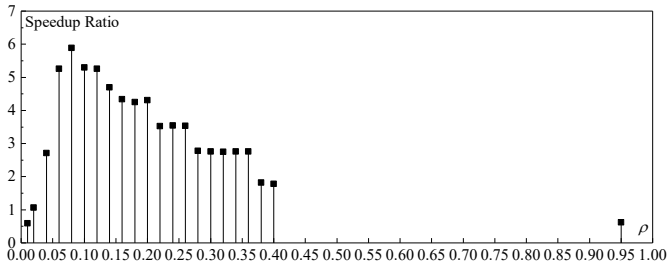

Fig. 4 Speedup ratios with respect to different thresholds.

Fig. 4 shows that speedup ratio first increases with the increase in ρ. While after reaching the peak speedup ratio of 5.89 with ρ of 0.08, it gradually decreases. The reason of a rapid increase of speedup ratio in the first phase is that: as ρ gradually increases, (i) size of the coordination network decreases, and computation time for solving its state vector via (10) is reduced; and (ii) number of subnetworks also decreases, which in turn reduces communication burden between the coordination network and subnetworks Specifically, when ρ takes values of 0.01, 0.02, 0.04, 0.06, and 0.08, sizes of the coordination network respectively are 375, 354, 197, 153, and 126, while numbers of subnetworks respectively are 71, 37, 21, 15, and 11. On the other hand, when ρ is within the range of [0.10, 0.95], size of the coordination network varies between 122 (for ρ of 0.10) and 17 (for ρ of 0.95), and number of subnetworks ranges from 9 (for ρ of 0.10) to 1 (for ρ of 0.95). Specifically, with a larger ρ, both size of the coordination network and number of subnetworks are reduced, which consequently lead to shorter computational time for solving the coordination network and lower communication overhead between the coordination network and subnetworks. However, as scales of subnetworks are much larger (i.e., ranges from 150 to 1337 versus from 20 to 123 with ρ of [0.01, 0.08]), computational times for subnetworks are significantly longer. In turn, after reaching the peak at ρ of 0.08, speedup ratio gradually reduces when ρ further increases.

In summary, Fig. 4 clearly shows that size of the coordination network and subnetworks' expected size $N\rho$ are two key factors that could impact computational performance of parallel PF. Specifically, a larger size of the coordination network corresponds to a longer computational time of the coordination network, while calculation time of individual subnetworks are shorter because of their smaller sizes. On the other hand, when individual subnetworks' expected size decreases with a smaller ρ, the number of subnetworks and consequently communication overhead between the coordination network and subnetworks are larger, although computational time of individual subnetworks is shorter. As shown in Fig. 4, these two values are tightly coupled, and the optimal speedup ratio of 5.89 is reached with ρ of 0.08 to leverage the two.

B. Extensive Studies on Other Systems

This section conducts additional studies including: (i) the proposed approach is compared with other traditional segmentation approaches [12], [25]; (ii) parallel PF study is further evaluated via six systems, including 2383wp, 2736sp, 2869pegase, 3012wp, IEEE 30-bus, and 300-bus, which are respectively indexed as 2-7, with 1354pegase indexed as 1.

B.1 Comparison with Other Segmentation Approaches

The proposed approach is compared with two segmentation approaches in [12], [26] via 300-bus and 25-bus systems. Table IV reveals that differences in numbers of nodes contained in the largest and smallest subnetworks from the proposed approach are smaller (i.e., 2/0 versus 57/5). In addition, computational time of the approach in [26] is about two times of the proposed approach. These studies clearly show advantages of the proposed network segmentation approach.

TABLE IV COMPARISON OF DIFFERENT SEGMENTATION APPROACHES

System		300-bus		25-bus	
		Proposed	[12]	Proposed	[26]
Number of nodes	Pattern	4+1	4+1	3+1	3+1
	The largest subnetwork	65	98	7	10
	The smallest subnetwork	63	41	7	5
Computational time (s)		0.18	N/A	0.068	0.13

B.2 Optimal Network Segmentation Schemes

Following the similar idea in Section III.A.3 for the 1354pegase system, Table V shows optimal network segmentation schemes of these six systems that would derive the best speedup ratio. Specifically, sizes of coordination networks for the six systems are respectively 7.05%, 12.28%, 11.36%, 9.99%, 16.67%, and 14.67% of their original networks. In addition, differences in sizes of the largest and the smallest subnetworks are 24, 19, 13, 16, 2, and 2 nodes, which are respectively about 1.01%, 0.69%, 0.45%, 0.53%, 6.67%, and 0.67% of their original networks. These values indicate that the two objectives of BBDF are well satisfied. Moreover, optimal thresholds for the six systems are 0.12, 0.1, 0.1, 0.08, 0.2, and 0.24, respectively.

TABLE V OPTIMAL DIVISION PATTERNS

System		2	3	4	5	6	7
Number of nodes	Pattern	8+1	9+1	9+1	11+1	4+1	4+1
	Coordination network	168	336	326	301	5	44
	The largest subnetwork	292	280	292	257	7	65
	The smallest subnetwork	268	261	279	241	5	63
Optimal objective value of (12)		12.26	10.77	7.31	7.81	1.01	9.27

B.3 Acceleration Performance of Individual Systems

Fig. 5 shows separate speedup ratios of the three key steps via the proposed parallel computation against corresponding sequential calculations. Due to space limitation, we take nodal power mismatch vector for the detailed discussion. Fig. 5 indicates that speedup ratios respectively are 11.05, 10.63, 10.95, 12.33, and 12.26 for systems 1-5, which correspond to 90.95%, 90.59%, 90.87%, 91.89%, and 91.85% reduction in time as compared to their sequential correspondences. This clearly shows significance and necessity in conducting parallel formation of nodal power mismatch vectors, similar as the other two major steps.

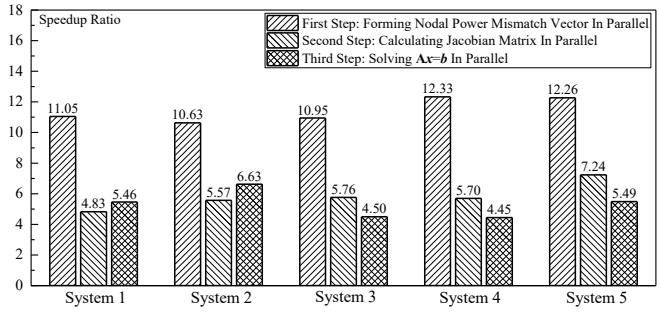

Fig. 5 Separate speedup ratios of the three key steps for systems.

Fig. 5 also reveals that for all five systems, parallel implementations of nodal power mismatch vector present the highest speedup ratio, while the other two are at similar level. Here, we use the 3012wp system for a detailed discussion. Specifically, for the 3012wp system with the optimal network segmentation scheme of "11+1", 12/12 threads are executed to form nodal active/reactive power mismatch vectors, 11/11/11/1 threads are built to calculate $\mathbf{J}_{ww}/\mathbf{J}_{wt}/\mathbf{J}_{tw}/\mathbf{J}_{tt}$, and 11/11 threads are created to compute state vectors of the coordination network/individual subnetworks. In this case, as 34 threads are deployed for calculating Jacobian matrices in parallel while only 11 threads are used for solving $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$, we may expect that speedup ratio of calculating Jacobian matrices is the highest. However, because of additional communication overhead for transferring data between the main thread and individual sub-threads [27], magnitutes of speedup ratios of the three key steps are significantly different. Maximum data exchange between main thread and corresponding sub-threads in each main PF calculation step is listed in Table VI, where data are expressed in double-precision floating-point format. As shown in Table VI, maximum data exchange for calculating nodal active power mismatch vectors is far lower than the other two steps (i.e., 2,400 bytes for nodal active power mismatch vector versus 2,163,200 bytes for Jacobian matrices and 5,953,088 for $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$). In this case, although the total number of threads for forming nodal power mismatch vectors is 24, less than that used for calculating Jacobian matrices, acceleration performance is higher due to its relatively smaller data exchange, three orders of magnitude lower than those of the other two steps.

TABLE VI MAXIMUM DATA EXCHANGE AMONG THREATS IN 3012WP SYSTEM

Main PF Calculation Steps	Maximum Data Exchange	Bytes
Nodal active power injection	$d\boldsymbol{P}_t$	2,400
Nodal Reactive power injection	$d\boldsymbol{Q}_t$	1,848
Jacobian matrix	$\mathbf{J}_{tt}$	2,163,200
$\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ Subnetwork	$\mathbf{J}_{1010}$,$\mathbf{J}_{10t}$,$d\boldsymbol{S}_{10}$,$d\boldsymbol{X}_t$	3,931,328
$\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$ Cooperate network	$\mathbf{J}_{t10}$,$\mathbf{J}_{1010}$,$\mathbf{J}_{10t}$,$d\boldsymbol{S}_{10}$	5,953,088

Fig. 6 further shows speedup ratios of PF calculations, by comparing cases when parallel computation is implemented on only one of the three key steps versus all three steps. As shown in Fig. 6, implementing parallel calculation on only one step of PF calculation may not achieve good enough computational gains. Taking the 3012wp system as an example, speedup ratio achieved by implementing parallel computing for all three steps is 7.08, compared to 1.23, 1.50, and 1.47 for solely one step. This clearly shows advantage of the proposed fully parallel PF calculation approach over conventional parallel PF studies [12]-[13], which have neglected potential benefits by further conducting parallel calculation on nodal power mismatch vectors and Jacobian matrices. Moreover, although Fig. 5 shows that forming nodal power mismatch vectors has the highest speedup ratio among the three steps, its contribution to boosting up the entire PF parallel performance is rather limited, i.e., 1.23 versus 1.50 for Jacobian matrices and 1.47 for $\mathbf{A}\boldsymbol{x}=\boldsymbol{b}$. The reason is that forming nodal power mismatch vectors only takes about 20% of the total PF calculation time.

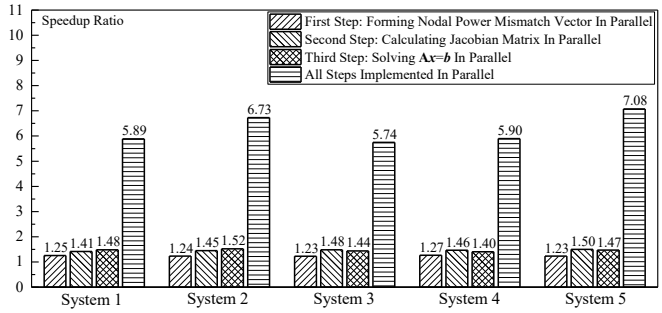

Fig. 6 The whole speedup ratios of each portion implemented in parallel.

B.4 Tradeoff Between Sequential and Proposed Approaches

This section details the tradeoff between sequential and the proposed parallel PF approaches. Fig. 7 shows total computational time of the two approaches on various systems. It indicates that the proposed approach is more computationally efficient for larger systems. However, for small-scale systems like the 30-bus system, time consumed by the proposed approach is 44ms, about 4 times larger than the sequential one. The reason is that although parallel implementation improves calculation performance, it also introduces more time to create/ destroy threads, which potentially limits theoretical accelerating advantage of the proposed approach, especially for small-scale systems.

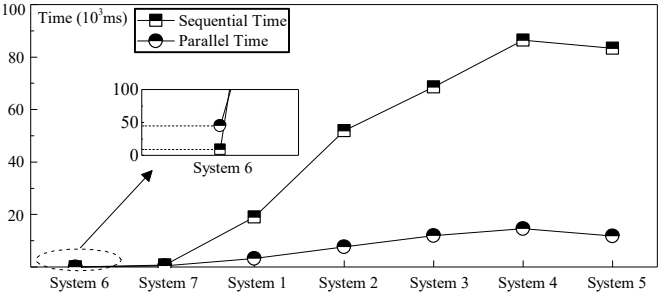

Fig.7 Comparison between sequential and the proposed PF approaches.

Our extensive tests show that average time to create one threat via TPL is about 3ms. Thus, total time of creating all parallel threads to compute the 30-bus system is about 43ms, much higher than total commutation time of 9ms for the sequential approach. In comparison, the proposed approach stands out for large-scale systems, namely, speedup ratios of the other six systems are 1.72, 5.89, 6.73, 5.74, 5.90, and 7.08.

Thus, the proposed parallel approach would be more beneficial, especially for systems of extremely larger scales.

B.5 Discussion on Speedup Ratios and Parallel Efficiency

To further test overall performance of the proposed fully parallel PF approach, relationship between speedup ratio/ parallel efficiency and number of processors is explored. Parallel efficiency [3] is calculated as sequential calculation time over the product of computation time consumed in parallel and the total number of involved processors.

Fig. 8 shows the relationship between speedup ratio/parallel efficiency and number of processors for the 2383WP system, in which CSM-NBBDF is considered as the base case. All parallel computational time reported in Fig. 8 is average time of 100 tests to mitigate side effects of other potential tasks running on the same computer. Fig. 8 shows that speedup ratio gradually approaches to its saturation state of 6.73, when the number of processors exceeds 25. That is, when the number of processers is larger than that of maximum parallel tasks (which is 25 from Table V), acceleration effects becomes saturate. Fig. 8 also shows that parallel efficiency presents a decreasing trend against number of processors. The reason is that, with a larger number of processors, workload processed by individual processors is smaller and consequently parallel efficiency will gradually decrease.

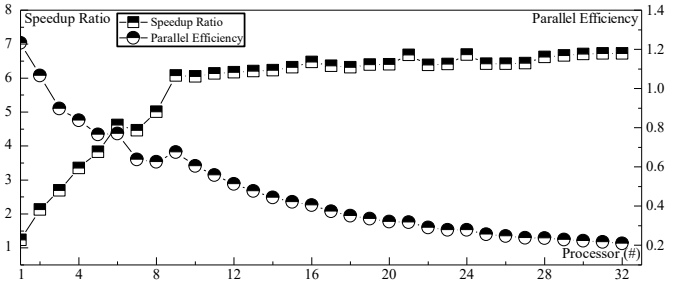

Fig. 8 Relationship between speedup ratio/parallel efficiency and number of processors for 2383WP system.

V. CONCLUSION

This paper proposes a fine-grained fully parallel PF approach to enhance the computational performance, by incorporating node-tearing based BBDF into NR method while also implementing three major PF calculation steps in parallel. Extensive simulation results show that:

(i) The proposed network segmentation method can derive proper network partition schemes with relatively equally-sized subnetworks and a small-sized coordination network, which would help enhance performance of parallel PF calculation;

(ii) The idea of BBDF on parallelly calculating subnetworks is further extended to compute state vector of the coordination network in parallel, which is of significant benefit in improving acceleration performance;

(iii) Implementing parallel computation on all three key steps of PF calculation could achieve the best computational gains.

In summary, this work demonstrates significant computational benefits of the proposed fine-grained fully parallel PF approach. If embedded into other power system applications such as static security assessment that require repeated PF computations, a significant computational improvement could be expected. Future work will target on further improving computational performance of the network segmentation approach, and on integrating sparsity approach, vectorization parallelization [7], and GPU technique to further accelerate parallel PF calculation.

REFERENCES

[1] V. Roberge, M. Tarbouchi, and F. Okou, "Parallel power flow on graphics processing units for concurrent evaluation of many networks," *IEEE Trans. Smart Grid*, vol. 8, no. 4, pp. 1639-1648, Jul. 2017.

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

[3] D. Chen, H. Jiang, Y. Li, and D. Xu, "A two-layered parallel static security assessment for large-scale grids based on GPU," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1396-1405, May 2017.

[4] X. Li, F. Li, H. Yuan, H. Cui, and Q. Hu, "GPU-based fast decoupled power flow with preconditioned iterative solver and inexact Newton method," *IEEE Trans. Power Syst.*, vol. 32, no. 4, pp. 2695-2703, Jul. 2017.

[5] L. Ao, B. Cheng, and F. Li, "Research of power flow parallel computing based on MPI and PQ decomposition method," in *2010 International Conference on Electrical and Control Engineering (ICECE)*, pp. 2925-2928, 2010.

[6] Y. Li, F. Li, and W. Li, "Parallel power flow calculation based on multi-port inversed matrix method," in *2010 International Conference on Power System Technology Parallel*, pp. 1-6, 2010.

[7] B. Zhang and S. Chen, Advanced Power System Network Analysis. Tsinghua University Press, Beijing, 1996.

[8] B. Zhang, N. Xiang, and S. Wang, "Unified piecewise solution of power-system networks combining both branch cutting and node tearing," *Int. J. Electr. Power Energy Syst.*, vol. 11, no. 4, pp. 283-288, Oct. 1989.

[9] R. Green, L. Wang, and M. Alam, "Applications and trends of high performance computing for electric power systems: Focusing on smart grid," *IEEE Trans. Smart Grid*, vol. 4, no. 2, pp. 922-931, Jun. 2013.

[10] T. Yang, W. Xiang, H. Wang, and H. Pen, "An algorithm for solving the block bordered diagonal form of electrical power system in data center," *Proc. CSEE*, vol. 35, no. 3, pp. 512-518, Feb. 2015.

[11] NVIDIA, CUDA Toolkit Document 2017. [Online]. Available: http://docs.nvidia.com/cuda/incomplete-lu-cholesky/index.html.

[12] L. Wan, Y. Chen, and J. Chen, "Node migration based optimized network partitioning strategy for power system parallel computation," *Power Syst. Technol.*, vol. 31, no. 11, pp. 42-48, Jun. 2007.

[13] L. Wan and Y. Chen, "Stabilized border matrix newton parallel load flow method," *High Volt. Eng.*, vol. 33, no. 4, pp. 106-109, Apr. 2007.

[14] N. Garcia, "Parallel power flow solutions using a biconjugate gradient algorithm and a Newton method: A GPU-based approach," *Power Energy Soc. Gen. Meet. 2010 IEEE*, pp. 25-29, Jul. 2010.

[15] Y. Saad, Parallel Iterative Methods for Sparse Linear Systems *2nd ed*. Philadelphia, PA, USA: SIAM, 2001.

[16] J. Shu, W. Xue, and W. Zheng, "A parallel transient stability simulation for power systems," *IEEE Trans. Power Syst.*, vol. 20, no. 4, pp. 1709-1717, Nov. 2005.

[17] R. Zimmerman, C. Murillo-Sánchez, and R. 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.

[18] C. Murillo-Sanchez, R. Zimmerman, C. Anderson, and R. Thomas, "Secure planning and operations of systems with stochastic sources, energy storage, and active demand," *IEEE Trans. Smart Grid*, vol. 4, no. 4, pp. 2220-2229, Dec. 2013.

[19] X. Wang, X. Li, and G. Chen, Complex Network Theory and Applications. Beijing, Tsinghua University Press, 2006.

[20] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi, "Defining and identifying communities in networks," *Prob. Natl. Acad. Sci*, vol. 101, pp. 2658-2663, 2004.

[21] T. Cormen, C. Leiserson, R. Riverst, and C. Stein, Introduction to Algorithms. MIT Press, 2001.

[22] J. Clausen, "Branch and bound algorithms-principles and examples," *Dep. Comput. Sci. Univ.*, pp. 1-30, 1999.

[23] W. Dong and M. Zhou, "A supervised learning and control method to improve particle swarm optimization algorithms," *IEEE Trans. Syst. Man, Cybern. Syst.*, vol. 47, no. 7, pp. 1135-1148, Jul. 2017.

[24] F. Gao,Intelligent Algorithm Super Learning Manual. Posts&Telecom Press, Beijing, 2014.

[25] http://people.clarkson.edu/~lwu/data/PF/Supp_Material_Parallel_PF.pdf.
[26] A.S.Vincentelli, L.O. Chua, and L.K. Chen, "An efficient heuristic cluster algorithm for tearing large-scale networks," *IEEE Trans. Circuits Syst.*, vol. 24, no. 12, pp. 709-717, Dec. 1977.
[27] Microsoft .Net. Managed Threading, 2017. [Online]. Available: https://docs.microsoft.com/zh-cn/dotnet/opbuildpdf/standard/threading/toc.pdf?branch=live.

BIOGRAPHIES

Xueneng Su (S'17) received the B.S. degree in electrical engineering from Sichuan University, Chengdu, China in 2014, where he is currently working toward the Ph.D. degree.

He has been a visiting Ph.D. student at Clarkson University, Potsdam, NY, USA since 2017. His research interests include parallel computing and its application on power systems.

Tianqi Liu (SM'16) received the B.S. and the M.S. degrees from Sichuan University, Chengdu, China, in 1982 and 1986, respectively, and the Ph.D. degree from Chongqing University, Chongqing, China, in 1996, all are in Electrical Engineering. Currently, she is a professor in school of electrical engineering and information at Sichuan University. Her main research interests are power system analysis and stability control, HVDC, optimal operation, dynamic security analysis, dynamic State Estimation and load forecast.

Lei Wu (SM'13) received the B.S. degree in electrical engineering and the M.S. degree in systems engineering from Xi'an Jiaotong University, Xi'an, China, in 2001 and 2004, respectively, and the Ph.D. degree in electrical engineering from Illinois Institute of Technology (IIT), Chicago, IL, USA, in 2008. From 2008 to 2010, he was a Senior Research Associate with the Robert W. Galvin Center for Electricity Innovation, IIT. He worked as summer Visiting Faculty at NYISO in 2012. Currently, he is an Associate Professor with the Electrical and Computer Engineering Department, Clarkson University, Potsdam, NY, USA. His research interests include power systems operation and planning, energy economics, and community resilience microgrid.