An implementation of parallel power flow calculation based on graph partitioning algorithm

Wenkai Zhao[1], Xinyan Fang[1], Xu Dong[1], Yuan Bi[2] and Ying Wu[3]
(1) Department of Electrical Engineering
Shanghai Jiaotong University
No.800, Dongchuan Road, Minhang District, Shanghai 200240
(2) East China Grid Company Limited
No.201, East Nanjing Road, Huangpu District, Shanghai 200002
(3) Electrical Energy Measurement Center
Shanghai Municipal Electric Power Company
No.1758, Tianshan Road, Changning District, Shanghai 200051
China
zwk602@sjtu.edu.cn http://www.sjtu.edu.cn

Abstract: - In coarse-grained parallelism, it is effective to partition the network into Bordered Block Diagonal Form (BBDF) before subsequent parallel computation. An implementation of parallel power flow calculation based on a novel graph partitioning algorithm, which transforms the admittance matrix into nested BBDF (NBBDF) , is presented in this paper. In order to avoid excessive fill-ins during Gaussian elimination, a vertex ordering scheme is discussed. Distributed file storage combined with task scheduling is proposed for improving parallel efficiency. Testing results for grids with up to 5317 buses indicate that this proposed method is able to bring superlinearity into parallel power flow calculation for large-scale power systems.

Key-Words: - Parallel power flow calculation, power systems, graph partitioning algorithm.

1 Introduction

With the advent of smart grid, a wide variety of applications such as network restructuring, transient stability analysis, state estimation, etc, are in urgent need of the powerful computing capacity for on-line monitoring and control in power systems. Thus, fast power flow calculation is essential for large-scale power systems with continually increasing amount of bus bars on the basis of these functions.

In conjunction with the development of high-performance computing environment, researches on parallel computation in power systems mostly focus on graph partitioning algorithms whose aim to find a network made up of several even-sized subdivisions connected by few tie-lines.

Graph partitioning algorithms originate from the concept of Diakoptics, proposed by Gabriel Kron in 1963 [1]. It involves breaking a problem down into some subproblems that can be solved independently before being joined back together to attain a solution to the whole problem. This sort of graph partitioning has 2 criteria as:

1) Equal-sized partitions;
2) Few connections among partitions.

According to the partitions obtained, the system matrix is rearranged by way of taking each partition as a diagonal block and nodes adjacent to different partitions as a boundary block. Thereby, the system matrix has a structure of Bordered Block Diagonal Form (BBDF).

Contour Tableau, prevailing in the 1970s, serves to build BBDF system matrix through searching for independent clusters coupled with bottleneck nodes. A heuristic algorithm based on Contour Tableau succeeds in transforming the system matrix into BBDF [2]. However, its optimality in partitioning is conditional upon the selection of each seed node. Meanwhile, nodes of boundary block are not as few as expected. Sequential Binary Partition (SBP) uses a pseudo optimal ordering scheme after partitioning for preserving the sparsity of BBDF matrix during Gaussian elimination [3]. Nevertheless, there are no good answers to the above two questions.

Nested Dissection, ancestor of multilevel graph partitioning, has achieved high performance in parallel electronic circuit simulation [4]. Albeit it is superior to Contour Tableau in respect of balanced partition size, the number of partitions is fixed to the power of 2. Node merging method contracts vertices to establish partitions, similar to the coarsening stage in multilevel graph partitioning [5]. Relations between starting nodes and their growing clusters

place uncertainties in partitioning results like those of Contour Tableau based algorithms.

K. W. Chan has proposed a factorization path tree based partitioning algorithm, which restructures the original path tree ahead of grouping branches [6]. This algorithm not only completes construction of BBDF matrix, the post-partitioning ordering keeps the number of fill-ins in the later elimination to a minimum as well [7]. The pre-partitioning ordering, which helps build a short and broad path tree before partitioning, raises a question that partitions may be not as good as anticipated without a well-organized initial path tree.

Given that the partitioning time that most parallel computations do not concern, if it takes long time to partition the original network, the performance of algorithm will be pulled down totally no matter how effective the parallel computation is. To solve this problem, multilevel graph partitioning is employed, which operates within three phases as coarsening, partitioning and refining [8]. It is the partitioning phase, where the original graph has been contracted significantly, that shortens the entire execution time. Multilevel recursive bisection and multilevel *k*-way partitioning satisfy criteria of graph partitioning and operate in a short time [9]. Moreover, the refinement with KL algorithm has contributed to fewer cut edges between different partitions [10].

It is evident that increasing partition number will enlarge the size of boundary block. Large boundary block will hinder accelerating parallel computation, because elimination of boundary block is the only serial portion of parallel Gaussian elimination. One way to avoid this situation is to partition the original network recursively so as to obtain nested BBDF (NBBDF) system matrix [11-13]. Unlike BBDF matrix, task scheduling should be included in the parallel computation of NBBDF matrix in that assigning each block to an exclusive processor will cost more processors than actually needed, followed by possible low efficiency in parallelism.

A novel implementation of parallel power flow calculation, based on a graph partitioning algorithm which transforms the system matrix into NBBDF by edge cut sets in a recursive way, is presented in this paper. Different from common NBBDF partitioning algorithms, the only relationship between blocks of contiguous levels instead of any two levels reduces data dependencies and communication overheads. In view of the possible computational burden due to excessive fill-ins in the matrix Gaussian elimination, a sub-optimal vertex ordering scheme is employed after partitioning. Additionally, both distributed file storage and task scheduling are incorporated into the implementation for reducing initialization time and improving parallel efficiency. Experimental results for grids of up to 5317 buses demonstrate that this proposed approach is able to bring superlinearity into parallel power flow computation for large-scale power systems.

In next section, the general process for coarse-grained parallel computation is briefly described as well as principles underlying in a graph contraction associated with the matrix Gaussian elimination. In section 3, basic data structures used for NBBDF partitioning with its implementation are interpreted in detail. A vertex ordering scheme and distributed file storage are illustrated in section 4. Section 5 discusses task scheduling with parallel Gaussian elimination. Analysis to testing results is presented in section 6. Conclusions are followed in section 7.

2 Parallel computing in power system

According to parallel transient stability analysis and distributed state estimation [14-16], it is found that the ordinary process for parallel computation in power systems comprises four stages as portrayed in figure 1.

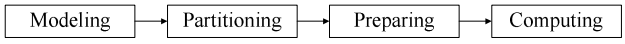

Fig.1 Ordinary process of parallel computation in power systems

- Modeling is responsible for establishing an undirected graph representing the electric grid, in which bus bars are indicated by vertices and tie-lines or transformers connecting different buses are represented by edges.
- Partitioning serves to separate the undirected graph into several loosely-coupled equal-sized subgraphs. The most common method is to use vertex coloring, which assigns different colors to vertices subordinating to different subgraphs.
- Preparing deals with data file deployment and vertex ordering. Data file deployment can help reduce initialization overheads before parallel computation while vertex ordering is used for preserving the sparsity of partitioned matrix.
- Computing is engaged in task assignment along with parallel computation of partitioned matrix. Task assignment attaches great importance to a shared memory parallel architecture. Whatever, it also full utilizes the processors available in a distributed memory parallel architecture.

The solution of network equations in transient stability analysis and that of least square in state estimation have an identical formulation of solving sparse system of equations given by equation (1).

$$\boldsymbol{Ax} = \boldsymbol{y} \quad (1)$$

$\boldsymbol{A}$ is a structurally symmetric matrix, $\boldsymbol{x}$ and $\boldsymbol{y}$ are n-dimensional vectors.

The corresponding undirected graph $G(V, E)$, in which V and E denote vertex set and edge set respectively, can be described as follows [3]:

1) Each vertex $\boldsymbol{v}_i$ in V maps both x_i in $\boldsymbol{x}$ and y_i in $\boldsymbol{y}$ in a bijective way;

2) Each edge $\boldsymbol{e}_{ij} = (\boldsymbol{v}_i, \boldsymbol{v}_j)$ in E represents the non-zero element a_{ij} in $\boldsymbol{A}$, where $j > i$.

The mapping from the system of equations to its associated undigraph can be found in figure 2.

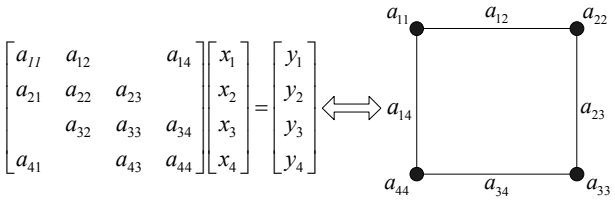

Fig.2 Mapping from system of equations to an undigraph

The coefficient matrix $\boldsymbol{A}$ is always referred to as the adjacent matrix of graph.

When matrix $\boldsymbol{A}$ is processed by a direct method such as Gaussian elimination, its undirected graph contraction conforms to the following lemma.

Lemma 1: During graph contraction associated with structurally symmetric matrix Gaussian elimination, contracting vertex $\boldsymbol{v}_k$ makes all vertices adjacent to $\boldsymbol{v}_k$ constitute a complete graph.

For instance, in figure 2, eliminating pivot a_{11} of matrix means contracting vertex a_{11} in the undigraph, which causes a_{22} and a_{44} contiguous as shown in figure 3.

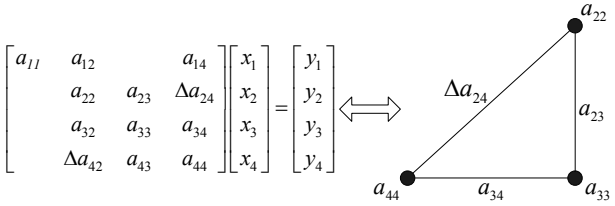

Fig.3 Contracting vertex a_{11} in the undirected graph

3 Graph partitioning

The goal of effective partitioning is to divide the original undigraph into several subgraphs satisfying 2 criteria of graph partitioning discussed in section 1. Neither multilevel graph partitioning nor geographic information based partitioning is able to acquire the partitioned graph with BBD pattern directly, thus a post-process should be added to partitioning before parallel computation.

3.1 BBDF partitioning by an edge cut set

Before illustrating BBDF partitioning by an edge cut set, a definition and a theorem are put forward respectively in the following paragraphs.

Definition 1: In graph $G(V, E)$, P and Q are non-empty vertex sets, containing no same vertex. The edge set $[P, Q]_G$ comprises all edges with one vertex belonging to P and the other belonging to Q. Specially, if $P + Q = V$, namely $Q = \overline{P}$, the edge set $\left[P, \overline{P}\right]_G$ is called an edge cut of graph $G(V, E)$.

Theorem 1: In the connected graph $G(V, E)$, there are n connected subgraphs $G_i(V_i, E_i)$ satisfying $V = \bigcup_{i=1}^{n} V_i$ and $\forall V_i \bigcap V_j = \phi (i \neq j)$. The edge-induced subgraph $G_{\text{boundary}}(V_{\text{boundary}}, \Im)$, whose edge set $\Im$ is the union of edge cuts $\xi_i = [V_i, \overline{V_i}]_G$ in terms of $\Im = \bigcup_{i=1}^{n} \xi_i$, separates graph $G(V, E)$ into n decoupled subgraphs $G_i^{'}(V_i^{'}, E_i^{'})$, in which $V_i^{'} = V_i - V_{\text{boundary}}$. All these subgraphs $G_i^{'}(V_i^{'}, E_i^{'})$ are only connected to the subgraph $G_{\text{boundary}}(V_{\text{boundary}}, \Im)$ as a result.

After converting the original graph to an NBDF one by some kind of partitioning method [17], this NBDF graph can be converted into a BBDF graph according to theorem 1. For instance, the case in figure 4 illustrates such conversion.

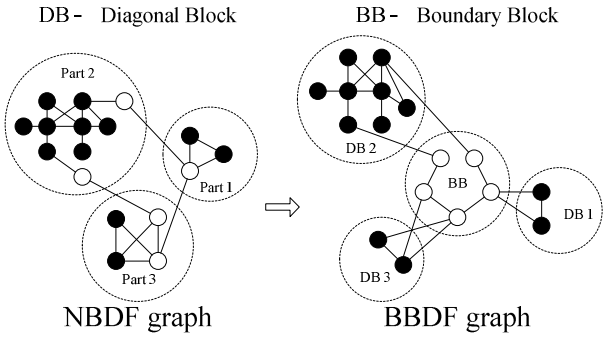

Fig.4 BBDF partitioning by an edge cut set

In figure 4, the original graph was partitioned into 3 parts to begin with. White vertices in NBDF graph are ones that belong to the boundary block of BBDF graph.

The system matrix is then reordered by arranging diagonal blocks in diagonal and a boundary block in the lower right corner to obtain a structure of BBDF.

If the boundary block of this BBDF matrix is reordered by grouping those vertices connected to an identical diagonal block together successively, it is only a portion of boundary block not all that connects to a diagonal block and one such portion has only one corresponding diagonal block. This property helps decrease communication overheads and prevents conflicting operands on the same element of the boundary block in parallel Gaussian elimination.

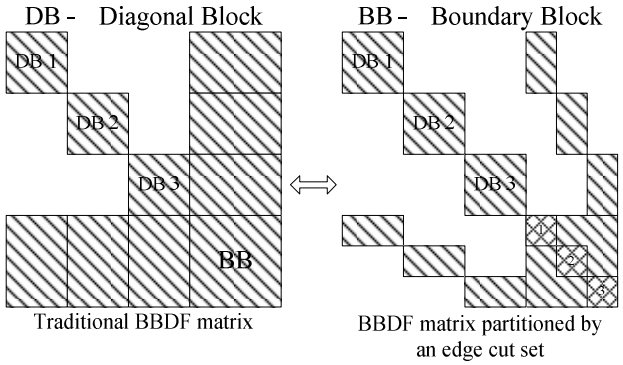

Fig.5 Traditional BBDF matrix versus BBDF matrix partitioned by an edge cut set

The differences between the traditional BBDF matrix and the BBDF matrix partitioned by an edge cut set can be found in figure 5.

3.2 Basic data structures

With the NBDF graph, basic data structures, used to implement NBBDF partitioning, are defined in the following tables. However, only abstract data types are stated here in that the specific realization depends on either the programming language or the development environment or both.

Table 1 Vertex prototype

Property	Type	Remarks
ID	Integer	Unique identifier
Level	Integer	Nested level of the block to which this vertex belongs
Colors	Integer array	Colors of the partition, from which this vertex springs, of all levels
AdjVs	Vertex set	Vertices adjacent to this vertex
BusInfo	User-defined	Bus information

Table 2 Vertex set prototype

Property	Type	Remarks
Entry	Vertex	One vertex in this set

Table 3 Edge prototype

Property	Type	Remarks
ID	Integer	Unique identifier
V1	Vertex	One incident vertex
V2	Vertex	The other incident vertex
BranchInfo	User-defined	Tie-line or transformer information

Table 4 Edge set prototype

Property	Type	Remarks
Entry	Edge	One edge in this set

In order to give a detailed explanation for such data structures, the case in figure 4 is taken into account. The original graph has been converted into a 1-nested BBDF graph, in which property Colors of each vertex is a one-dimensional array with one integer indicating color of the partition from which this vertex came. Property Level of vertices in the diagonal block is 1 while that of vertices in the boundary block is 0. If a vertex in the BBDF graph has Colors of {3} (symbol {} denotes an array) and Level of 0, it means that this vertex is located in the boundary block and connected to diagonal block 3.

3.3 NBBDF partitioning by edge cut sets

Advances in a wide variety of parallel issues in power systems have proven the validity of primitive BBDF partitioning [6, 7, and 14-18]. Whatever, new challenges arise from the rapid growing electric grid. On one hand, more partitions will bring more edges into the edge cut set, followed by a larger boundary block. On the other hand, computation overheads in each diagonal block are considerable due to fewer partitions. As a result, partitioning the network in a recursive way to gain a system matrix with NBBDF is a viable method.

If NBBDF partitioning is just a recursive way of the primitive BBDF partitioning as stated in [18] and [19], it embraces an underlying risk in itself as the boundary block of level k is connected to all boundary blocks of level k-1 to level 0. In other words, a boundary block has to communicate with boundary blocks of all lower levels in the parallel execution, which introduces great communications volume and renders parallel programming complex. Besides, synchronization is another problem needed concern.

A NBBDF partitioning method, which is derived from the BBDF partitioning by an edge cut set, is proposed to solve these problems. The distinction between this method and conventional ones is that boundary blocks of level k are connected to those of

level *k*-1 and *k*+1 only. This character can be found in the associated NBBDF matrix shown in figure 6.

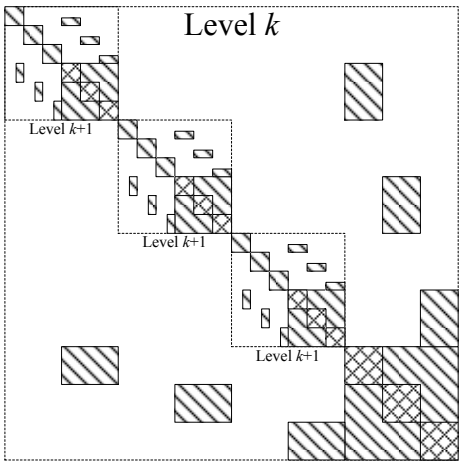

Fig.6 NBBDF matrix partitioned by edge cut sets

The recursive procedure of NBBDF partitioning by edge cut sets is described as follows.

Firstly, partitioning the diagonal block of level 0, which is a NBDF graph actually, into a BBDF graph partitioned by an edge cut set. And, the number of diagonal blocks is $n_{k,p}$, where $k=0$ and $p=1$.

Secondly, assuming the original graph has been divided into *k* nested levels. Then, the *p*th diagonal block of level *k*-1 is partitioned into $n_{k,p}$ diagonal blocks and one boundary block $G_{k,p}^{\mathrm{bb}}(V_{k,p}^{\mathrm{bb}},E_{k,p}^{\mathrm{bb}})$. Meanwhile, there are $n_{k+1,i}$ parts, which do not contain any common vertex, in each diagonal block $G_{k,p}^{\mathrm{db}i}(V_{k,p}^{\mathrm{db}i},E_{k,p}^{\mathrm{db}i})$ $(i=1,2,\cdots,n_{k,p})$.

Upon that, the NBBDF partitioning algorithm is operated in a recursive way as explained below.

1) Combining vertex set $V_{k,p}^{\mathrm{db}i}$ of the *i*th diagonal block $G_{k,p}^{\mathrm{db}i}(V_{k,p}^{\mathrm{db}i},E_{k,p}^{\mathrm{db}i})$ with vertex set $V_{k,p}^{\mathrm{bb}}$ of the boundary block $G_{k,p}^{\mathrm{bb}}(V_{k,p}^{\mathrm{bb}},E_{k,p}^{\mathrm{bb}})$ to form an induced subgraph $G_{k,p}^{i}(V_{k,p}^{i},E_{k,p}^{i})$, where $V_{k,p}^{i}=V_{k,p}^{\mathrm{db}i}\bigcup V_{k,p}^{\mathrm{bb}}$;

2) Known by the assumption, there are $n_{k+1,i}+1$ parts containing different vertices in the induced subgraph $G_{k,p}^{i}(V_{k,p}^{i},E_{k,p}^{i})$ for $V_{k,p}^{\mathrm{db}i}\bigcap V_{k,p}^{\mathrm{bb}}=\phi$. By way of theorem 1, partitioning subgraph $G_{k,p}^{i}(V_{k,p}^{i},E_{k,p}^{i})$ into $n_{k+1,i}+1$ decoupled subgraphs $G_{j}^{'}(V_{j}^{'},E_{j}^{'})$ and one edge-induced subgraph $G_{\mathrm{bb}}(V_{\mathrm{bb}},\Im)$;

3) Excluding a subgraph, which is induced by $V_{k,p}^{\mathrm{bb}}$ of boundary block $G_{k,p}^{\mathrm{bb}}(V_{k,p}^{\mathrm{bb}},E_{k,p}^{\mathrm{bb}})$, from these $n_{k+1,i}+1$ decoupled subgraphs $G_{j}^{'}(V_{j}^{'},E_{j}^{'})$, then the rest $n_{k+1,i}$ decoupled subgraphs $G_{j}^{'}(V_{j}^{'},E_{j}^{'})$ are level *k*+1 diagonal blocks $G_{k+1,i}^{\mathrm{db}j}(V_{k+1,i}^{\mathrm{db}j},E_{k+1,i}^{\mathrm{db}j})$ of diagonal block $G_{k,p}^{\mathrm{db}i}(V_{k,p}^{\mathrm{db}i},E_{k,p}^{\mathrm{db}i})$, where $j=1,2,\cdots,n_{k+1,i}$;

4) Excluding vertices, which belong to boundary block $G_{k,p}^{\mathrm{bb}}(V_{k,p}^{\mathrm{bb}},E_{k,p}^{\mathrm{bb}})$, from edge-induced subgraph $G_{\mathrm{bb}}(V_{\mathrm{bb}},\Im)$, then the rest vertex-induced subgraph is the level *k*+1 boundary block $G_{k+1,i}^{\mathrm{bb}}(V_{k+1,i}^{\mathrm{bb}},E_{k+1,i}^{\mathrm{bb}})$ of the *i*th diagonal block $G_{k,p}^{\mathrm{db}i}(V_{k,p}^{\mathrm{db}i},E_{k,p}^{\mathrm{db}i})$ satisfying $V_{k+1,i}^{\mathrm{bb}}=\left\{v\middle|v\in V_{\mathrm{bb}}\ \&\ v\notin V_{k,p}^{\mathrm{bb}}\right\}$.

After the original graph has been partitioned into a 1-nested BBDF graph in figure 4, which includes $n_{0,1}=3$ diagonal blocks, diagonal block 2 serves as $G_{0,1}^{\mathrm{db}2}(V_{0,1}^{\mathrm{db}2},E_{0,1}^{\mathrm{db}2})$, where there are $n_{1,2}=2$ parts with no common vertex. Utilizing the above method to partition diagonal block 2 of the BBDF graph in figure 4, the process is demonstrated in figure 7.

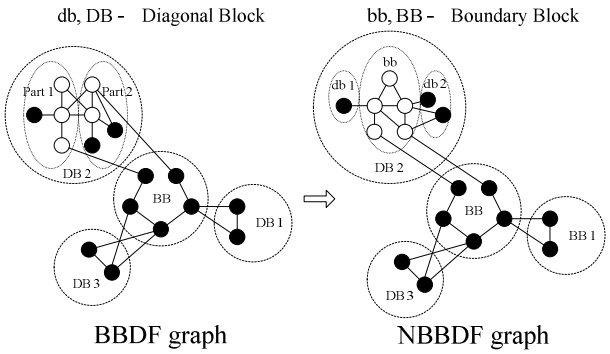

Fig.7 NBBDF partitioning by edge cut sets

First of all, combining DB 2 with BB to get the induced subgraph $G_{0,1}^{2}(V_{0,1}^{2},E_{0,1}^{2})$. Next, according to theorem 1, partitioning subgraph $G_{0,1}^{2}(V_{0,1}^{2},E_{0,1}^{2})$ into $n_{1,2}+1=3$ decoupled subgraphs $G_{j}^{'}(V_{j}^{'},E_{j}^{'})$ and one edge-induced subgraph $G_{\mathrm{bb}}(V_{\mathrm{bb}},\Im)$. Vertices of the 3 decoupled subgraphs $G_{j}^{'}(V_{j}^{'},E_{j}^{'})$ are black nodes in part 1, black nodes in part 2 and the bottom 3 black nodes in BB respectively. Vertices of edge-induced subgraph $G_{\mathrm{bb}}(V_{\mathrm{bb}},\Im)$ are white nodes in DB 2 and the top 2 nodes in BB. Furthermore, excluding subgraph $G_{j}^{'}(V_{j}^{'},E_{j}^{'})$, which contains 3 black nodes in BB, leaves two diagonal blocks of level 1 as db 1 and db 2. At last, excluding 2 black nodes in BB from edge-induced subgraph $G_{\mathrm{bb}}(V_{\mathrm{bb}},\Im)$ leaves the boundary block bb of level 1.

3.4 Implementation of NBBDF partitioning

NBBDF partitioning is a recursive method from outside to inside, that is, the outermost is level 0 and the innermost is level N if the original graph has taken an N-nested partitioning.

In an NBBDF graph, blocks, whether diagonal or boundary, are referred to as boundary blocks for the reason that a diagonal block of level k is also a boundary block of level k+1.

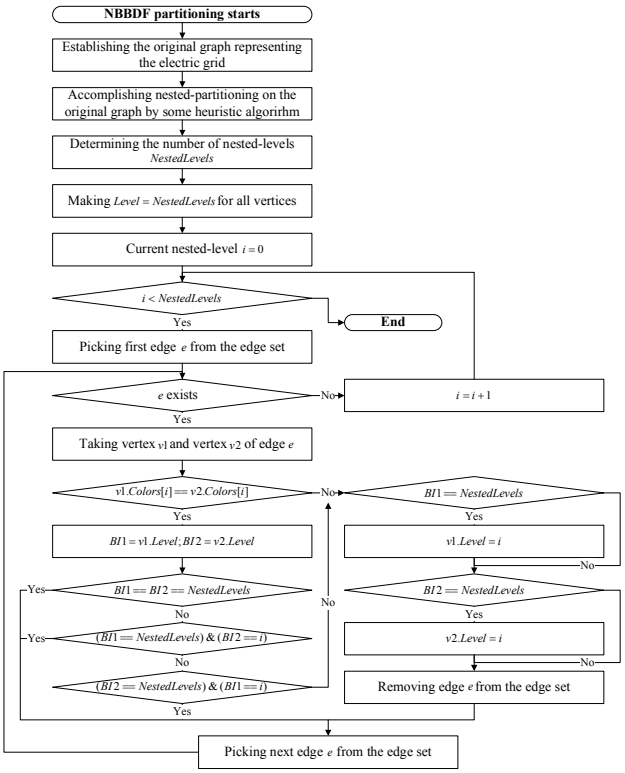

Fig.8 Flowchart of NBBDF partitioning by edge cut sets

The implementation of NBBDF partitioning by edge cut sets is illustrated in detail in figure 8, in which a flowchart involves some fundamental data structures described in section 3.2.

According to this flowchart, it is found that the computational complexity is less than $N*E$, where N is the nested-level number and E is the edge number. Generally, N is not more than 4 for power systems, that is, the complexity of NBBDF partitioning is proportional to the number of branches. Therefore, NBBDF partitioning is comparable with multilevel graph partitioning in speed.

There are 2 characteristics for vertices in the NBBDF graph partitioned by edge cut sets as:

1) In array Colors of a vertex, first Level-1 integers indicate colors of the block this vertex belongs to while first Level integers indicate colors of the block this vertex is connected to.

2) Vertices in the vertex set AdjVs of a vertex, whose property Level equals k, can have property Level of k-1 or k+1 only.

The first one is helpful in the late vertex ordering process, and the second one ensures the hierarchical structure of distributed file storage.

4 Preparing for parallel computing

In accordance with the NBBDF graph obtained, distributed file storage is employed to take on the task of reducing initialization overheads in parallel computation. In order to keep fill-ins of matrix Gaussian elimination to a low level, a sub-optimal vertex ordering scheme is also adopted.

4.1 Vertex ordering of NBBDF graph

Parallel Gaussian elimination of NBBDF matrix is operated on boundary blocks of each level in turn. Thereby, renumbering nodes in each corresponding submatrix is a key point for preserving the sparsity of the entire NBBDF matrix.

Node renumbering for a matrix corresponds to vertex ordering for a graph, which is known as a non-polynomial complete problem [3]. This is the reason why our vertex ordering scheme is called sub-optimal instead of optimal.

Theorem 2: In the connected graph $G(V, E)$, there is a connected subgraph $G_c(V_c, E_c)$. Vertices, which belong to V-V_c and are adjacent to graph $G_c(V_c, E_c)$, constitutes a complete graph after graph $G_c(V_c, E_c)$ is eliminated from graph $G(V, E)$.

Theorem 2 points out that no matter what orders vertices take in a diagonal block, as long as it is a connected diagonal block, vertices adjacent to it are connected with each other in the boundary block after it is eliminated. In other words, extra fill-ins before the process of boundary block elimination are predetermined and have nothing to do with the vertex order of any diagonal block.

In the NBBDF graph partitioned by edge cut sets, boundary blocks of level k are only connected to those of level k-1 and level k+1. To an NBBDF graph with the deepest level of N, its vertex ordering is given below.

1) In a boundary block $G_{k,p}^{\mathrm{bb}}(V_{k,p}^{\mathrm{bb}}, E_{k,p}^{\mathrm{bb}})$ of level k, where $k = 0,1,\cdots,N$, appending extra edges on those vertices connected to the same diagonal block as to make them connected with each other;

2) Transferring to stage 3 when $k = 0$. Otherwise, locating the boundary block $G_{k-1,q}^{\mathrm{bb}}(V_{k-1,q}^{\mathrm{bb}}, E_{k-1,q}^{\mathrm{bb}})$ of level k-1, to which $G_{k,p}^{\mathrm{bb}}(V_{k,p}^{\mathrm{bb}}, E_{k,p}^{\mathrm{bb}})$ is connected.

Combining $V_{k-1,q}^{\mathrm{bb}}$ with $V_{k,p}^{\mathrm{bb}}$ to form a vertex-induced subgraph $G_{k,p}^{\mathrm{ex}}(V_{k,p}^{\mathrm{ex}}, E_{k,p}^{\mathrm{ex}})$;

3) Exploiting the vertex ordering method in [20], called simplified Tinney 3 scheme, to order vertices in subgraph $G_{k,p}^{\mathrm{ex}}(V_{k,p}^{\mathrm{ex}}, E_{k,p}^{\mathrm{ex}})$;

4) Returning to stage 3 unless all vertices, which sprang from $G_{k,p}^{\mathrm{bb}}(V_{k,p}^{\mathrm{bb}}, E_{k,p}^{\mathrm{bb}})$, have been sorted in subgraph $G_{k,p}^{\mathrm{ex}}(V_{k,p}^{\mathrm{ex}}, E_{k,p}^{\mathrm{ex}})$.

Both theorem 2 and the foregoing procedure reveal that a boundary block of level *k* has no need to wait for its descendants to finish their jobs before starting its own vertex ordering. As a result, vertex ordering of the NBBDF graph can be parallelized effortlessly.

4.2 Distributed file storage

The overall performance of parallel power flow calculation is not only subject to parallel computing as Gaussian elimination, however, the formulation of Jacobian matrix should be taken into account because loading data from files, which contains the entire system, is time-consuming, especially when the system is quite large.

Moreover, the NBBDF graph partitioned by edge cut sets has an inherent property as boundary blocks of level *k* are only connected to those of level *k*-1 and *k*+1. Therefore, distributed data files can be deployed in the following way.

For a boundary block of level *k*, there are four data files available as Local, Inner, Outer and BS.

- File Local stores buses and branches of this boundary block;
- File Inner stores buses and branches, which are related to this boundary block, of level *k*+1 boundary blocks;
- File Outer stores buses and branches, which are linked to this boundary block, of level *k*-1 boundary blocks;
- File BS stores slack nodes and corresponding branches connected to this boundary block.

Figure 9 displays the structure of distributed file storage.

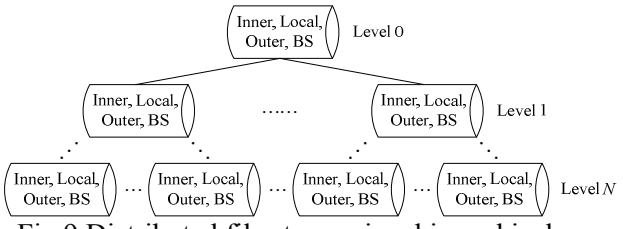

Fig.9 Distributed file storage in a hierarchical way

As the structure of grid is ongoing changing, for example, adding or dropping some branches, several cases are discussed below.

1) Buses (including the new one), which are connected by a branch, resident in an identical boundary block. It is the file Local of this boundary block that needs to be updated only;

2) Buses, which are connected by a branch, resident in different boundary blocks, of which one is level *k* block and the other is level *k*+1 block. Both the file Outer of level *k*+1 block and the file Inner of level *k* block need to be updated;

3) Buses, which are connected by a branch, resident in different boundary blocks that are not adjacent to each other and named level *i* block and level *j* block respectively. These two buses should be put into the first outer level boundary block preceding both level *i* block and level *j* block. And four files of all boundary blocks along 2 paths from the first outer level to level *i* and to level *j* need to be updated simultaneously.

4.3 Mapping tables

One process is assigned to one boundary block to complete initialization and computation. Here, one process doesn't mean one processor since it will take more processors than actually needed.

Every process reads these 4 data files to form 4 corresponding vertex sets as InnerVs, LocalVs, OuterVs and BSVs. Identical with section 3.2, some fundamental data structures, which are defined in the following tables, are used within the processes.

Table 5 Process prototype

Property	Type	Remarks
ParentID	Integer	Unique identifier of its parent process
ChildProcCount	Integer	Number of its child processes
ChildProcIDs	Integer array	IDs of its child processes
TableLFI	Mapping table	Mapping vertices in InnerVs of its parent process to vertices in LocalVs
TableOFL	Mapping table	Mapping vertices in LocalVs of its parent process to vertices in OuterVs
TableLTO	Mapping table	Mapping vertices in LocalVs to vertices in OuterVs of its child processes

Table 6 Mapping table prototype

Property	Type	Remarks
Entry	Mapping table entry	One mapping in this table

Table 7 Mapping table entry prototype

Property	Type	Remarks
ProcID	Integer	Mapping process ID
Value	Integer	Mapping value

For an NBBDF graph partitioned by edge cut sets, three mapping tables as TableLFI, TableOFL and TableLTO in table 5 are explained in figure 10, where dashed frames represent processes and solid frames represent nested levels.

Each process has 3 mapping tables except those corresponding to innermost and outermost boundary blocks.

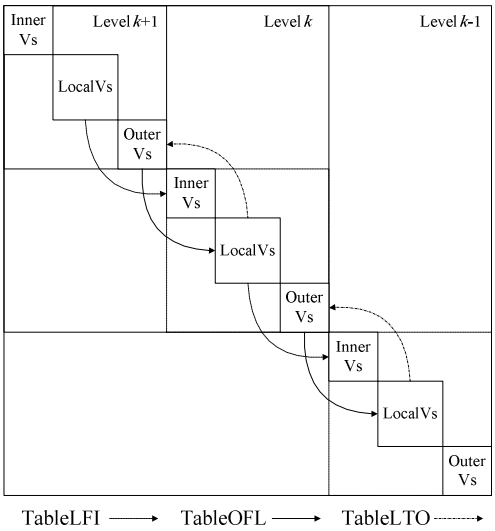

Fig.10 Mapping tables for an NBBDF graph

4.4 Parallel vertex ordering of NBBDF graph

A parallel implementation can be inferred from the procedure of vertex ordering given in section 4.1. Nevertheless, according to theorem 2, considering a situation where there are unconnected portions in a boundary block, in other words, this boundary block is not a connected graph, thus appending extra edges on vertices which are connected to this unconnected boundary block is invalid.

To solve this problem, the implementation of parallel vertex ordering of NBBDF graph, depicted in figure 11, has employed vertex coloring approach. Through vertex coloring method, vertices belonging to different connected components are designated different colors in the vertex set LocalVs of a process. After that, a parent process receives vertex colors from its child processes to update the color information of its own vertex set InnerVs.

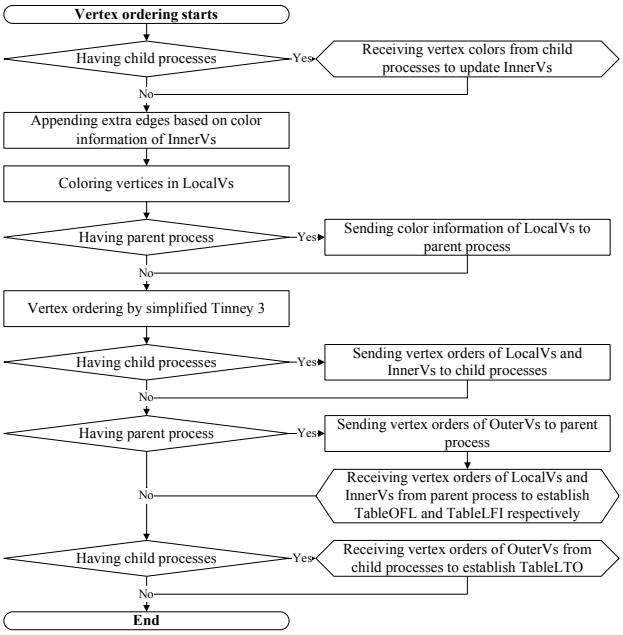

Fig.11 Parallel vertex ordering of NBBDF graph

5 Parallel computing

This section is separated into two parts with respect to parallel power flow calculation for large-scale power systems. For one thing, task scheduling is illustrated for achieving high parallel efficiency. For another thing, attention is paid to the iterative procedure of parallel power flow calculation.

5.1 Task scheduling

Whether in the BBDF or NBBDF graph, it is evident that blocks of different levels are calculated in sequence. To full utilize processors available, computational tasks should be scheduled to increase the overall parallelism degree.

Conventional methods for task scheduling rely on a task graph, which gives precedence relations between tasks [12, 21]. In the NBBDF graph, a task is a running process which reads 4 data files and commits corresponding computation. Thus, the task graph is identical with distributed files in structure as shown in figure 9.

For parallel power flow calculation with an NBBDF system matrix, task scheduling is operated in a bottom-to-up manner as described below.

1) Assigning each bottom task in the task graph to different processors successively;

2) If this level is level 0, then quitting;

3) Searching for a group of tasks having a same parent task which is assigned to no processor yet;

4) Assigning this parent task to the processor which contains the task with heaviest computing loads in this group;

5) If there are such groups of tasks remaining on this level, going back to stage 3. Otherwise, moving up one level along the task graph and returning to stage 2.

In stage 4, a task with heaviest computing loads essentially coincides with a boundary block with most vertices.

The reason why assignes a task together with its heaviest-load child task to the same processor is that smaller boundary block needs less communication during parallel computation. Consequently, this task scheduling scheme can decrease communication overheads to a great extent and improve the entire parallel efficiency of power flow calculation.

5.2 Parallel power flow calculation

In parallel power flow calculation, each process deals with one boundary block of NBBDF graph and reads 4 data files as Inner, Local, Outer and BS to carry out initialization before iterations.

Also, parallel power flow calculation is running in a recursive way. Considering a process dealing with a level *k* boundary block, whose parent process deals with a related boundary block of level *k*-1 and child processes deal with related boundary blocks of level *k*+1, the iterative procedure is stated as follows.

1) Updating Jacobian matrix

Jacobian matrix is formulated via electrical datas in Inner, Local, Outer and BS, which is expressed by equation (2).

$$\begin{bmatrix} \boldsymbol{J}_{11} & \boldsymbol{J}_{12} \\ \boldsymbol{J}_{21} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{y}_1 \\ \boldsymbol{0} \end{bmatrix} \tag{2}$$

Referring to the NBBDF matrix in figure 6, $\boldsymbol{J}_{11}$ is the corresponding matrix of a level *k* boundary block, $\boldsymbol{J}_{12}$ and $\boldsymbol{J}_{21}$ are incidence matrices relating the level *k* boundary block to the level *k*-1 boundary block. Vector $\boldsymbol{x}_1$ records voltage deviations of buses in the level *k* boundary block. Vector $\boldsymbol{x}_2$ records voltage deviations of related buses in the level *k*-1 boundary block. Vector $\boldsymbol{y}_1$ records power deviations of buses in the level *k* boundary block.

2) Forward substitution

It is not until all child processes have completed their forward substitutions that forward substitution of current process can commence. The preparation for forward substitution is displayed in figure 12.

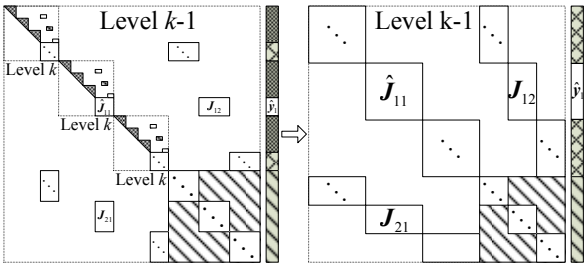

Fig.12 Preparing for forward substitution of current process

Matrix $\Delta\boldsymbol{J}_{11}$ and vector $\Delta\boldsymbol{y}_1$, which are made up of elements received from all child processes, are used to update matrix $\boldsymbol{J}_{11}$ and vector $\boldsymbol{y}_1$ respectively.

$$\begin{aligned} \hat{\boldsymbol{J}}_{11} &= \boldsymbol{J}_{11} + \Delta\boldsymbol{J}_{11} \\ \hat{\boldsymbol{y}}_1 &= \boldsymbol{y}_1 + \Delta\boldsymbol{y}_1 \end{aligned} \tag{3}$$

Applying forward substitution to equation (2), equation (4) is given below.

$$\begin{bmatrix} \boldsymbol{U}_{11} & \boldsymbol{U}_{12} \\ \boldsymbol{0} & \Delta\boldsymbol{J}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_1 \\ \boldsymbol{x}_2 \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_1 \\ \Delta\boldsymbol{y}_2 \end{bmatrix} \tag{4}$$

Equation (4) is based on equation (5) and equation (6).

$$\begin{bmatrix} \boldsymbol{L}_{11} & \boldsymbol{0} \\ \boldsymbol{L}_{21} & \boldsymbol{I}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{U}_{11} & \boldsymbol{U}_{12} \\ \boldsymbol{0} & \Delta\boldsymbol{J}_{22} \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{J}}_{11} & \boldsymbol{J}_{12} \\ \boldsymbol{J}_{21} & \boldsymbol{0} \end{bmatrix} \tag{5}$$

$$\begin{bmatrix} \boldsymbol{L}_{11} & \boldsymbol{0} \\ \boldsymbol{L}_{21} & \boldsymbol{I}_{22} \end{bmatrix} \begin{bmatrix} \boldsymbol{z}_1 \\ \Delta\boldsymbol{y}_2 \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{y}}_1 \\ \boldsymbol{0} \end{bmatrix} \tag{6}$$

Here, $\Delta\boldsymbol{J}_{22} = -\boldsymbol{L}_{21} \times \boldsymbol{U}_{12}$ and $\Delta\boldsymbol{y}_2 = -\boldsymbol{L}_{21} \times \boldsymbol{z}_1$.

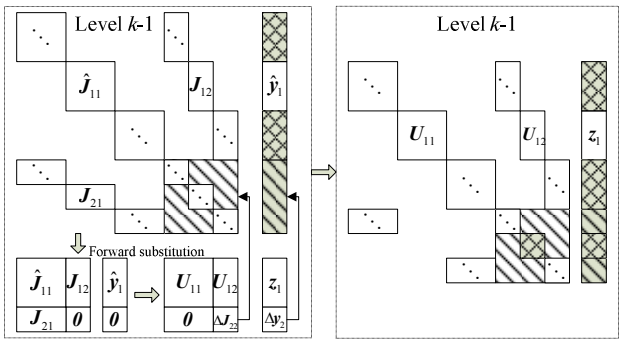

Fig.13 Forward substitution of current process

If there is a parent process, current process has to send elements of $\Delta\boldsymbol{J}_{22}$ and $\Delta\boldsymbol{y}_2$ to its parent process for filling $\Delta\boldsymbol{J}_{11}$ and $\Delta\boldsymbol{y}_1$ respectively. The procedure is displayed in figure 13.

3) Backward substitution

After updating vector $\boldsymbol{x}_2$ by elements received from parent process, current process does backward substitution to equation (4) to solve vector $\boldsymbol{x}_1$.

$$[\boldsymbol{x}_1] = [\boldsymbol{U}_{11}]^{-1}[\boldsymbol{z}_1 - \boldsymbol{U}_{12} \times \boldsymbol{x}_2] \quad (7)$$

Here, $\boldsymbol{x}_2 = \boldsymbol{0}$ if there is no parent process.

If there are child processes, current process has to send elements of $\boldsymbol{x}_1$ to its child processes for updating vector $\boldsymbol{x}_2$.

4) Updating voltage vectors

Vector $\boldsymbol{x}_0$ records voltage deviations of related buses in level *k*+1 boundary blocks.

If there is a parent process, current process has to send corresponding elements of $\boldsymbol{x}_1$ to its parent process for updating $\boldsymbol{x}_0$.

Voltage vectors of vertices in InnerVs, LocalVs and OuterVs are updated by $\boldsymbol{x}_0$, $\boldsymbol{x}_1$, $\boldsymbol{x}_2$ respectively.

5) Convergence testing

Vector $\boldsymbol{y}_1$ is calculated by the voltage vector of LocalVs just obtained in stage 4. Current process exchanges the maximum $\|\boldsymbol{y}_1\|_\infty$ of $\boldsymbol{y}_1$ with others, and acquires the maximum $\max\|\boldsymbol{y}\|_\infty$ of all $\|\boldsymbol{y}_1\|_\infty$.

When the maximum $\max\|\boldsymbol{y}\|_\infty$ is less than the required precision, parallel power flow calculation exits with a convergent result.

6 Experimental results

There is a use-case that three power systems described in table 8 are used to investigate the efficiency of parallel power flow calculation which is based on NBBDF partitioning through edge cut sets. The programming language is C++ for NBBDF partitioning and parallel power flow calculation. The network environment is a LAN with bandwidth of 1000 Mbps. Taking account of collisions detected in Ethernet, the utility ratio of LAN is presumed to be 10% at least under light or no communication load, that is to say, the actual bandwidth is not less than 100 Mbps. In this distributed memory parallel architecture, each processor has a 1.8 GHz CPU and 1 GB memory individually.

Table 8 Three power systems for testing

System	Bus No	Line No	Transformer No
Case 1	300	304	107
Case 2	2806	1652	2305
Case 3	5317	3890	3275

In table 8, the first case is a standard grid model of IEEE while the following two are real electric grids in East China.

6.1 NBBDF partitioning

Before NBBDF partitioning, initial areas of grids should be given. Although geographic information based partitioning is an intuitional and convenient way to create few cut edges, the computational load imbalance among processors will influence parallel efficiency ultimately. Multilevel graph partitioning is used to determine initial areas of grids for the reason that it is superior to geographic information based partitioning in balancing computational loads

Table 9 Initial areas of three electric grids

System	Area code		Vertex number		
	Level 1	Level 2	PV	PQ	Total
Case 1	1	1	9	41	50
		2	9	40	49
	2	1	13	37	50
		2	16	32	48
	3	1	6	45	51
		2	15	36	51
Case 2	1	1	33	427	460
		2	54	433	487
	2	1	59	397	456
		2	57	416	473
	3	1	75	402	477
		2	56	396	452
Case 3	1	1	34	828	862
		2	104	738	842
	2	1	83	796	879
		2	121	812	933
	3	1	89	785	874
		2	33	893	926

Table 10 Boundary blocks of three electric grids after 2-nested BBDF partitioning

Level	Boundary block	Vertex number		
		Case 1	Case 2	Case 3
0	-	24	52	33
1	1	35	27	54
	2	16	61	45
	3	17	95	29
2	11	27	436	830
	12	24	480	805
	21	33	436	850
	22	43	409	905
	31	35	429	862
	32	45	380	903

Initial areas of three cases by multilevel graph partitioning are presented in table 9. In the original graph representing the electric grid, those vertices corresponding to slack nodes are not included in that voltages of slack nodes keep unchanged during the

calculation and those vertices need not participate in the iterative procedure. Here, there is only one slack node in each grid for testing.

After 2-nested BBDF partitioning to three grids, boundary blocks of all levels are shown in table 10. In this table, the first two columns correspond to property Level and property Colors of vertices in all boundary blocks respectively.

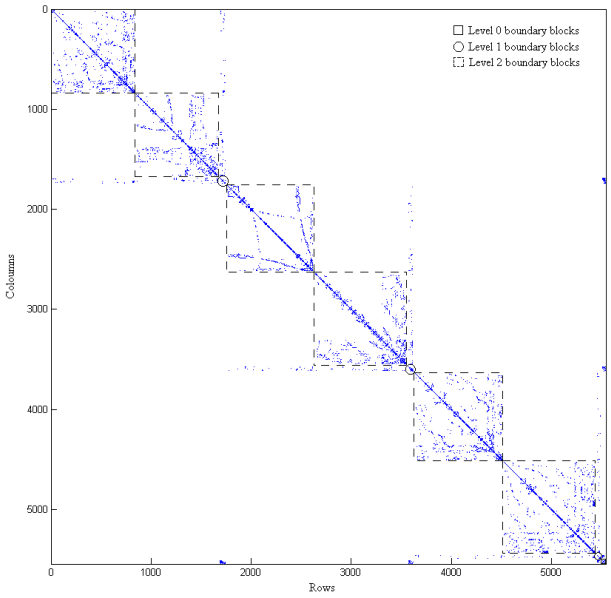

Fig.14 Admittance matrix of case 3 with NBBDF

The NBBDF system matrix of case 3 is exhibited in figure 14. It is evident that there is no connection between boundary blocks of non-contiguous levels. This conclusion coincides fully with the character of NBBDF partitioning by edge cut sets.

6.2 Partitioning performance assessment

The proposed NBBDF partitioning algorithm is compared with multilevel *k*-way partitioning in terms of a) runtime, b) cut-edge number and c) extra edges supplemented during the graph contraction to investigate its partitioning performance. Metis [22], a graph partitioning software package, is used to realize multilevel *k*-way partitioning, where *k* equals 6 to satisfy the same number of processors with NBBDF partitioning.

Comparison results are demonstrated in figure 15, in which the baseline represents results of multilevel *k*-way partitioning and bars represent ratios of results of NBBDF partitioning to those of multilevel *k*-way partitioning.

For three cases, runtimes of NBBDF partitioning are less than those of multilevel *k*-way partitioning. This is because computational complexities of these two methods are $o(|E|)$, where E is the edge count. Furthermore, multilevel *k*-way partitioning involves initial partitioning and refining which increase time overheads to the whole partitioning while NBBDF partitioning has to do nothing but traverse all edges.

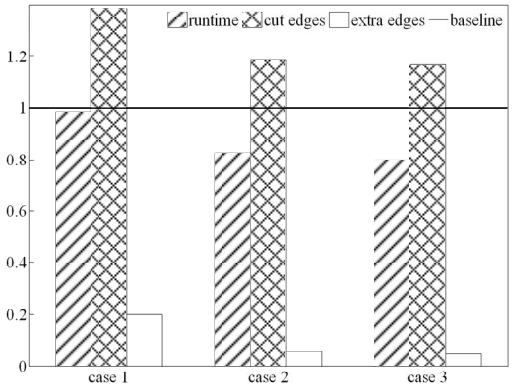

Fig.15 Performance of NBBDF partitioning by edge cut sets relative to that of multilevel *k*-way partitioning for three cases

In figure 15, it is found that cut edges of NBBDF partitioning are a bit more, particularly in case 1, than multilevel *k*-way partitioning. This is because cut edges among partitions are increasing as the partition number grows. Meanwhile, IEEE grid of case 1 is a strongly connected network, whereas real grids are loosely coupled among various geographic domains. In addition, the increase of nested levels is another factor for additional cut edges.

Table 11 Fill-ins in one iteration of Jacobian matrix Gaussian elimination

Partitioning		Fill-ins		
		Case 1	Case 2	Case 3
NBBDF	Level 0	124	368	198
	Level 1	1200	1696	952
	Level 2	1084	4672	19162
	Total	2408	6736	20312
Multilevel *k*-way	Level 0	3008	2896	4216
	Level 1	7918	117072	427078
	Total	10926	119968	431294

Matrix fill-ins during Gaussian elimination are proportional to extra edges in the graph contraction. For three cases, the fact that extra edges of NBBDF partitioning are significantly fewer than multilevel *k*-way method justifies a good effect of Tinney 3 scheme applied in the vertex ordering procedure. In table 11, fill-ins in one iteration of Jacobian matrix Gaussian elimination substantiate the effectiveness of vertex ordering to NBBDF graph as well.

6.3 Parallel performance evaluation

Aside from two parallel power flow calculations based on NBBDF and multilevel *k*-way partitioning, serial power flow calculation is involved in parallel performance evaluation. Particular attention is paid to one iteration of Gaussian elimination in order to investigate the efficiency of parallel power flow calculation in detail.

For further explanation, some symbols are given below, where parallel Gaussian elimination is based on NBBDF partitioning.

- T_{serial} - runtime of one iteration of serial Gaussian elimination;
- T_{parallel} - runtime of one iteration of parallel Gaussian elimination;
- Q_{Com} - communications volume in one iteration of parallel Gaussian elimination;
- T_{Com} - communication time in one iteration of parallel Gaussian elimination;
- T_{operands} - serial computation time of one iteration of parallel Gaussian elimination in one processor;
- T_{Steps} - parallel computation time of one iteration of parallel Gaussian elimination in more than one processor;
- λ_{p} - average parallelism degree;
- S_{p} - speedup ratio;
- E_{p} - parallel efficiency;
- p - number of processors;
- N_{I} - iterations of Gaussian elimination;
- T_{s} - runtime of serial power flow calculation;
- T_{p} - runtime of parallel power flow calculation based on NBBDF partitioning.

Definition 2: To an algorithm, average parallelism degree is the quotient of total operands divided by the number of steps.

In a parallel algorithm, whether operands or steps are proportional to the execution time which includes no communication time. Hence, T_{oprands} is responsible for operands and T_{Steps} is for steps.

Following equations are presented for revelation about relations of these symbols.

$$\lambda_{\text{p}} = \frac{T_{\text{operands}}}{T_{\text{Steps}}} = \frac{T_{\text{operands}}}{T_{\text{Parallel}} - T_{\text{Com}}} \tag{8}$$

$$S_{\text{p}} = \frac{T_{\text{serial}}}{T_{\text{Parallel}}} \tag{9}$$

$$E_{\text{p}} = \frac{S_{\text{p}}}{p} \tag{10}$$

A parallel algorithm is recognized as superlinear one when $E_{\text{p}} > 1$.

Upon the presumption of light-loaded LAN with bandwidth of 100 Mbps at least, T_{Com} is directly proportional to Q_{Com}. Communication statistics in one iteration of parallel Gaussian elimination are listed in table 12.

Table 12 Communication overheads in one iteration of parallel Gaussian elimination

System	Q_{Com}/kb	Bandwidth/Mbps	T_{Com}/ms
Case 1	43.80	100	0.4380
Case 2	88.98	100	0.8898
Case 3	180.71	100	1.8071

In parallel Gaussian elimination, it is the forward substitution, in which elements of updating matrix $\Delta \boldsymbol{J}_{22}$ need to be sent, that dominates the whole communication. Moreover, the number of vertices of any level boundary block places an upper limit on the size of the updating matrix $\Delta \boldsymbol{J}_{22}$ under *N*-nested BBDF partitioning.

According to table 10, for three cases, sums of vertices of level 1 and level 0 boundary blocks are 92, 235 and 161 respectively. Thus, there is least communications volume in case 1 for its smallest sum. The reason for more communications volume of case 3 than case 2 is that there are nearly double cut edges in case 3 than those of case 2.

Average parallelism degrees are listed in table 13, followed by speedup ratios and parallel efficiencies in table 14. As a result of similar task graphs, each case has been assigned six processors, that is, $p = 6$ for all cases.

Table 13 Average parallelism degrees in one iteration of parallel Gaussian elimination

System	T_{operands}/ms	T_{Steps}/ms	λ_{p}
Case 1	4.1260	1.1867	3.4769
Case 2	38.1378	7.8315	4.8698
Case 3	109.0773	20.1700	5.4079

Table 14 Speedup ratios and parallel efficiencies in one iteration of parallel Gaussian elimination

System	T_{serial}/ms	T_{parallel}/ms	S_{p}	E_{p}
Case 1	5.1303	1.6247	3.1577	0.5263
Case 2	50.0430	8.7213	5.7380	0.9563
Case 3	134.2626	21.9771	6.1092	1.0182

According to table 13 and table 14, it is telling that average parallelism degrees along with speedup

ratios are increasing as the system scale grows. And, the parallel efficiency surpasses 100% in case 3.

There are three causes for this superlinearity. The first one is that sum of consuming times in updating partial Jacobian matrices in parallel processors is smaller than that of updating the integrated Jacobian matrix in one processor. The second reason is that manipulating non-zero elements of a large matrix, like inserting or deleting one, has to rearrange more non-zero elements than those of several submatrices which make up this large matrix if Compressed Row Storge (CRS) has been adopted. The third cause is a large computation-communication ratio denoted by the quotient of the computation time divided by the communication time as shown in equation (11).

$$\chi = \frac{T_{\text{Steps}}}{T_{\text{Com}}} \tag{11}$$

Where, χ is the computation-communication ratio.

The first two causes do well in explaining why $T_{\text{operands}} < T_{\text{serial}}$ for three cases. Besides, increasing computation-communication ratios that are inferred from table 12 and table 13 by virtue of equation (11) lend evidence to the third cause.

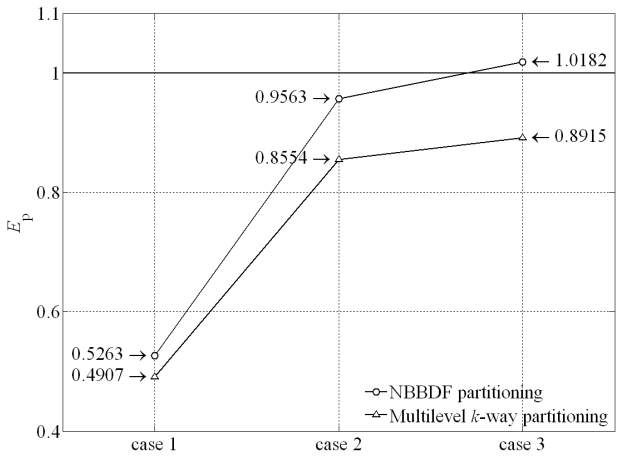

Fig.16 Parallel efficiencies in one iteration of Gaussian elimination under different partitioning algorithms

The comparison of parallel efficiencies between proposed NBBDF partitioning and multilevel *k*-way partitioning are shown in figure 16. Multilevel *k*-way partitioning uses centralized file storage. In this figure, it is found that there is no way of achieving superlinearity without distributed file storage no matter how efficient the partitioning method is.

Table 15 lists the performance of parallel power flow calculation including loading data from files.

Table 15 Performances of parallel power flow calculation under 2-nested BBDF partitioning

System	N_I	T_s /s	T_p /s	S_p	E_p
Case 1	6	0.0468	0.0129	3.6279	0.6047
Case 2	7	1.4042	0.2178	6.4472	1.0745
Case 3	7	5.3810	0.7389	7.2824	1.2137

By contrast with table 14, the parallel efficiency has made greater progress than that in one iteration of parallel Gaussian elimination. Furthermore, its growing rate has also kept pace with the expanding electric grid. In case 3, the growing rate of parallel efficiency reaches up to 19.6%. This achievement is attributed primarily to distributed file storage, which decreases loading time, especially the topological analysis of the grid, to a great extent. Thereby, the total computation time has declined.

7 Conclusion

A new partitioning algorithm based, which is engaged in transforming system matrices of power grids into NBBDF, parallel power flow calculation is presented in this paper. Features distinguishing the proposed method from other algorithms are concluded below.

1) Partitioning the original graph into NBBDF by edge cut sets in a recursive way cuts off relations between boundary blocks of non-contiguous levels, and improves average parallelism degree as a result;

2) In order to bring as few fill-ins as possible into the matrix Gaussian elimination, a Tinney 3 based vertex ordering scheme together with its parallel implementation is employed after partitioning.

3) The problem of time-consuming initialization before starting parallel computing is solved by way of distributed file storage that provides a task graph for task scheduling.

4) Aiming at full utilizing processors available, tasks are assigned to running processors in terms of precedence relations in the task graph. At this point, the detailed implementation of parallel power flow calculation is given.

Final experimental results for three power grids imply that the proposed method not only gains high performance, what's more, it is able to contribute to superlinearity in large-scale grids.

In the future research, this partitioning method should be investigated in various applications of power systems such as stability analysis of voltages, dynamic stability computation and so forth.

References:

[1] G. Kron, Diakoptics: The piecewise solution of large scale systems, MacDonald, 1963.
[2] A. Sangiovanni-Vincentelli, L. K. Chen, and L. Chua, An efficient heuristic cluster algorithm for tearing large-scale networks, IEEE Trans. on Circuits and Systems, Vol.24, No.12, 1977, pp. 709-717.
[3] E. C. Ogbuobiri, W. F. Tinney, and J. W. Walker, Sparsity-directed decomposition for Gaussian elimination on matrices, IEEE Trans. on Power Apparatus and Systems, Vol.89, No.1, 1970, pp. 141-150.
[4] C. P. Yuan, R. Lucas, P. Chan, et al, Parallel electronic circuit simulation on the IPSC system, IEEE Custom Integrated Circuits Conference, 1988, pp. 6.5.1-6.5.4.
[5] N. Frohlich, V. Glockel, and J. Fleischmann, A new partitioning method for parallel simulation of VLSI circuits on transistor level, Design, Automation and Test in Europe Conference and Exhibition, 2000, pp. 1-6.
[6] K. W. Chan, A. R. Daniels, R. W. Dunn, et al, A partitioning algorithm for parallel processing of large power systems network equations, IEE 2nd International Conference on Advances in Power System Control, Operation and Management, 1993, pp. 893-898.
[7] K. W. Chan, R. W. Dunn, and A. R. Daniels, Efficient heuristic partitioning algorithm for parallel processing of large power systems network equations, IEE Proc.-Generation, Transmission and Distribution, Vol.142, No.6, 1995, pp. 625-630.
[8] B. Hendrickson, and R. Leland, A multilevel algorithm for partitioning graphs, ACM/IEEE Supercomputing Conference, 1995, pp. 1-14.
[9] G. Karypis, and V. Kumar, Multilevel k-way partitioning scheme for irregular graphs, Journal of Parallel and Distributed Computing, Vol.48, No.1, 1998, pp. 96-129.
[10] B. W. Kernighan, and S. Lin, An efficient heuristic procedure for partitioning graphs, Bell System Technical Journal, Vol.49, No.2, 1970, pp. 291-307.
[11] M. C. Chang, and I. N. Hajj, iPRIDE a parallel integrated circuit simulator using direct method, IEEE International Conference on Computer-Aided Design, 1988, pp. 304-307.
[12] C. C. Chen, and Y. H. Hu, Parallel LU factorization for circuit simulation on an MIMD computer, IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1988, pp. 129-132.
[13] P. Y. Chung, and I. N. Hajj, Parallel solution of sparse linear systems on a vector multiprocessor computer, IEEE International Symposium on Circuits and Systems, 1990, pp. 1577-1580.
[14] J. Shu, W. Xue, and W. M. Zheng, A parallel transient stability simulation for power systems, IEEE Transactions on Power Systems, Vol.20, No.4, 2005, pp. 1709-1717.
[15] K. W. Chan, R. C. Dai, and C. H. Cheung, A coarse grain parallel solution method for solving large set of power systems network equations, International Conference on Power System Technology, Vol.4, 2002, pp. 2640-2644.
[16] Y. Li, X. X. Zhou, and J. Y. Zhou, A new algorithm for distributed power system state estimation based on PMUs, International Conference on Power System Technology, 2006, pp. 1-6.
[17] M. H. M. Vale, D. M. Falcao, and E. Kaszkurewicz, Electrical power network decomposition for parallel computations, IEEE International Symposium on Circuits and Systems, 1992, pp. 2761-2764.
[18] A. I. Zecevic, and D. D. Siljak, Balanced decompositions of sparse systems for multilevel parallel processing, IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications, Vol.41, No.3, 1994, pp. 220-233.
[19] M. Vlach, LU decomposition and forward-backward substitution of recursive bordered block diagonal matrices, Electronic Circuits and Systems, Vol.132, No.1, 1985, pp.24-31.
[20] R. Berry, An optimal ordering of electronic circuit equations for a sparse matrix solution, IEEE Trans. on Circuits Theory, Vol.18, No.1, 1971, pp. 40-50.
[21] K. W. Chan, Parallel algorithms for direct solution of large sparse power system matrix equations, IEE Proc.-Generation, Transmission and Distribution, Vol.148, No.6, 2001, pp. 615-622.
[22] G. Karypis, METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices, http://www. cs.umn.edu/~ karypis, 1998.

Appendix:

Proof of Theorem 1: According to theorem 1, the original graph $G(V,E)$ has been partitioned into n independent subgraphs $G_i^{'}(V_i^{'},E_i^{'})$ and one subgraph

$G_{\text{boundary}}(V_{\text{boundary}}, \Im)$ that is related to each $G_i^{'}(V_i^{'}, E_i^{'})$. This substantiation involves two respects.

a) Independency

Assuming that there are two subgraphs $G_i^{'}(V_i^{'}, E_i^{'})$ and $G_j^{'}(V_j^{'}, E_j^{'})$ which are connected with each other, thus, there is one edge $e_{kl} = (v_k, v_l)$ at least satisfying $e_{kl} = \left\{(v_k, v_l) \middle| v_k \in V_i^{'} \,\&\, v_l \in V_j^{'}\right\}$ and $e_{kl} \notin \left[\mathrm{V}_i, \overline{\mathrm{V}}_i\right]_G$.

Owing to $V_i \bigcap V_j = \phi$, $V_i^{'} \subset V_i$ and $V_j^{'} \subset V_j$, it is inferred that $V_i^{'} \bigcap V_j^{'} = \phi$ and $v_l \notin V_i^{'} \subset V_i$. Given that $v_k \in V_i^{'} \subset V_i$, it is concluded that $e_{kl} \in \left[\mathrm{V}_i, \overline{\mathrm{V}}_i\right]_G$, which is contradictory to $e_{kl} \notin \left[\mathrm{V}_i, \overline{\mathrm{V}}_i\right]_G$ and renders this assumption invalid.

b) Connectivity

Assuming that there is an independent subgraph $G_i^{'}(V_i^{'}, E_i^{'})$ not connected with $G_{\text{boundary}}(V_{\text{boundary}}, \Im)$, there is no edge $e_{kl} = \left\{(v_k, v_l) \middle| v_k \in V_i^{'} \,\&\, v_l \in V_{\text{boundary}}\right\}$ accordingly.

As $G(V, E)$ is a connected graph, the edge cut $\xi_i = [V_i, \overline{V_i}]_G$ of subgraph G_i is not empty. G_{boundary} is the edge-induced subgraph of $\xi_i = [V_i, \overline{V_i}]_G$, hence, $V_i \bigcap V_{\text{boundary}} \neq \phi$.

No such edge $e_{kl} = \left\{(v_k, v_l) \middle| v_k \in V_i^{'} \,\&\, v_l \in V_{\text{boundary}}\right\}$ means that there is no edge $e_{kj} = (v_k, v_j)$ satisfying $v_k \in V_i^{'} \subset V_i$ and $v_j \in V_i \bigcap V_{\text{boundary}}$. As a result, initial subgraph $G_i(V_i, E_i)$ is disconnected, which violates the precondition in theorem 1. So this assumption is unjustifiable.

Proof of Theorem 2: This substantiation is carried out in a recursive way.

First, there are *k* vertices in connected subgraph $G_c(V_c, E_c)$. According to lemma 1, this theorem is reasonable when $k = 1$.

Presuming this theorem is tenable when $k \le i$, two cases have to be considered when $k = i + 1$, and v_{i+1} is a vertex which is contracted last in $G_c(V_c, E_c)$.

a) v_{i+1} is a cut vertex.

There are two vertex sets S_1 and S_2, which are vertex sets of two connected components, satisfying $V_c = S_1 \bigcup S_2 \bigcup v_{i+1}$. In figure A1, A_0, A_1 and A_2 are adjacent vertex sets of v_{i+1}, S_1 and S_2 respectively.

According to the presumption, vertices consisted of v_{i+1} and vertices in $A_1 \bigcup A_2$ are connected with each other after S_1 and S_2 have been contracted. Then, the subgraph $A = A_0 \bigcup A_1 \bigcup A_2$, induced from adjacent vertices of $G_c(V_c, E_c)$, is a complete graph after v_{i+1} has been contracted according to lemma 1. Hence, this theorem is proven.

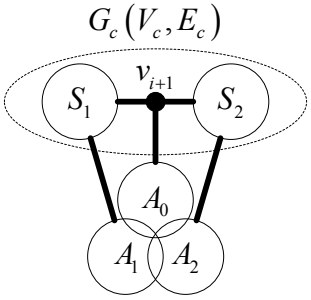

Fig. A1 v_{i+1} is a cut vertex

b) v_{i+1} isn't a cut vertex.

S_1, which is a vertex set of rest vertices, satisfies $V_c = S_1 \bigcup v_{i+1}$. In figure A2, A_0 and A_1 are adjacent vertex sets of v_{i+1} and S_1 respectively.

According to the presumption, vertices consisted of v_{i+1} and vertices in A_1 are connected with each other after S_1 has been contracted. Similar to case a), the subgraph $A = A_0 \bigcup A_1$, induced from adjacent vertices of $G_c(V_c, E_c)$, is a complete graph after v_{i+1} has been contracted according to lemma 1. Thereby, this theorem is justified.

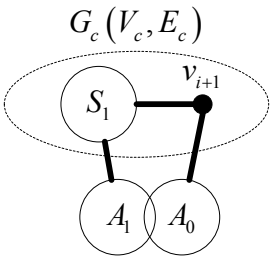

Fig. A2 v_{i+1} is a non-cut vertex