Available online at www.sciencedirect.com
ScienceDirect
Electrical Power and Energy Systems xxx (2006) xxx–xxx
ELECTRICAL POWER & ENERGY SYSTEMS
www.elsevier.com/locate/ijepes

Parallel solution of Newton's power flow equations on configurable chips

Xiaofang Wang [a], Sotirios G. Ziavras [b,*], Chika Nwankpa [c], Jeremy Johnson [d], Prawat Nagvajara [c]

[a] Department of Electrical and Computer Engineering, Villanova University, Villanova, PA 19085, USA
[b] Department of Electrical and Computer Engineering, New Jersey Institute of Technology, University Heights, Newark, NJ 07102, USA
[c] Department of Electrical and Computer Engineering, Drexel University, Philadelphia, PA 19104, USA
[d] Department of Computer Science, Drexel University, Philadelphia, PA 19104, USA

Received 19 January 2005; received in revised form 12 September 2006; accepted 17 October 2006

Abstract

The conventional Newton's method (also known as Newton–Raphson method) for the AC power flow problem is preferred in some situations due to its local quadratic convergence. However, its high computation and memory requirements due to the required LU factorization of the Jacobian matrix at each iteration limit its practical employment in the online operation of very large systems. We produce here a novel partitioning scheme for the nonsymmetric Jacobian matrices appearing in the Newtons's method. It results in the efficient parallelization of LU factorization and the subsequent solution of the power flow equations. We also present our implementation on our target computing platform comprising a single-chip shared-memory configurable multiprocessor. We designed and implemented our multiprocessor on an SOPC (system-on-a-programmable-chip) computer board containing an FPGA (field-programmable gate array) device. This new configurable computing paradigm combines the flexibility of microprocessors and programmable logic with the high performance of ASIC (application-specific integrated circuit) designs, and facilitates low-cost parallel implementations with reasonable turnaround times. Our good performance results for IEEE power test systems and others representing parts of the US power grid in the northeast demonstrate that our cost-effective and robust approach is viable and has tremendous potential to be enhanced further with steady advances in silicon technology, as predicted by Moore's Law.

Keywords: Newton's power flow equations; FPGA; Multiprocessor

1. Introduction

Among the vast number of power flow solutions, two major categories of solvers have been thoroughly investigated and widely employed by the power research and industry communities: Newton's method [1] and the Fast Decoupled Power Flow (FDPF) method [2]. Newton's method solves repeatedly simultaneous, large, sparse, and linear systems of equations. The LU factorization of the Jacobian matrix at each iteration in the direct solution of the linear equations has been a great challenge to the computation capability and memory capacity of available computing platforms [3]; this is the main reason that the employment of an exact Newton's method is often avoided, especially when the computation is carried out in real time and/or involves very large power systems. On the other hand, FDPF algorithms require LU factorization only once and the result is repeatedly used throughout the entire power flow analysis process by employing a fixed and smaller coefficient matrix. Thus, the solution time can be reduced dramatically. Although many improvements can make FDPF more robust, in some cases where the coefficient matrices are ill-conditioned FDPF has convergence

* Corresponding author. Tel.: +1 973 596 5651; fax: +1 973 596 5680.
E-mail address: ziavras@adm.njit.edu (S.G. Ziavras).

0142-0615/$ - see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.ijepes.2006.10.006

difficulties even with the application of good pre-conditioners. The conventional Newton's method is still commonly employed by the power industry.

To overcome the heavy computation demands caused by the LU factorization in Newton's method, parallel computing techniques may be applied. However, parallel computing is not very common in power engineering due mainly to the scarce availability of low-cost, high-performance parallel machines suitable for these tasks [3,4]. Although parallel computers have been successful in solving several computation-intensive problems, their high prices and long design and development cycles, and the high cost of maintaining them often make their long term availability unpredictable [5]. Moreover, the efficient parallelization of the exact Newton's method has proved to be a Herculean task and few good speedups have been reported in the literature. PC clusters have emerged in recent years as a parallel-computing alternative to take advantage of the ever-increasing computing power of commercial-off-the-shelf (COTS) general-purpose microprocessors. Thus, parallel implementations of FDPF methods on PC clusters for solving the AC power flow problem have appeared [6,7]. However, the high communication overheads present in these platforms quickly diminish any performance gains when increasing the cluster size; therefore, these implementations suffer in terms of scalability and efficiency. Details of relevant work are presented in Section 6 after the introduction of our proposed algorithm.

On the other hand, many good parallel direct solvers targeting fine- or medium-grain parallelism have been developed [8,9], such as SuperLU [10] and S+ [11]. These packages are often optimized for proprietary parallel computers. However, the performance of such solvers has not been thoroughly analyzed for power matrices. In [8], the speedup of the best solver for circuit simulation matrices, which are similar to power matrices but more dense, was shown to decrease with increases in the matrix sparsity. Some other solvers show no speedup at all for such matrices [8]. Some research also has shown that SuperLU does not result in performance gains for circuit simulation matrices [12]. Coarse-grain parallel algorithms based on network partitioning have been demonstrated to be very efficient and promising for power matrices [7,19]. Our first contribution in this paper is a parallel solution to Newtons's method for nonsymmetric Jacobian matrices based on a novel partitioning scheme that results in the efficient parallelization of LU factorization for the subsequent solution of the power flow equations.

On the hardware side, our primary motivation is to explore new, promising approaches in parallel computer architecture that are viable in the long term and are also cost-effective for computation-intensive applications, such as power flow analysis. Recent dramatic advances in multi-million gate platform FPGAs, with rich embedded feature sets, such as plenty of on-chip memory, DSP blocks and embedded hardware microprocessor IP (intellectual property) cores [15,21], have inspired our investigation of MPoPC (MultiProcessor on a Programmable Chip) designs for the power flow problem. FPGA-based configurable computing machines have shown significant results in the last decade for improving the performance of algorithms in numerous fields, such as DSP, data communication, genetics, image processing, pattern recognition, etc. [20], by carefully hand-tuning the hardware design. They provide the benefits of customized ASIC (Application-Specific Integrated Circuit) designs while reducing dramatically the high non-recurring engineering (NRE) costs, long design cycles, and inherent risks associated with the latter. However, due to limited on-chip resources of previous FPGAs, many advanced features, such as hardware floating-point units (FPUs) and large on-chip memory, were not feasible. With advances in recent years, the peak floating-point performance of FPGAs has outnumbered that of modern microprocessors and is growing much faster than the latter [22]. The major advantages of MPoPCs include flexibility (supported by both embedded programmable logic and microprocessors) and low cost. For MPoPC designs, the overall system performance never depends solely on the individual processor frequency or its execution rate provided that the design offers many alternatives in customizing the system to yield the best performance. They also feature dramatically reduced communication overheads and the potential for very high degree of parallelism, which make MPoPCs more scalable compared to other parallel machines. Scalability across multiple MPoPCs can be supported with several multi-chip interconnection schemes [13,14]. Taking advantage of the high density of new generation FPGAs, we are among the first groups that implement MPoPCs involving FPUs for the IEEE 754 FP standard.

We present here our improved implementation on an Altera SOPC development board [15] of a shared-memory MIMD (multiple-instruction, multiple-data) multiprocessor that employs Altera's Nios® configurable IP processor for computing nodes. We have also reported good performance for the parallel doubly-bordered block diagonal (DBBD) LU factorization algorithm on a previously designed machine [16,17]. In this paper, we propose a novel partitioning technique for nonsymmetric Jacobian matrices and use the DBBD algorithm to solve the power flow problem in parallel with Newton's method; we test our approach with the 57-, 118- and 300-bus IEEE test systems, and 1648- and 7917-bus systems representing parts of the Northeastern US power grid.

Section 2 discusses briefly Newton's approach to the power flow problem. Section 3 presents the reordering of sparse matrices into the DBBD form that can facilitate numerous simultaneous operations in LU factorization. Section 4 contains our parallel algorithm for Newton's approach. Section 5 presents the architecture of our configurable parallel computer. A comparison with other relevant parallel approaches based on DBBD partitioning is included in Section 6. Our experimental results are presented in Section 7. Section 8 presents our conclusions and future work.

Please cite this article as: Xiaofang Wang et al., Parallel solution of Newton's power flow equations ..., Int J Electr Power Energ Syst (2006), doi:10.1016/j.ijepes.2006.10.006

2. Newton's solution to the power flow problem

The main objective of power flow analysis is to determine precise steady-state voltages (magnitudes and angles) on all buses in a given network, and then to derive from them the real and reactive power flows into every line and transformer; the network topology and all information about the generation and load lines are known. For most network buses, the active and reactive powers are specified; they can be evaluated by the following equations for a network with N buses [18]:

$$P_i = \sum_{k=1}^{N} |y_{ik} V_i V_k| \cos(\theta_{ik} + \delta_k - \delta_i) \tag{1}$$

$$Q_i = \sum_{k=1}^{N} |y_{ik} V_i V_k| \sin(\delta_i - \delta_k - \theta_{ik}) \tag{2}$$

where P_i, Q_i and V_i, are the active power, reactive power and complex voltage at bus i, respectively, with $V_i = |V_i|\angle\delta_i, V_k = |V_k|\angle\delta_k, y_{ik} = |y_{ik}|\angle\theta_{ik} = g_{ik} + jb_{ik}$, for $i,k \in [1,N]$; y_{ik} is an element of the bus admittance matrix (Y_{bus} matrix). If the number of voltage-controlled buses in the system is N_g, then we need to solve $(2N–N_g–2)$ equations.

Newton's method [1] expands these equations into a Taylor series and incorporates the first-derivative information when updating the voltages. Thus, the following linear equations are produced to be solved iteratively until the mismatches $\triangle\delta$ and $\triangle V$ are smaller than a pre-specified tolerance:

$$\begin{bmatrix} J^{11} & J^{12} \\ J^{21} & J^{22} \end{bmatrix} \begin{bmatrix} \triangle\delta \\ \frac{\triangle V}{|V|} \end{bmatrix} = \begin{bmatrix} \triangle P \\ \triangle Q \end{bmatrix} \tag{3}$$

The Jacobian matrix $J = \{J^{11}, J^{12}, J^{21}, J^{22}\}$ is reevaluated at each iteration by the following equations that use updated voltages:

$$\mathbf{J}^{11}: \begin{cases} \frac{\partial P_i}{\partial \delta_j} = |V_i||V_j|(g_{ij}\sin\delta_{ij} - b_{ij}\cos\delta_{ij}) \quad j \neq i \\ \frac{\partial P_i}{\partial \delta_i} = -|V_i|\sum\limits_{\substack{i=1 \\ i\neq j}}^{N} |V_j|(g_{ij}\sin\delta_{ij} - b_{ij}\cos\delta_{ij}) \end{cases} \tag{4}$$

$$\mathbf{J}^{12}: \begin{cases} |V_j|\frac{\partial P_i}{\partial |V_j|} = |V_i||V_j|(g_{ij}\cos\delta_{ij} + b_{ij}\sin\delta_{ij}) \quad j \neq i \\ |V_i|\frac{\partial P_i}{\partial |V_i|} = 2|V_i|^2 g_{ii} + |V_i|\sum\limits_{\substack{i=1 \\ i\neq j}}^{N} |V_j|(g_{ij}\cos\delta_{ij} + b_{ij}\sin\delta_{ij}) \end{cases} \tag{5}$$

$$\mathbf{J}^{21}: \begin{cases} \frac{\partial Q_i}{\partial \delta_j} = -|V_i||V_j|(g_{ij}\cos\delta_{ij} + b_{ij}\sin\delta_{ij}) \quad j \neq i \\ \frac{\partial Q_i}{\partial \delta_i} = |V_i|\sum\limits_{\substack{i=1 \\ i\neq j}}^{N} |V_j|(g_{ij}\cos\delta_{ij} + b_{ij}\sin\delta_{ij}) \end{cases} \tag{6}$$

$$\mathbf{J}^{22}: \begin{cases} |V_i|\frac{\partial Q_i}{\partial |V_j|} = |V_i||V_j|(g_{ij}\sin\delta_{ij} - b_{ij}\cos\delta_{ij}) \quad j \neq i \\ |V_i|\frac{\partial Q_i}{\partial |V_i|} = |V_i|\sum\limits_{\substack{i=1 \\ i\neq j}}^{N} |V_j|(g_{ij}\sin\delta_{ij} - b_{ij}\cos\delta_{ij}) - 2V_i^2 b_{ii} \end{cases} \tag{7}$$

In order to derive the mismatches at each iteration from the above linear equations, two kinds of methods are usually employed: direct and iterative methods [18]. LU factorization followed by forward reduction and backward substitution [9] is one of the most widely used direct methods to solve the linear systems; it applies a sequence of Gaussian eliminations to form $PJ = LU$, where L is a unit lower triangular matrix (with 1's on the main diagonal), U is upper triangular, and P is a permutation matrix. Then, Eq. (3) can be solved by the following two sets of equations:

$$Lw = \begin{bmatrix} \Delta P \\ \Delta Q \end{bmatrix} \tag{8}$$

$$U \begin{bmatrix} \Delta\delta \\ \frac{\Delta V}{|V|} \end{bmatrix} = w \tag{9}$$

Direct methods are usually more robust and efficient for larger matrices, but it may be difficult to extract substantial parallelism while maintaining a low inter-processor communication overhead. Moreover, the typical computation complexity of LU factorization is $O(M^3)$, where $M \times M$ is the matrix size (i.e., the size of the Jacobian matrix in our case). In Newton's method, the LU factorization is applied on a new Jacobian matrix at each iteration. The repetitive solution of the linear equations in Newton's method is very time-consuming for large networks, if the problem is solved sequentially.

There have been many attempts to develop parallel algorithms optimized for different parallel architectures for efficiently solving the power flow problem. Excellent reviews of high-performance computing efforts in power engineering appeared in [3,4]. Our parallel approach stems from the fact that the Y_{bus} and Jacobian matrices are usually very sparse, especially for large networks. Table 1 shows the sparsity (percentage of nonzero elements in a matrix) in the benchmark matrices used in this paper. For networks with thousands of buses, the typical number of nonzero elements per row is less than four. Although the LU factorization of sparse matrices potentially has fewer operations than that of dense matrices, it can suffer tremendously from dynamic fill-ins. As we discussed in the Introduction, network partitioning is a promising approach to reduce the number of operations applied to power matrices. The basic idea behind such an approach is to divide an interconnected network into independent sub-networks and a collection of cutting nodes. The DBBD form is obtained by reordering a given sparse matrix based on network partitioning. This way, LU factorization can first be applied to completely independent sub-networks in parallel, thus speeding up the algorithm dramatically. The information corresponding to the cutting nodes is then processed at a lower rate. Details follow in Section 3.

Please cite this article as: Xiaofang Wang et al., Parallel solution of Newton's power flow equations ..., Int J Electr Power Energ Syst (2006), doi:10.1016/j.ijepes.2006.10.006

Table 1
Sparsity of the benchmark matrices

Systems	57	118	300	1648	7917
Braches	80	186	411	2516	12,147
Dimensionality of the Jacobian matrix	106	181	530	2982	14,508
% of nonzeros in the Y_{bus} matrix	6.56	3.42	1.24	0.246	0.0514
% of nonzeros in the J matrix	6.40	3.21	1.33	0.244	0.0513

3. Parallel LU factorization of Jacobian matrices

3.1. Network partitioning

The sparse Y_{bus} matrix can be reordered into the DBBD form shown in Fig. 1 by the node tearing technique [19] or other similar heuristics-based algorithm. $\{Y_{ii}, Y_{in}, Y_{ni}\}$, for $i \in [1, n-1]$, are matrix blocks representing a sub-network; for a given value of i, this will be referred to as a 3-block group. Y_{nn} represents cutting nodes and is referred as the last block. All other blocks contain all zeros. Let N_c be the number of cutting nodes, i.e., the size of Y_{nn} is $N_c \times N_c$. The maximum number of nodes in each diagonal block and, hence, the sparsity of the blocks can be tuned by a user parameter, *MaxNodes*, during network partitioning. Generally speaking, the more diagonal blocks (sub-networks) in the DBBD matrix, the denser the blocks are and the larger the last block Y_{nn} is. Different orderings may result in big differences in the total solution time of the equations. In our implementation of the reordering technique, we seek an ordering with a large number of diagonal blocks but not a too large Y_{nn}. This objective is justified in Subsection 3.2.

The node tearing technique assumes that the matrix is symmetric whereas Newton's method requires employs a nonsymmetric Jacobian matrix as the coefficient matrix. To obtain the DBBD form for the Jacobian matrix, we first examine Eqs. (4)–(7) to produce J^{ik} $(i, k = 1, 2)$. We observe that every element in J^{ik} is directly related to the corresponding element in the Y_{bus} matrix; the zero elements in the Y_{bus} matrix cause the corresponding elements in each of the four quadrants in the Jacobian matrix to be zero. This reveals a structural similarity involving nonzero elements in the J^{ik} and Y_{bus} matrices. After we order the Y_{bus} matrix into the DBBD form, the corresponding Jacobian matrix should have the form shown in Fig. 2(a). In this figure, $\{J_{i\_d11}, J_{i\_u11}, J_{i\_b11}$, for $i \in [1, n-1]\}$, J_{n_d11} and along with all the other zero elements in the corresponding quadrant constitute J^{11} in Eq. (3), whereas $\{J_{i\_d12}, J_{i\_u12}, J_{i\_b12}\}$ and J_{n_d12} are in J^{12}, and so on. The sizes of the diagonal blocks in the four quadrants are shown in Table 2. S_{yi} is the size of the ith diagonal block in the Y_{bus} matrix and N_{gi} is the number of PV buses that appear in the ith diagonal block of the Y_{bus} matrix. $N_g = \sum_{i=1}^{n} N_{gi}$ is the total number of PV buses in the system.

$$\begin{pmatrix} Y_{11} & 0 & \dots & 0 & Y_{1n} \\ 0 & Y_{22} & \dots & 0 & Y_{2n} \\ \vdots & 0 & \dots & 0 & \vdots \\ 0 & 0 & \dots & Y_{n-1,n-1} & Y_{n-1,n} \\ Y_{n1} & Y_{n2} & \dots & Y_{n,n-1} & Y_{nn} \end{pmatrix}$$

Fig. 1. Sparse DBBD Y_{bus} matrix.

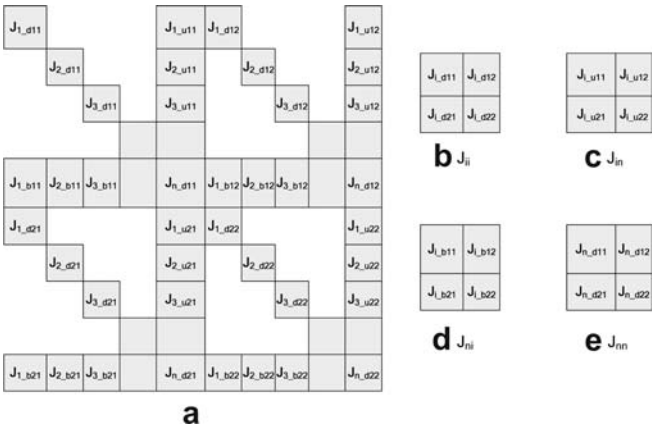

Fig. 2. The Jacobian matrix produced from the DBBD Y_{bus} matrix.

Table 2
The sizes of the blocks in the Jacobian matrix

Block	J_{i_d11}	J_{i_d12}	J_{i_d21}	J_{i_d22}
Rows	S_{yi}	S_{yi}	$S_{yi}-N_{gi}$	$S_{yi}-N_{gi}$
Columns	S_{yi}	$S_{yi}-N_{gi}$	S_{yi}	$S_{yi}-N_{gi}$

If we permute the Jacobian matrix shown in Fig. 2(a) in such a way that the four blocks related to the same ith diagonal block in the Y_{bus} matrix are grouped together, we can find that the Jacobian matrix can also be represented in the DBBD form, with the ith diagonal block being of size $(2S_{yi} - N_{gi})$. The blocks $\{J_{i\_d11}, J_{i\_d12}, J_{i\_d21}, J_{i\_d22}\}$, $\{J_{i\_u11}, J_{i\_u12}, J_{i\_u21}, J_{i\_u22}\}$, and $\{J_{i\_b11}, J_{i\_b12}, J_{i\_b21}, J_{i\_b22}\}$, for $i \in [1, n-1]$, form the new 3-block groups $\{J_{ii}, J_{in}, J_{ni}\}$ in the DBBD Jacobian matrix (shown in Figs. 2(b), (c) and (d)), and $J_{nn} = \{J_{n\_d11}, J_{n\_d12}, J_{n\_d21}, J_{n\_d22}\}$ (shown in Fig. 2 (e)) becomes the new last block in the DBBD Jacobian matrix. Fig. 3 shows the nonzero elements in different matrices for the 7917-bus system.

3.2. Parallel LU factorization of the DBBD Jacobian matrix

After we order the Jacobian matrix into the DBBD form, Eq. (3) can be solved by the parallel DBBD LU

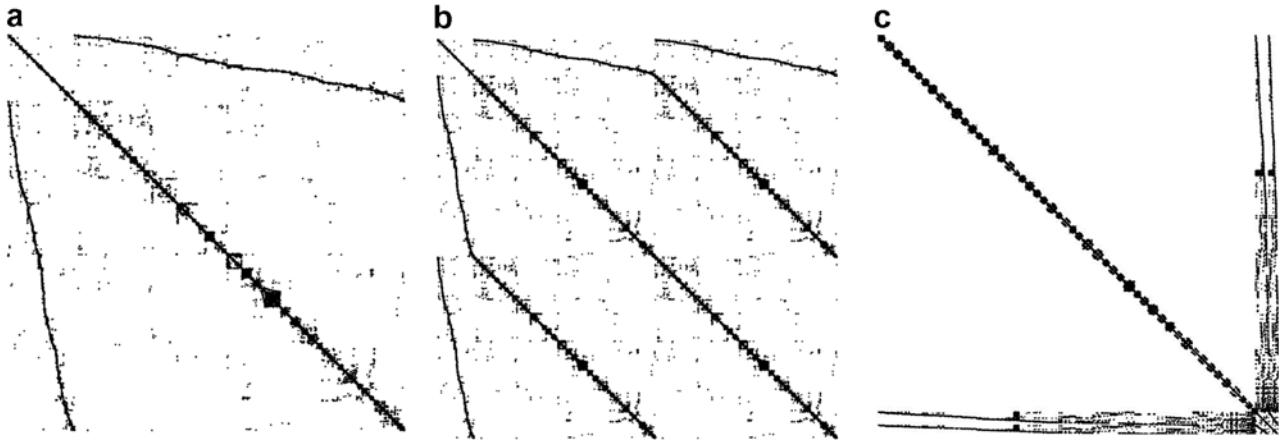

Fig. 3. Nonzero elements for the 7917-bus system: (a) original Y_{bus} matrix; (b) original Jacobian matrix and (c) DBBD Jacobian matrix.

$$\begin{Bmatrix} J_{11} & 0 & 0 & 0 & J_{1n} \\ 0 & J_{22} & 0 & 0 & J_{2n} \\ 0 & 0 & J_{33} & 0 & J_{3n} \\ 0 & 0 & 0 & \dots & \dots \\ J_{n1} & J_{n2} & J_{n3} & \dots & J_{nn} \end{Bmatrix} = \begin{Bmatrix} L_{11} & 0 & 0 & 0 & 0 \\ 0 & L_{22} & 0 & 0 & 0 \\ 0 & 0 & L_{33} & 0 & 0 \\ 0 & 0 & 0 & \dots & \dots \\ L_{n1} & L_{n2} & L_{n3} & \dots & L_{nn} \end{Bmatrix} \times \begin{Bmatrix} U_{11} & 0 & 0 & 0 & U_{1n} \\ 0 & U_{22} & 0 & 0 & U_{2n} \\ 0 & 0 & U_{33} & 0 & U_{3n} \\ 0 & 0 & 0 & \dots & \dots \\ 0 & 0 & 0 & \dots & U_{nn} \end{Bmatrix} \quad (10)$$

where

$$J_{kk} = L_{kk}U_{kk}$$
$$U_{kn} = L_{kk}^{-1}J_{kn}$$
$$L_{nk} = J_{nk}U_{kk}^{-1},\ \textit{for}\ k \in [1, n-1]$$
$$L_{nn}U_{nn} = J_{nn} - \sum_{k=1}^{n-1} L_{nk}U_{kn}$$

Fig. 4. LU factorization of the DBBD Jacobian matrix.

factorization algorithm described in [16], which is then followed by parallel forward reduction and backward substitution [17]. The factored LU matrix produced by this algorithm still retains the DBBD form, as shown in Eq. (10) of Fig. 4; it demonstrates inherent parallelism in the forward reduction and backward substitution phases as well. The calculations of L_{kk}, U_{kk}, L_{nk} and U_{kn} for different k's (i.e., 3-block groups) are independent of each other. So we can distribute different 3-block groups to different processors to be factored in parallel, with no data exchanges until the factorization of J_{nn}. This is the reason that we try to maximize the number of diagonal blocks in the matrix reordering procedure. The last block, J_{nn}, requires data produced in all the right and bottom border blocks, so its factorization is the last step. Before factoring the last block, pairs of the already factored border blocks are first multiplied in parallel to produce $J_{nn}^k = L_{nk}U_{kn}$, for $k \in [1, n-1]$. The summation of the $n-1$ products obtained for the different values of k and the addition of its results to J_{nn} is then carried out in a binary tree fashion in parallel and the results are sent to the processors assigned to the factorization of the last diagonal block (for a highly parallel approach). The last block becomes denser after all the partial results are applied to it and its factorization require a significant amount of time. Thus, the most computation-intensive part of the entire problem can be solved efficiently in parallel. Another advantage of the DBBD ordering is that the results can be used repeatedly for different values of the right hand side in the linear system and the 3-block groups assigned to each processor remain the same as long as the network topology does not change.

4. Parallel solution of Newton's power flow equations

We summarize here our parallel DBBD Newton algorithm for power flow analysis based on the above parallel LU factorization approach.

Step 1. Use the heuristics-based node tearing algorithm to reorder the Y_{bus} matrix into the DBBD form and sort the 3-block groups $\{Y_{ii}, Y_{in}, Y_{ni}$, for $i \in [1, n-1]\}$ according to their computation cost (by estimating the number of floating-point operations based on the number of nonzero elements);

try to get the best partitioning results according to our rules discussed earlier. Then, renumber the buses in the original network file according to the selected partitions so that the ordered DBBD matrix has continuous bus numbering; the purpose of the renumbering is to speedup the remaining steps. This preprocessing step is performed on a PC.

Step 2. Assign the relevant bus data along with the 3-block group(s) in the Y_{bus} matrix to the processors. The bus data for the cutting nodes is copied into every processor in order to subsequently minimize the number of data collisions during processor communication.

Step 3. Initialize in parallel the voltages to a flat voltage start.

Step 4. Evaluate Eqs (1) and (2) and calculate $\triangle P$ and $\triangle Q$ in (3) in parallel using 3-block groups, bus data and voltages (they are updated at every iteration). Then, every processor checks to see if all $\triangle P$ and $\triangle Q$ in its assigned bus range are sufficiently small (we set the tolerance at 0.001 p.u.); it sends its decision to a control processor, which may stop the iterative procedure based on the information reported by all the computing processors.

Step 5. Form in parallel the 3-block groups $\{J_{ii}, J_{in}, J_{ni}\}$ in the DBBD Jacobian matrix. In this step, every processor uses the data assigned to it in Step 1 and performs calculations on its assigned 3-block groups. J_{nn} is processed by the control processor.

Step 6. Apply in parallel LU factorization to the 3-block groups $\{J_{ii}, J_{in}, J_{ni}\}$; use the procedure described earlier that appeared in [16]. Also apply our dynamic load balancing techniques [24] during this procedure in order to increase the efficiency. We employ the control processor to monitor the load information of every processor and predict and assign jobs based on its capabilities, job costs related to computation and communication.

Step 7. Solve Eq. (3) in parallel by forward reduction and backward substitutions in order to derive the voltage corrections $\triangle\delta$ and $\triangle V$. Then, apply these corrections to the current voltage values.

Step 8. Go back to Step 4.

From this description, we can see that the most time-consuming steps are carried out in parallel and little inter-processor communication is needed.

5. Multiprocessor implementation on a configurable chip

Fig. 5 shows our configurable multiprocessor architecture designed for the Newton's method. The processing elements (PEs) form binary trees to support communication patterns in our algorithm (as discussed in Section 3 and [16,17]). Each PE is guided by the SC (system controller) that utilizes the boot-up code stored in the PE's local program memory (LPM). An interrupt-driven control channel in a star configuration connects the SC to every PE. There is also a direct communication channel between the SC and every binary tree root. Our binary tree network for data communications reduces global transfers and is also scalable in size. All the required interconnection between on-chip memories and/or processors is implemented based on the multi-mastering, fully connected AVALON® bus of Altera. Every PE in our system has a local, exclusive on-chip program memory and a shared on-chip data memory. Each PE shares its on-chip data memory with its sibling and parent in the binary tree. All PEs share the on-board synchronous SRAM (SSRAM) memory; an access takes at least four clock cycles on the average. The SC also pre-fetches instructions and data from the on-board memory into the PEs; the PEs use on-chip memory to run the application code because of its much lower latency, which is only one clock cycle, compared to the on-board SRAM memory.

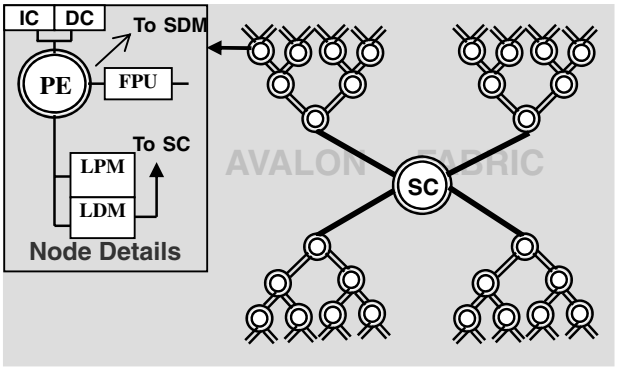

Fig. 5. Our multiprocessor architecture. SC: System Controller; PE: Processing Element; IC: Instruction Cache; DC: Data Cache; LDM: Local Data Memory; SDM: Shared on-board memory; LPM: Local Program Memory and FPU: Floating-point Unit.

Our MPoPC design employs an Altera configurable processor IP core (i.e., Nios) optimized for configurable logic to implement each PE. Processor cores of this type greatly empower FPGA-based system implementations. With user-defined configuration of the data bus width (16 or 32 bits), register file size, peripherals, multipliers, on-chip memory size and type (RAM or ROM), and memory address map, we can fine-tune our design to better match the application and also speedup the design of multiprocessors. The Nios® RISC processor is fully configurable in the aforementioned features and its implementation yields over 200 DMIPS (Dhrystone MIPS (Millions Instructions per Second)) in Altera Stratix II FPGAs. It utilizes a 5-stage pipeline and conforms to a modified Harvard memory architecture. Configurable processors necessitate trade-offs between performance and the consumed resources. A typical Nios® processor without an FPU consumes about 1600 logic elements (LEs) in the APEX20KE device. Our multiprocessor targets large matrix operations in general that require floating-point arithmetic for dynamic data

Please cite this article as: Xiaofang Wang et al., Parallel solution of Newton's power flow equations ..., Int J Electr Power Energ Syst (2006), doi:10.1016/j.ijepes.2006.10.006

ranges; we implemented in hardware a pipelined IEEE 754 single-precision FPU inside every PE. Our FPU runs at 128.3 MHz for the 3-stage adder/subtractor, 150.8 MHz for the 5-stage multiplier and 165.4 MHz for the 28-stage divider. These efficient realizations result in significant performance improvements for matrix operations [16,17]. Other functions that we implemented in hardware and used as custom instructions are sine and cosine; look-up tables for them were built. A software implementation of the sine/cosine function with Nios takes more than 300,000 clock cycles while our hardware implementation takes only six cycles.

6. Relevance to other work

We focus here on a comparison with other DBBD-related parallel algorithms. Several parallel implementations of the power flow analysis problem by FDPF based on DBBD partitioning have been reported, where the target matrices for the LU factorization are symmetric and smaller than the corresponding Jacobian matrices [6,7,23]. In contrast, the Jacobian matrices in our work are nonsymmetric. We have not found any important literature on the parallel solution of the exact Newton's method with parallel DBBD LU factorization. In former publications the total number of independent blocks in the DBBD Y_{bus} bus matrix (i.e., of size $n-1$) is limited by the number of available processors, which is normally small. For large power systems with thousands of buses, the resulting 3-block groups are still very sparse. It is also very difficult in this situation to balance the processor work loads. Since these approaches often target PC clusters or other loosely coupled systems with high communication latencies, more independent blocks will incur more communication overhead at the end of the procedure, thus limiting the performance. Good performance is only observed for a small number of processors.

Since all the processors in our system are embedded into a single chip and we also employ an architecture optimized for the application, our system presents very low communication costs to our algorithm. The system architecture can even be changed at run time, as needed, to match the dynamically changing characteristics of the application by taking advantage of the reconfigurability of FPGAs. Another major contribution of our approach is the application of our dynamic load balancing techniques [24] in the parallel LU factorization of large DBBD Jacobian matrices where load imbalance is a major bottleneck that can inadvertently affect the performance [25].

7. Experimental results and performance analysis

We implemented our parallel power flow analysis algorithm on our configurable multiprocessor for the benchmark matrices shown in Table 1. The SOPC development board employed for the implementation of the multiprocessor is populated with the largest APEX20KE FPGA device, the EP20K1500EBC652-1x, which includes 51,840 logic elements and 442,368 bits of on-chip memory. The system frequency is 50 MHz and seven processors with hardwired FPUs fit into the FPGA device.

We observed in our experiments that the chosen matrix partitioning solution can have a tremendous impact on the execution time of parallel LU factorization that dominates the solution of power flow equations. We first simulated the parallel LU factorization of DBBD Jacobian matrices for a wide range of matrix partitioning results; this way, we produced a near-optimal partitioning for each power benchmark. For example, Fig. 6 (a) shows the general trend in the number of cutting nodes (N_c) and the number of independent diagonal blocks ($n-1$) produced when increasing *MaxNodes* (the maximum number of nodes allowed in each sub-network) for the 1648-bus network. The relative execution time of parallel LU factorization based on different partitioning results in Fig. 6(a) is shown in Fig. 6(b). A near-optimal partitioning of the 1648-bus system can be observed when *MaxNodes* is 120, since it results in the smallest execution time. It is clear from Fig. 6(b) that the choice made in the network partitioning phase can have a big impact on the LU factorization and equation solution times. Table 3 shows results of near-optimal partitioning for our benchmark systems. The near-optimal partitioning results highly depend on the individual physical characteristics of the power system. This also shows the necessity for load balancing, especially for large power systems, such as the 1648- and 7917-bus systems; they produce very irregular blocks and nonzero patterns. For the two small cases (i.e., the 57 and 118-bus systems), we choose the total number of independent blocks to be the same as the number of processors.

Our experiments also show that, with an increase in the network size (N) the 3-block groups tend to be more sparse. Minimum degree ordering [26] or a similar technique could be applied inside these matrix blocks to reduce the fill-ins during the LU factorization and also to balance the processor loads. However, we do not apply here such a technique since we do not intend to mix the effects of hardware techniques with those of software techniques. We focus on the performance of this new parallel computing platform for the power flow problem.

Table 4 shows the execution times of LU factorization, forward reduction and backward substitution (including communication times) for the chosen benchmark systems; the partitioning results shown in Table 3 were employed. The corresponding execution times for Newton's power flow solution are listed in Table 5. Table 5 also presents uni-processor solution times for the DBBD algorithm. The obtained speedups are very good for our 7-processor parallel system. The speedup is also system-dependent. For example, for the 7917-bus system the speedup is lower than for the 300- and 1648-bus systems because of a larger number of cutting nodes; the process associated with these nodes becomes a bottleneck. From Tables 4 and 5, we can deduce that for a large system the time spent on the LU

Please cite this article as: Xiaofang Wang et al., Parallel solution of Newton's power flow equations ..., Int J Electr Power Energ Syst (2006), doi:10.1016/j.ijepes.2006.10.006

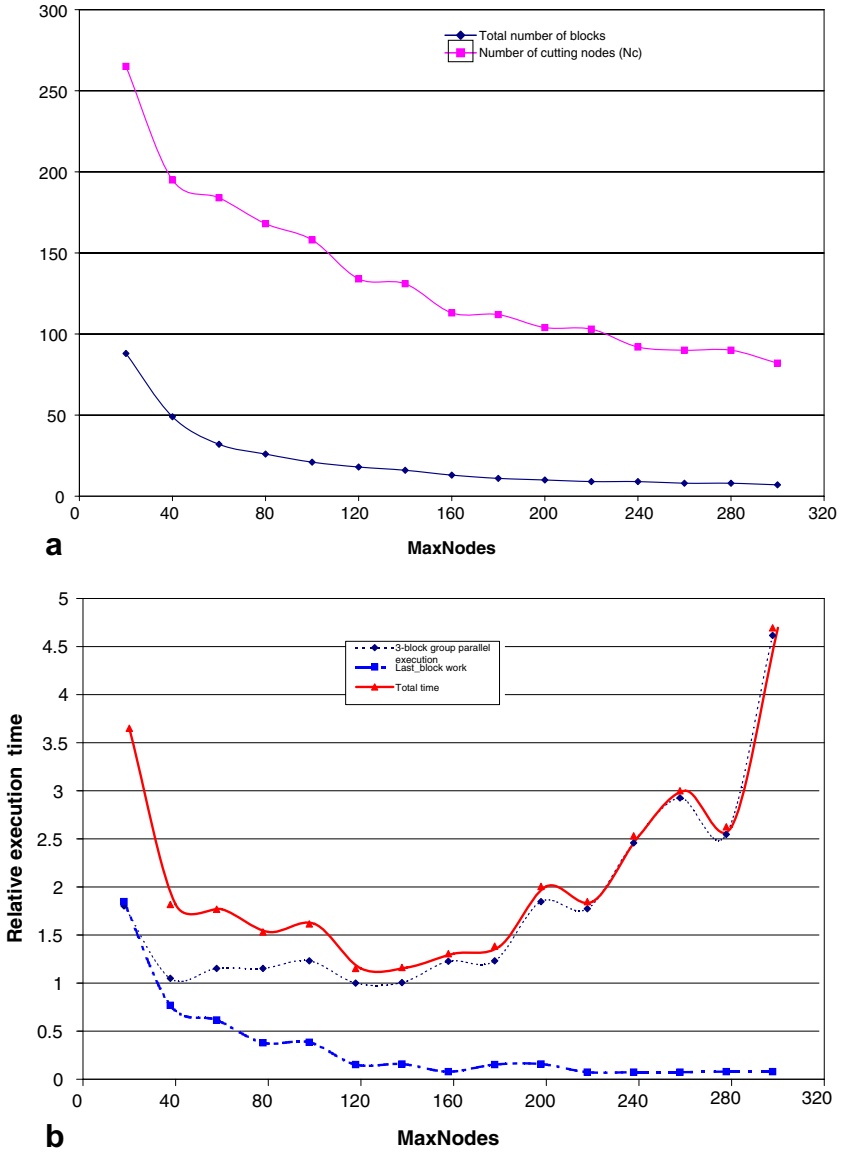

Fig. 6. Impact of network partitioning on the execution time of parallel LU factorization for the DBBD Jacobian matrix of the 1648-bus system: (a) the effect of *MaxNodes* on network partitioning and (b) relative execution time of parallel LU factorization for the DBBD Jacobian matrix.

Table 3
Optimal partitioning of the Y_{bus} matrices of the benchmark systems

Dimensionality of admittance matrix (Y_{bus})	57	118	300	1648	7917
Maximum nodes in a block	7	18	16	120	150
Number of independent diagonal blocks	7	7	21	18	67
Minimum dimensionality of independent diagonal blocks	4	11	6	33	15
Maximum dimensionality of independent diagonal blocks	7	18	16	120	150
Dimensionality of the last block	12	12	42	134	541
Size distribution of independent diagonal blocks[a]	5×7[b] 6, 4	18, 18, 17, 16, 14, 12, 11	5×9, 6×16, 15, 3×14, 2×12, 3×10, 6	120, 109, 99, 3(90), 5(85), 79, 5(75), 33	7(150),17(130), 10(120), 13(100), 19(90), 1(15)

[a] 19(30) stands for 19 blocks of size close to 30×30.
[b] 5×7 stands for 5 blocks of size 7×7.

Please cite this article as: Xiaofang Wang et al., Parallel solution of Newton's power flow equations ..., Int J Electr Power Energ Syst (2006), doi:10.1016/j.ijepes.2006.10.006

Table 4
Execution times (ms) to solve the linear equations for the benchmark systems on our configurable multiprocessor with seven processors

Benchmark systems	57-bus	118-bus	300-bus	1648-bus	7917-bus
Dimensionality of the Jacobian matrix	106	181	530	2982	14,508
LU factorization of the Jacobian matrix	13.42	39.11	479.12	7425	107,391
Forward reduction	0.56	1.12	6.61	102.1	3210.7
Backward substitution	0.59	1.30	8.81	109.3	3291.5
Total execution time	14.57	41.53	494.54	7636.4	113,893

Table 5
Execution times (s) for Newton's power flow equations with seven processors

Benchmark systems	57-bus	118-bus	300-bus	1648-bus	7917-bus
Iterations	4	4	5	5	6
Total time	0.069	0.198	2.582	39.21	712.4
Uni-processor	0.425	1.148	15.75	247.8	4210.3
Speedup	6.16	5.79	6.10	6.32	5.91

Tolerance: 0.001 p.u.

Table 6
Comparison of the execution times (sec) on the EP20KE1500 and EP2S180 (predicted) FPGAs

Benchmark systems	57-bus	118-bus	300-bus	1648-bus	7917-bus
EP20KE1500	0.069	0.198	2.582	39.21	712.4
ES2S180	0.0256	0.073	0.396	5.80	92.89
Improvement	2.7	2.7	6.52	6.76	7.67

factorization dominates the total execution time, which justifies our effort and time spent on ordering the Y_{bus} and Jacobian matrices.

The EP20K1500E, which was introduced in 1999, is a relatively slow device compared to recently released FPGAs. For example, the Stratix II EP2S180 device [15] with 9,383,040 bits of on-chip memory and 384 hardware multipliers can accommodate 23 copies of our processor with a system frequency of more than 135 MHz. We should then expect the execution times to be much better than those in Table 5. Unfortunately, a development board with such a device is not available in the market yet. Nevertheless, we are interested in predicting the performance of the application on state-of-the-art FPGAs. The predicted execution times for the five benchmarks with 23 processors on the EP2S180 are compared in Table 6 with the measurements (Row 3 in Table 5) on the EP20KE1500. The performance improves with an increase in the problem size. Many processors are idle during the execution of the first two benchmarks due to insufficient jobs; i.e., the number of the independent blocks is less than the number of processors.

8. Conclusions

The importance of real-time power flow monitoring and reaction necessitates the introduction of new cost-effective parallel processing platforms that can be viable in the long run. Taking advantage of recent advances in silicon-based configurable technologies, our research focuses on developing low-cost specialized parallel machines to speedup computation-expensive applications related to power applications. Good performance can only be achieved if there is a good match between the algorithm and the chosen computer architecture. We proposed a parallel solution to Newton's power flow equations based on a novel ordering technique for the Jacobian matrix; it exploits structural similarities between the bus admittance and the Jacobian matrices. The algorithm was benchmarked on our MPoPC system that was implemented on the Altera SoPC board. We also demonstrated that a pre-processing procedure to find a near-optimal partitioning of a given power matrix is necessary for large systems in order to maximize the performance. The good results demonstrate that our MPoPC approach is viable and promising in power engineering. The importance of our results is further exacerbated by Moore's Law that predicts continuous significant transistor density increases per chip every year. Our parallel machine and the parallel DBBD technique can also be used to solve other power engineering problems, such as transient analysis.

There are still many hardware and software issues to explore and many existing research results have to be reexamined under this new, configurable computing approach. For example, among other issues, we may take advantage of the re-configurability of FPGAs to dynamically change the interconnection and configuration of the processors at run time; we may even use heterogeneous processors to better match the applications. Finally, a better memory interface and I/O channel will definitely increase the throughput of the system and boost its frequency. These issues will be investigated in future research.

Please cite this article as: Xiaofang Wang et al., Parallel solution of Newton's power flow equations ..., Int J Electr Power Energ Syst (2006), doi:10.1016/j.ijepes.2006.10.006

Acknowledgements

This work was supported in part by the US Department of Energy under Grant DE-FG02-03CH11171. The authors also would like to thank Mr. Xin He for his implementation of a preliminary version of the partitioning algorithm.

References

[1] Tinney WF, Hart CE. Power flow solution by Newton's method. IEEE Trans PAS 1967;PAS-86 (11):1449–60.
[2] Stott B, Alsac O. Fast decoupled load flow. IEEE Trans PAS 1974;93:859–69.
[3] IEEE Committee Report. Parallel processing in power systems computation. IEEE Trans Power Syst 1992;7(2): 629–38.
[4] Falcão DM. High performance computing in power system applications. In: Proceedings of the Second Intl Meet Vector Paral Proces (VECPAR'96), Porto, Portugal, September 1996.
[5] Bell G, Gray J. What is next in high-performance computing? Commun ACM 2002;45 (2):91–5.
[6] Tu F, Flueck AJ. A message-passing distributed-memory parallel power flow algorithm. IEEE Power Eng Soc (PES) Winter Meet 2002;1:211–6.
[7] Chen SD, Chen JF. A novel approach based on global positioning system for parallel load flow analysis. Int J Elect Power Energy Syst 2005;27 (1):53–9.
[8] Gupta A. Recent advances in direct methods for solving unsymmetric sparse systems of linear equations. ACM Trans Math Soft 2002;28 (3):301–24.
[9] Duff IS, Direct methods, Technical Report RAL-98-056, Rutherford Appleton Laboratory, Oxfordshire, UK, 1998.
[10] Demmel JW, Gilbert JR, Li XS. An asynchronous parallel supernodal algorithm for sparse gaussian elimination. SIAM J Matrix Anal Appl 1999;20 (4):915–52.
[11] Fu C, Jiao X, Yang T. Efficient sparse LU factorization with partial pivoting on distributed memory architectures. IEEE Trans Parall Distr Syst 1998;9 (2):109–25.
[12] Bomhof W, van der Vorst HA. A parallel linear system solver for circuit simulation problems. Numer Linear Algebra Appl 2000;7:649–65.
[13] Ziavras SG. On the problem of expanding hypercube-based systems. J Parall Distr Comput 1992;16 (1):41–53.
[14] Ziavras SG. RH: a versatile family of reduced hypercube interconnection networks. IEEE Trans Parall Distri Syst 1994;5 (11):1210–20.
[15] Altera Corp., http://www.altera.com.
[16] Wang X, Ziavras SG, Parallel LU. factorization of sparse matrices on FPGA-based configurable computing engines. Concur Comp: Practice Exper 2004;16 (4):319–43.
[17] Wang X, Ziavras SG. Parallel direct solution of linear equations on FPGA-based machines. In: Proceedings of the Seventieth IEEE International Parallel and Distributed Processing Symposium IPDPS2003 (Eleventh IEEE International Workshop on Parallel and Distributed Real-Time Systems). Nice, France, April 22–26, 2003.
[18] Grainger JJ, Stevenson Jr WD. Power system analysis. McGraw Hill Publ; 1994.
[19] Sangiovanni-Vincentelli A, Chen LK, Chua LO. An efficient heuristic cluster algorithm for tearing large-scale networks. IEEE Trans Circuit Syst 1977;24 (12):709–17.
[20] Compton K, Hauck S. Reconfigurable computing: a survey of systems and software. ACM Comput Surveys 2002;34 (2):171–210.
[21] Xilinx Corp., http://www.xilinx.com.
[22] Underwood K, FPGAs vs. CPUs: trends in peak floating-point performance. In: Proceedings of the 12th ACM/SIGDA international symposium on field programmable gate arrays, Monterey, CA, February 2004:171–80.
[23] Koester DP, Ranka S, Fox GC. Parallel block-diagonal-bordered sparse linear solvers for electrical power system applications. IEEE Proc Scal Parall Lib Conf 1994.
[24] Wang X, Ziavras SG.A multiprocessor-on-a-programmable-chip reconfigurable system for matrix operations with power-grid case studies. Int J Comput Sci Eng, Special Issue on Parallel and Distributed Scientific and Engineering Computing, in press.
[25] Chai JS, Bose A. Bottlenecks in parallel algorithms for power system stability analysis. IEEE Trans Power Syst 1993;8 (1):9–15.
[26] George A, Liu JWH. The evolution of the minimum degree ordering algorithm. SIAM Rev 1989;31 (1):1–19.

Please cite this article as: Xiaofang Wang et al., Parallel solution of Newton's power flow equations ..., Int J Electr Power Energ Syst (2006), doi:10.1016/j.ijepes.2006.10.006