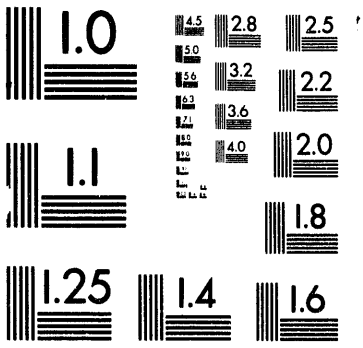
1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
6.3
7.1
8.0
9.0

1 of 1

PNL-SA-23064

ITERATIVE METHODS FOR THE WLS STATE ESTIMATION ON RISC, VECTOR, AND PARALLEL COMPUTERS

J. Nieplocha
C. C. Carroll[a]

October 1993

Presented at the
1993 North American Power Symposium
October 11-12, 1993
Washington, D.C.

Work supported by
the U.S. Department of Energy
under Contract DE-AC06-76RLO 1830

Pacific Northwest Laboratory
Richland, Washington 99352

(a) University of Alabama
Tuscaloosa, Alabama

Iterative Methods for the WLS State Estimation on RISC, Vector, and Parallel Computers

J. Nieplocha†
Pacific Northwest Laboratory*
Richland, WA 99352

C. C. Carroll
The University of Alabama
Tuscaloosa, AL35487

Abstract

We investigate the suitability and effectiveness of iterative methods for solving the weighted-least-square (WLS) state estimation problem on RISC, vector, and parallel processors. Several of the most popular iterative methods are tested and evaluated. The best performing preconditioned conjugate gradient (PCG) is very well suited for vector and parallel processing as is demonstrated for the WLS state estimation of the IEEE standard test systems. A new sparse matrix format for the gain matrix improves vector performance of the PCG algorithm and makes it competitive to the direct solver. Internal parallelism in RISC processors, used in current multiprocessor systems, can be taken advantage of in an implementation of this algorithm.

Introduction

The state estimation program is a crucial component of the electric energy management and control system. It processes on-line telemetered and pseudomeasurement data to provide dependable estimates of the power system state vector [1,2]. The measurements, which include voltage magnitudes, both real and reactive line flows and nodal power injections, are corrupted with errors. The weighted-least-square (WLS) algorithm is the most widely used state estimation method. However, the linear-programming based state estimation has been recently receiving much attention due to its bad data rejection property [3-6].

The real-time state estimation requires a fast and dependable implementation. As technological limits in speed of the sequential computers are being approached, there is a growing need for algorithms suitable for parallel processing. Several years ago, the power industry recognized this problem and a very significant effort has been made to develop such algorithms [7]. Recently, the Dantzig-Wolfe algorithm was proposed for decomposition and parallel processing in the LP-based state estimation [8, 9]. Much more has been done in this direction for the WLS estimator, which has traditionally incorporated direct methods based on the LDU, LU, or Cholesky factorizations for solving large and sparse systems of linear equations in each iteration of the state estimation algorithm. Direct methods perform very well on sequential computers. However, they demonstrate limited speedups and performance on parallel computers due to sequential character of the operations involved. Existing state estimation algorithms may be divided into two classes: suboptimal and optimal.

Suboptimal methods are based on 'tearing' the system into several subsystems, and involve a hierarchical processing [1.III, 4, 10-19]. State estimation is performed at the subsystem level independently and then results are adjusted at the coordination level. The solution is not optimal but the subsystem level may be executed on different processors concurrently.

Optimal methods in WLS state estimation are based on transformations and repartitioning of the original problem which preserve optimality of the solution [19, 20] or may be based only on parallelization of the original direct solver. The latter approach employs data domain decomposition (block schemes) or task decomposition (elemental schemes); also a hybrid technique which combines the both methods is possible [21]. The amount of parallelism in the direct solver may be improved by reordering/repartitioning of the gain matrix [22, 23] (which is not optimal with respect to the number of fill-ins generated in the factorization process [23, 24]), yet is rather limited according to the precedence relationship graph in factorization and substitutions [25]. Some new algorithms attempt to minimize the precedence relationships using multiple factorization [26] or sparse inverse factors [27-30].

Most publications to date have concentrated on simulation techniques only to show expected performance of these parallel algorithms. Few papers provide experimental results for machines ranging from the Intel iPSC hypercube, used for parallel factorization and substitutions [22], and the MIT Tagged Token Dataflow Architecture, which can take advantage of the fine-grain parallelism in parallel LU factorization [31], to vector computers like the Cray-1 [32] or the Cray/XMP [33] for solving fast decoupled load flow, which also involves solving large and sparse systems of linear equations.

Iterative methods are widely used for solving large, sparse systems of linear equations generated when finite differences methods are applied in the solution process of partial differential equations [34-36]. Some iterative methods possess much higher levels of parallelism than the direct ones. In this work, the most frequently used iterative methods are tested for solving sparse systems of linear equations in the WLS state estimation problem. The best performing method, precondi-

* Pacific Northwest Laboratory is operated for the U.S. Department of Energy (DOE) by Battelle Memorial Institute under contract DE-AC06-76RLO 1830.

† The author did this work while at The University of Alabama.

tioned conjugate gradient, has a very good level of coarse-grain parallelism. Following the trends in the parallel computer technology–the use of Reduced Instruction Set Computing (RISC) processors and/or powerful vector processors in massively parallel multiprocessor systems like: the Intel Paragon, the Thinking Machines CM-5, or the Fujitsu VPP500–this algorithm is implemented and tested on a RISC, vector, and parallel computer.

In the following section, the WLS state estimation problem is outlined. Then, the most popular basic and acceleration iterative methods are described and tested for the WLS state estimation problem on the Cray/XMP using the ITPACK package [38]. The convergence and execution time results are presented. Later section describes details of our optimized implementations and results for the PCG algorithm on a RISC, vector, and parallel computer. A new sparse matrix format for storage of the gain matrix is introduced to improve efficiency of the PCG algorithm on vector computers. The tests are performed for the WLS state estimation of the IEEE 30- and 118-bus systems. For the purpose of comparison, performance results for the direct solver are also included.

The WLS State Estimation

The nonlinear equations relating the measurements to the system state can be written as:

$$z = h(x) + e \tag{1}$$

where:

- z - measurement vector of dimension m;
- x - state vector of dimension $(n=2N - 1) \times m$;
- N- number of buses;
- e - vector of measurement errors;
- h - vector of non-linear functions which relate the states to the measurements.

The truncated Taylor expansion of (1) yields:

$$z = h(x^{\vartheta}) + H(x^{\vartheta})\,\Delta x + e, \tag{2}$$

where x^{ϑ} is the linearization point, and

$$H(x^{\vartheta}) = \left[\partial h(x_j^{\vartheta}) / \partial (x_j^{\vartheta})\right]. \tag{3}$$

The weighted-least-square state estimation problem is formulated as the minimization of the following form:

$$J(x) = [z - h(x)]^T R^{-1} [z - h(x)], \tag{4}$$

where the covariance matrix of the noise is:

$$R = E(ee^T) = \begin{cases} \sigma^2 \text{ if } i=j \\ 0 \text{ otherwise.} \end{cases} \tag{5}$$

The state estimate can be found by iteratively solving the following equation:

$$A(x^{\vartheta})\,(x^{\vartheta+1} - x^{\vartheta}) = H^T(x^{\vartheta}) R^{-1} [z - h(x^{\vartheta})], \tag{6}$$

where A denotes the gain matrix:

$$A(x^{\vartheta}) = H^T(x^{\vartheta}) R^{-1} H(x^{\vartheta}). \tag{7}$$

Iterative Solution of Systems of Linear Equations in WLS State Estimation

To avoid confusion, for the purpose of this paper, from now on, an iteration of the WLS state estimation will be referred to as a "cycle" while the "iteration" will be used in context of the iterative solver.

In each cycle of WLS state estimation, a large and sparse system of linear equations is being solved:

$$A\Delta x = b, \tag{8}$$

where the matrix A is the symmetric-positive-definite (SPD) gain matrix and equ. (8) is another form of equ. (6).

The direct solver which has traditionally been employed in the WLS state estimation is based on either the Gaussian elimination, LU, LDU, or Cholesky factorization, followed by the backward and forward substitutions. Iterative methods may be used as an alternative approach to direct methods. They are frequently used for solving partial differential equations (PDE) on vector and parallel computers [35, 36]. Iterative methods could be divided into two classes: basic and acceleration methods.

Basic iterative methods

Basic iterative methods are the Richardson, Jacobi, Gauss-Seidel, Successive Over-Relaxation (SOR), and Symmetric SOR (SSOR) methods. They may be represented as:

$$u^{k+1} = Gu^k + d, \tag{9}$$

where u^k is the approximation of the solution vector in the k-th iterative step, G is the iterative matrix given as:

$$G = I - Q^{-1}A, \tag{10}$$

where Q is the splitting matrix and $d=Q^{-1}b$.

For the Richardson method, $Q=I$. For the Jacobi method, $Q=D$(A) contains diagonal elements of matrix A. The splitting matrix for the Gauss-Seidel method is:

$$Q = D(A) - C_L, \tag{11}$$

where C_L is the strictly triangular lower part of matrix A. However, the Gauss-Seidel method is implemented as a different form of iteration of equation (9) using the iterative matrix for the Jacobi method, see [34]. The Successive Over-Relaxation method is derived from the Gauss-Seidel method by introducing a relaxation factor ω. The value of ω is $0<\omega<2$. The method when $\omega<1$ is said to be underelaxed, for $\omega>1$ is overrelaxed, and when $\omega=1$ it reduces to the Gauss-Seidel. The iterative matrix is modified in such a way that diagonal elements of G are premultiplied by $(1-\omega)$, other elements and d are premultiplied by the ω factor. Similarly to forward/backward substitutions, the SOR method is essentially sequential; however, it may be implemented very efficiently on parallel computers as an asynchronous algorithm [37].The Symmetric Successive

Over-Relaxation is basically identical to the SOR method but each iteration consists of forward and backward sweep.The splitting matrix for the SSOR method is:

$$Q = \frac{\omega}{2-\omega}\left(\frac{1}{\omega}D - C_L\right)D^{-1}\left(\frac{1}{\omega}D - C_U\right). \qquad (12)$$

Unlike the SOR method, the SSOR can be accelerated using the Chebyshev polynomial or conjugate gradient accelerations.

Sequence of u vectors converges to the true solution of system (8) if the spectral radius of G is less than one [34].

In initial studies, performance of the basic iterative methods for solving WLS state estimation problem was tested using the ITPACK [38] and NSPCG [39] packages. The system of linear equations corresponding to the first cycle of the WLS state estimation for the IEEE 30-bus system was used. The stopping number was $\zeta = 10^{-6}$. (Throughout the paper, the sparse problem formulation was employed.) The following table presents performance results of basic iterative methods (time given in CPU seconds) on the Cray/XMP using the ITPACK package.

Table 1: Performance of basic iterative methods.

Method	Richardson	Jacobi	Gauss-Seidel	SOR	SSOR
iterations	diverged	diverged	> 5000	2782	> 5000
CPU time	-	-	> 4.79	2.66	> 6.3

The best overrelaxation factor for the SOR was ω=1.75, which was found using a linear search technique. Convergence of the Gauss-Seidel and SSOR was very slow so that the iterative process was terminated after exceeding 5000 iterations, while the Richardson and Jacobi methods diverged. As could be seen from Table 1, basic iterative methods performed poorly.

Acceleration methods

In many cases, convergence of the basic iterative methods can be accelerated using the polynomial or conjugate gradient acceleration [34]. Methods in the conjugate gradient class also include: GMRES, ORTHOMIN, ORTHO-DIR, ORTHORES, and their Lanczos versions [40].

The conjugate gradient method was originated by Hestenes and Stiefel [41] for solving SPD systems of linear equations. While they employ iterative solution methods, CG methods can be considered direct solvers since they theoretically yield exact solutions within a finite number of steps. If roundoff error could be discounted, the CG method would be guaranteed to converge to the exact solution in at most N iterations, where N is the dimension of the linear system (8). Despite its interesting properties, the method was largely ignored after its initial introduction because of the computational expense required for convergence. This handicap was eliminated when it was discovered that the CG method could be used with a preconditioned system of equations to effect convergence in far fewer than N iterations. The CG method has since become popular because of its capacity for efficiently solving sparse matrix problems, its suitability for implementation on vector and parallel computers, and developments that allow it to be generalized for use with nonsymmetric matrices. Several modifications to the Hestenes and Stiefel's original method [41] have been proposed that make the method suitable for nonsymmetrizable matrix problems, see [40]. Practically, the convergence rate depends on the condition number, the ratio of the largest to smallest eigenvalue, of the matrix, A. When the matrix's condition number is minimized, the method usually converges much faster than in N iterations. The condition number can be minimized by premultiplying both sides of equation (8) by the inverse of a preconditioner matrix, Q:

$$Q^{-1}A\,\Delta x = Q^{-1}b \qquad (13)$$

to yield a new system to be solved:

$$\hat{A}\Delta x = \hat{b}. \qquad (14)$$

The matrix Q should have a simple structure and approximate the gain matrix in order that the new coefficient matrix might have a smaller condition number.

Three methods exist for preconditioning an equation system: the matrix as an approximation of A^{-1}, as an incomplete Cholesky or LDL^T decomposition, and as a splitting matrix from an iterative technique. Ortega and Voigt [35] give a synopsis of each of these strategies in a useful survey of preconditioners for CG methods and their implementation on vector and parallel computers. When a splitting matrix is used, the CG method is referred to as accelerator of the iterative method for which the splitting matrix was used as a preconditioner. The conjugate gradient method can be employed only for solving SPD and mildly nonsymmetric linear systems.

Algorithm: **Preconditioned Conjugate Gradient**

1. Input
   Initial guess of x i.e., u^0
   Stopping number ζ
2. Compute
   $\delta^0 = b - u^0$
   Solve linear system $Qz^0 = \delta^0$ for z^0
   Set $p^0 = z^0$ and $a_0 = (z^0, \delta^0)$
3. For $k = 1, 2, \ldots$
   Compute $c^{k-1} = Ap^{k-1}$
   Set $\lambda_{k-1} = a_{k-1}/(p^{k-1}, c^{k-1})$
   Take $u^k = u^{k-1} + \lambda_{k-1}p_{k-1}$
   Compute $\delta^k = \delta^{k-1} - \lambda_{k-1}c_{k-1}$
   Solve for z^k the linear system $Qz^k = \delta^k$
   Check for convergence.
   If stopping number $< \zeta$ then STOP else

Set $a_k = (z^k, \delta^k)$

Set $\alpha_k = a_k / a_{k-1}$

Compute $p^k = z^k + \alpha_k p^{k-1}$

As before, performance of the PCG method was tested using ITPACK. Test results for different preconditioners are given in Table 2. The CG method with the SSOR preconditioner performed best. The best overrelaxation factor ω was in the range <0.95,1>.

Table 2: Performance of the preconditioned conjugate gradient.

Preconditioner	none	Jacobi	SSOR
number of iter.	229	115	55
CPU time	0.210	0.107	0.0977

The Incomplete Cholesky Factorization (ICF) algorithm, as implemented in ITPACK and NSPCG packages, breaks down for this particular problem. Moreover, the Chebyshev polynomial acceleration method [33] with the Jacobi or SSOR, although nondivergent, failed to converge for the same stopping number.

As can be seen in Table 2, the CG method without preconditioner required many more iterations than the system dimension (N=59 for IEEE 30-bus system) to converge. This can be explained by the large condition number of the gain matrix in the WLS state estimation problem, see Table 3.

Table 3: Condition number of the gain matrix

WLS cycle	1	2	3	4
30-bus system	$2.705 \cdot 10^5$	$3.919 \cdot 10^5$	$3.859 \cdot 10^5$	$3.859 \cdot 10^5$
118-bus system	$7.712 \cdot 10^5$	$8.803 \cdot 10^5$	$9.035 \cdot 10^5$	$9.042 \cdot 10^5$

Initial studies also included the use of different scaling techniques for the gain matrix and right-hand-side vector. Scaling was successfully used in the LP state estimation to alleviate the problem of round-off errors that reduce accuracy and increase amount of computations [42, 8, 43]. The gain matrix and the right-hand-side vector were scaled, and then the iterative solver was applied to solve the scaled problem. However, none of the scaling techniques described in [42] helped to improve the convergence.

Implementation Considerations

Performance of the PCG algorithms for solving large sparse systems of linear equations on vector and parallel machines depends on the efficiency with which data structures are handled [44, 46]. Often, to get an iterative solution to converge, one has a choice between many "cheap" iterations or a few expensive iterations. Iterative procedures such as Jacobi or SOR belong to the first class while PCG methods belong to the second. Likewise preconditioners for CG methods may be so classified. The Jacobi preconditioner is very cheap while the SSOR or Incomplete Cholesky preconditioners are expensive in their CPU time consumption. In general, some algorithms may be better suited to a particular computer than others, and such issues as data structures and parallelization must be considered when a particular algorithm is selected.

In tested problems, 118 and 472 measurements for the IEEE 30-bus and 118-bus systems were used, respectively.

Initial studies showed that among iterative algorithms, the PCG with the Jacobi and SSOR preconditioners works the best in the WLS state estimation. In the next step, this algorithm was implemented on RISC, vector, and parallel processors. Even in a single processor implementation, the CG with the Jacobi preconditioner gave better results than with the SSOR or Incomplete Cholesky Factorization (ICF) (which did not fail when implemented according to a different formula [45] than one used in ITPACK and NSPCG) preconditioners. Moreover, since unlike in many applications where iterative solvers are used, here, the structure of the matrix A is not regular, the SSOR and ICF preconditioners exhibit sequential nature, and therefore, are not well suited for vector/parallel implementation [46].

Iterative solvers compute approximations to the solution vector until a stopping criterion is met. In this application, the following stopping test was determined to work the best:

$$\zeta = \frac{\| u^{(k+1)} - u^{(k)} \|_2^2}{\| u^{(k+1)} \|_2^2}. \qquad (15)$$

Of course, a lower value of the stopping number leads to a better accuracy of the iterative solution but also increases the amount of computations. In each cycle of WLS state estimation, updates of the state vector as well as values of the mismatch vector become smaller in magnitude as the Newton-Raphson algorithm converges. It was found that the stopping number for the iterative solver should be lower in the initial cycles of the state estimation procedure. Comparing values of the state estimates for the IEEE 118-bus system calculated using iterative and direct solvers, the following values of the stopping numbers for iterative solver in consecutive cycles of the state estimation procedure were determined as providing an adequate accuracy: 10^{-6}, $3 \cdot 10^{-6}$, $8 \cdot 10^{-6}$, $2 \cdot 10^{-5}$. These values were employed for both the IEEE 30- bus and 118-bus systems on all computers used in this work. Because implementation and accuracy of floating point computations differ from one computer to another, in each case, a slightly different number of iterations were performed.

The accuracy of the state estimates computed using the iterative solver was determined separately for voltages and angles comparing the appropriate values to these obtained using the direct solver on the same computer. The relative accuracy was calculated in percents using the infinity norm:

$$Max\left(\left| \frac{x_{iterative} - x_{direct}}{x_{direct}} \right| \right) \cdot 100\%, \qquad (16)$$

that is, as the largest absolute value of the ratio of mismatches between estimates for direct and iterative solvers divided by the value of this estimate.

In the next three sections implementations of the Jacobi-PCG on RISC, vector, and parallel computers are discussed. The performance of this algorithm is compared to the performance of the direct solver, the LDL^T factorization with forward and backward substitutions. For the direct solver, to reduce the number of fill-ins in the factorized matrix, the system was reordered using the Tinney scheme two.

Implementation on a RISC processor

The preconditioned Jacobi-CG method is very well suited for RISC processors like the IBM RS/6000 used in this study. Architectures of RISC processors allow execution of more than one instruction per clock cycle [47]. This can be done by taking advantage of internal parallelism of these processors, pipelining, and processor instruction overlap scheduling. RS/6000 has independent branch, fixed, and floating point units, which can work in parallel [48]. Presence of Floating Multiply and Add (FMA) instruction, which executes $a*b+c$ operation (two floating point operations) in one clock cycle, makes implementation of vector products or vector SAXPY operations—straightforward but computationally expensive elements of the CG algorithm—very efficient. In order to improve processor overlap scheduling, loop unrolling to depth three was used in the vector operations of the PCG algorithm. This technique helps the compiler to generate a more efficient code suitable for pipeline execution [48].

Most modern computers use cache—very fast but limited in size—memory. On the RS/6000 access to a word takes one or eight clock cycles depending on whether the word is located in cache or main memory. Operation of RISC processors can be very seriously degraded if inside DO loops data is accessed with a large stride causing frequent cache misses. Sparse matrix-vector product in CG algorithm as well as factorization and substitutions in direct solver use indirect addressing that translates into varying in size strides, access to nonadjacent memory locations, and less efficient execution.

The performance of direct solver, the LDL^T factorization with forward and backward substitutions with and without reordering of the gain matrix, is shown in Table 4. The performance of iterative solvers is independent of ordering since fill-ins are not generated.

Direct solver is about twice as fast as the Jacobi-CG for the IEEE 118-bus system, see Table 4 and 5. However, comparing the performance on traditional CISC and RISC processors, see Table 9 and 10, it can be noticed that performance gap between the iterative and direct solver has narrowed more than twice after migration to the RISC platform. RISC processors are much more powerful than CISC processors (processors of the IBM RS/6000, Model 320 and Sequent Symmetry have identical clock cycle) and are recently used in many parallel computers. This performance gap could be easily eliminated providing that the iterative solver exhibits a significantly better level of parallelism than the iterative solver. It is not uncommon that some algorithms that are slower than the best sequential algorithms perform much better on parallel machines.

Table 4: CPU time for direct solver on the RS/6000.

WLS cycle	IEEE 30-bus system		IEEE 118-bus system	
	w/o ordering	with ordering	w/o ordering	with ordering
1	0.033	0.017	0.649	0.217
2	0.033	0.017	0.633	0.200
3	0.033	0.017	0.649	0.200
4	0.033	0.017	0.620	0.200
sum	0.10	0.068	1.951	0.817
reordering	-	0.017	-	0.067

Table 5: Performance of PCG on the RS/6000.

WLS cycle	IEEE 30- bus system		IEEE 118-bus system	
	iter.	time	iter.	time
1	93	0.050	143	0.400
2	92	0.050	153	0.433
3	81	0.033	196	0.533
4	46	0.017	123	0.333
sum:	-	0.15	-	1.7
relative accuracy	Voltage	Angle	Voltage	Angle
	0.03%	0.09%	0.006%	0.02%

Implementation on a vector computer

On a vector computer, like the Cray/XMP, the code of iterative solvers vectorizes much better than the code of direct solvers. However, special consideration must be given to the data structures to implement sparse-matrix vector product operation efficiently. Standard rowwise sparse matrix format leads to short innermost loops because their length corresponds directly to the number of nonzero elements per row in the gain matrix. It limits performance of this operation because the length of vector registers on the Cray/XMP is 64 and on others like the Hitachi S820, Convex C-1, or NEC SX/2 even longer [47].This can be alleviated by using a different matrix format. One such a format, developed at Purdue and used as the primary format in NSPCG package [39], is very well suited for solving PDEs on vector computers using the PCG method. The matrix is represented with two two-dimensional arrays. First one accommodates nonzero elements of the matrix and the other corresponding column indices. The size of these arrays is the number of unknowns times the maximum number of nonzero elements in a row. After transformation of DO loops, the innermost has length equal to the number of unknowns and vectorizes very well. However, because the number of nonzero

elements per row in the gain matrix varies a lot (from 3 to 43 elements for the IEEE 118-bus system), there is a significant amount of time wasted processing the extra zero-valued elements. Here, this problem was solved by developing of a new sparse matrix format. The corresponding code for the PCG algorithm vectorizes well and the overhead is reduced.

New Sparse Matrix Format

Instead of using a rectangular array, the sparse matrix is stored in blocks resembling stairs. The height of each consecutive block is a multiplicity of the vector register size, *V*, which varies on different supercomputers. This provides the lowest possible start-up cost of vector operation per element in the innermost DO loop [47]. The width is equal to the number of nonzero elements in the last row of the gain matrix, stored in this block minus width of the previous blocks. This format requires sorting out in increasing order and correspondingly renumbering elements of the state vector according to the number of nonzero elements per row in the gain matrix.

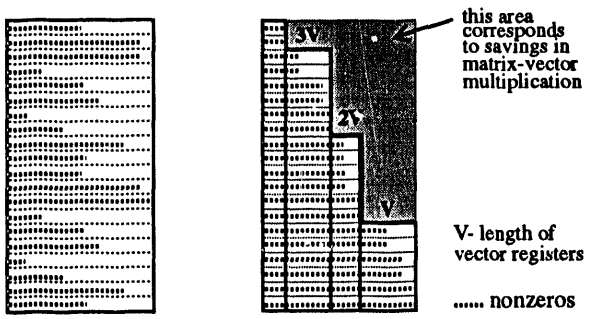

Fig. 1: Storage of nonzero elements in sparse matrix formats suitable for vectorization.

The performance of the matrix vector product operation using this new matrix format versus standard nonsymmetric (with and without vectorization) and Purdue formats was tested for the gain matrix for IEEE 30-bus and 118-bus systems. Results given in Table 6 show superiority of this new format over the others.

Table 6: CPU time for a single sparse matrix - vector product operation using different matrix formats on the Cray.

	std. nonvector.	std. vectorized	Purdue	new fmt.
30-bus system	$2.495 \cdot 10^{-4}$	$1.972 \cdot 10^{-4}$	$1.213 \cdot 10^{-4}$	$9.138 \cdot 10^{-5}$
118-bus system	$1.154 \cdot 10^{-2}$	$7.922 \cdot 10^{-4}$	$4.755 \cdot 10^{-4}$	$3.236 \cdot 10^{-4}$

The introduced sparse matrix format improved overall performance of the PCG algorithm as compared to the vectorized direct solver, see Tables 7-8. For the IEEE 30-bus system, direct solver is faster than the PCG but not for a larger 118-bus system where the PCG algorithm takes advantage of increased efficiency of vectorization in longer DO loops. It should be noted that, the CPU time for the iterative solver includes time spent to transform the standard sparse matrix storage of the gain matrix into the new format, which operation, of course, could be avoided in a production-type code.

Table 7: CPU time for direct solver on the Cray.

WLS cycle	30-bus	118-bus
1	$5.908 \cdot 10^{-3}$	$7.979 \cdot 10^{-2}$
2	$6.041 \cdot 10^{-3}$	$7.909 \cdot 10^{-2}$
3	$6.036 \cdot 10^{-3}$	$7.875 \cdot 10^{-2}$
4	$6.032 \cdot 10^{-3}$	$7.868 \cdot 10^{-2}$
sum	$2.402 \cdot 10^{-2}$	0.3163
reordering	$4.656 \cdot 10^{-3}$	$3.177 \cdot 10^{-2}$

Table 8: Performance of PCG on the Cray.

WLS cycle	IEEE 30- bus sys.		IEEE 118-bus sys.	
	iter.	time	iter.	time
1	78	$8.796 \cdot 10^{-3}$	140	$4.768 \cdot 10^{-2}$
2	65	$7.686 \cdot 10^{-3}$	161	$6.181 \cdot 10^{-2}$
3	63	$8.139 \cdot 10^{-3}$	187	$6.675 \cdot 10^{-2}$
4	50	$6.595 \cdot 10^{-3}$	116	$4.221 \cdot 10^{-2}$
sum:	-	$3.121 \cdot 10^{-2}$	-	0.2184
reordering	-	$3.053 \cdot 10^{-3}$	-	$1.976 \cdot 10^{-2}$
relative accuracy	Voltage	Angle	Voltage	Angle
	0.045%	0.13%	0.0051%	0.0078%

The PCG with Jacobi preconditioner appears to be very well suited for multiprocessor vector supercomputers since the code can be not only vectorized but also parallelized. This however, was not done in this work because of lack of access to the Cray/XMP-24 (having 2 processors only) in the dedicated mode that is required to perform reliable accounting when multitasking is used. Performance of the parallel PCG was tested on another parallel machine instead.

Implementation on parallel processors

The machine used in the tests was the Sequent Symmetry with 24 CISC processors and shared memory configuration, with 64-bit-wide bus. Each processor has 64KB of 2-way set associative write-back cache and floating point Weitek 1167 accelerator [47]. The WLS state estimation program was ported to the Sequent Symmetry and the PCG solver was parallelized based on the data domain decomposition approach.

Most existing power system network decomposition schemes [49, 50] are more oriented toward direct solvers especially using block schemes [21] where size of the interconnection area limits amount of parallelism in the algorithm. Here, the decomposition was accomplished by dividing the data into equal blocks assigned to different processors [44] what corre-

sponds to 'tearing' of the network into equally sized clusters and mapping them to different processors. Since amount of computations in all the operations of the PCG algorithm except sparse matrix vector multiplication—where the number of nonzero elements, related to the cluster size and connectivity, determines computational complexity—is directly proportional to the cluster size, this provided a good load balancing. The differences in execution times between processors were always less than 6%. However, the shared memory bus traffic related to the number of tielines was not optimized. It was probably the main source of performance degradation when more than eight processors for IEEE 118-bus system were used, see Figure 2.

Parallel execution was implemented with the *fork* and *join* schemes. From the main program, the same code of the iterative solver was forked for each processor. Communication was accomplished via shared memory and synchronization using barriers. The PCG algorithm requires synchronization when the vector product operation is computed. Each processor performs this operation on the appropriate parts of the vectors stored in its local memory. After this is done, the partial results are combined into the final result—each processor reads the numbers computed by the other processors and adds them up. The result is stored and used locally. Multiple-write conflict is avoided since each partial result is stored in a shared memory variable but written only by a single processor. The PCG with Jacobi preconditioner has a coarse-grain parallelism since the vector product operation occurs only twice per iteration.

Table 9: CPU time for direct solver on the Sequent.

WLS cycle	30-bus sys.	118-bus sys.
1	0.083	1.017
2	0.067	1.017
3	0.067	1.017
4	0.067	1.033
sum	0.284	4.083
reordering	0.017	0.167

Table 10: Performance of sequential PCG on the Sequent.

WLS cycle	IEEE 30- bus system		IEEE 118-bus system	
	iter.	time	iter.	time
1	92	0.600	143	4.483
2	91	0.583	151	4.600
3	82	0.533	195	5.933
4	48	0.317	132	4.133
sum	-	2.033	-	19.149
relative accuracy	Voltage	Angle	Voltage	Angle
	0.029%	0.096%	0.0063%	0.013%

Tables 9 and 10 show performance of sequential direct and sequential iterative solvers, respectively. Difference in performance between them is larger than on the RISC processor.

Table 11: CPU times and speedups in parallel PCG for 30-bus system

cycle⇨	1		2		3		4		total time	avg. Sp.
Proc.	time	Sp.	time	Sp.	time	Sp.	time	Sp.		
2	0.35	1.714	0.35	1.667	0.3	1.777	0.167	1.899	1.17	1.76
4	0.2	3.0	0.2	2.916	0.183	2.914	0.116	2.715	0.70	2.89
8	0.153	3.922	0.150	3.812	0.133	4.749	0.067	4.133	0.50	4.13

Table 12: CPU times and speedups in parallel PCG for 118-bus system

cycle⇨	1		2		3		4		total time	avg. Sp.
Proc.	time	Sp.	time	Sp.	time	Sp.	time	Sp.		
2	2.316	1.936	2.383	1.930	3.183	1.864	2.166	1.908	10.05	1.91
4	1.183	3.789	1.266	3.632	1.600	3.708	1.150	3.509	5.12	3.68
8	0.667	6.725	0.750	6.133	0.900	6.512	0.566	7.294	2.88	6.67
12	0.516	8.689	0.566	8.128	0.716	8.279	0.466	8.886	2.26	8.49
16	0.450	9.962	0.483	9.871	0.633	9.373	0.400	10.33	1.97	9.89

The performance of parallel PCG solver is shown in Tables 11-12 and Figure 2. The tables include the longest CPU time of a processor and the speedup (calculated with respect to time results for the sequential implementation of the PCG) in each cycle of the WLS state estimation, total CPU time to solve the problem and average speedup (overall) on 2, 4, 8, 12, and 16 processors. These results demonstrate a very high level of parallelism of this algorithm as applied to the WLS state estimation problem.

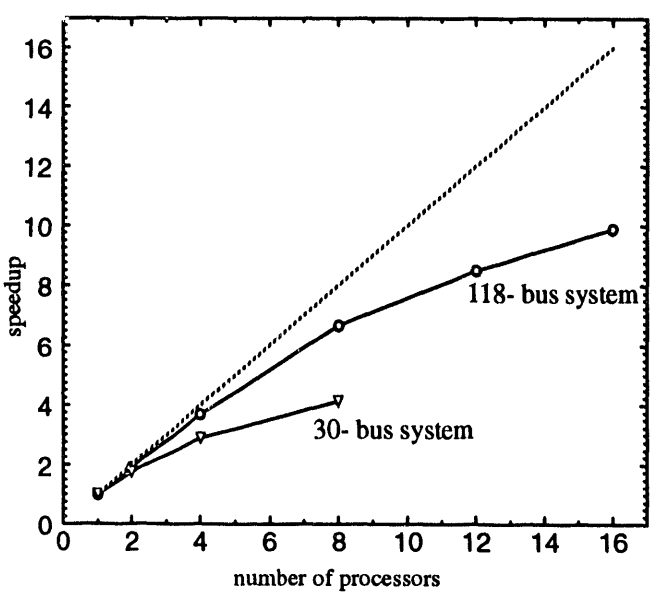

Fig. 2: Speedup for the PCG solver for the IEEE 30- and 118- bus systems.

Conclusions

Parallel direct solvers for the power systems applications have been studied for a long time. In the current work, we investigate the use of an alternative approach, iterative methods, for this purpose. It was found that among iterative methods, which have been tested for solving the WLS state estimation problem, the preconditioned conjugate gradient performs best. This algorithm was implemented on RISC, vector, and parallel computers. Although the iterative solver is still slower than the direct methods on a single scalar processor, this performance gap has very much narrowed after migration from a traditional Complex Instruction Set Computing (CISC) to a RISC platform, and has been completely eliminated on a vector processor. Moreover, the PCG algorithm appears to have a much better potential for parallel execution than direct solvers (based upon results reported in the literature, for example [22]). This should make it faster than parallelized direct solvers on multiprocessor RISC or/and vector systems. However, we feel that still more research needs to done. For example, further research in this area might be directed into developing a better, application-specific, preconditioner for the CG method to accelerate its convergence.

Acknowledgments

This research was supported in part by the NSF under grant ECS-8907742. Authors would like also to acknowledge support of Argonne National Laboratory and the Alabama Supercomputer Center.

References

[1] Schweppe, F.C., Wildes, J. and Rom, D.B., 'Power system static state estimation I,II,II', IEEE Trans. PAS, Vol. 89, No. 1, Jan. 1970.

[2] Larson, R.E., Tinney, W. F., Hadju, L.P. and Piercy, D.S, 'State estimation in power systems, Parts I and II', IEEE Trans. PAS, Vol. 89, No. 3, March 1970.

[3] Irving, M.R., Owen, R.C. and Sterling, M.J.H.,'Power system state estimation using linear programming', IEE Proc.,Vol. 125, No. 9, Sept. 1978.

[4] Falcao, D.M., Cooke, P.A. and Brameller, A., 'Power systems tracking state estimation and bad data processing', IEEE Trans. PAS, Vol. PAS-101, Vol. 101, No. 2, Feb. 1982.

[5] Falcao, D.M. and de Assis, S.M., 'Linear programming state estimation: Error analysis and gross error identification', IEEE Trans. Power Systems, Vol. 3, No. 3, August 1988.

[6] Abur, A., 'A bad data identification method for linear programming state estimation', IEEE Trans. Power Systems, Vol.5, No.3, August 1990.

[7] IEEE Committee Report, 'Parallel Processing in Power Systems Computation, IEEE Trans. Power Systems', Vol. 7, No. 2, May 1992.

[8] El-Keib A. A., Nieplocha, J., Singh, H. and Maratukulam, D.J., 'A decomposed state estimation technique suitable for parallel processor implementation', IEEE Trans. Power Systems, Vol. 7, No. 3, August 1992.

[9] Abur A. and Celik M.K.,'Multi-area linear programming state estimator using Dantzig-Wolfe Decomposition', 10th Power Syst. Comput. Conf., Gratz, Austria, August 19-24,1990

[10] Schweppe, F.C., 'Power systems 2000: hierarchical control strategies', IEEE Spectrum, July 1978.

[11] Kobayashi, H., Narita., S. and Hamman, M.S.A., 'Model coordination method applied to power system control and estimation problems', Proc. IFAC/IFI 4th Int. Conf. on Digital Comp. Appl. Process Control, 1974, pp. 291-298.

[12] El-Fattah, Y.M., Ribbens-Pavella, M., 'Multi-level approach to state estimation in electric power systems', part I, Proc. 4th IFAC Symp. on Identif. Syst. Param. Estimation, Tbilisi, USSR, 1976.

[13] Van Cutsem, T., Howard, J.L., Ribbens-Pavella M. and El-Fattah, Y.M., 'Hierarchical state estimation', Electric Power and Energy Systems, Vol. 2, No. 2, April 1980.

[14] Wallach Y., Handschin, E. and Bonger, C., 'An efficient parallel processing method for power system state estimation', IEEE Trans. PAS, Vol. PAS-100, No. 11, Nov. 1981.

[15] Van Cutsem, T., Howard, J.L. and Ribbens-Pavella, M., 'A two-level state estimation for electric power systems', IEEE Trans. PAS, Vol. PAS-100, No. 8, Aug. 1981.

[16] Van Cutsem, T. and Ribbens-Pavella, M., 'Critical survey of hierachical methods for state estimation for electric power systems', IEEE Trans. PAS, Vol. PAS-102, No. 8, Aug. 1981.

[17] Kurzyn, M.S., 'Real time state estimation for large-scale power systems', IEEE Trans. PAS, Vol. PAS-102, No. 7, July 1983.

[18] Iwamoto S., Kusano M. and Quintana V.H.,'Hierarchical state estimation using a fast rectangular coordinate method', IEEE/PES Winter Power Meeting, Feb. 1989

[19] Seidu, K., Mukai, H., 'Parallel multi-area state estimation', IEEE Trans. PAS, Vol. PAS-104, No. 5, May 1985.

[20] Sassaki, H., Aoki, K. and Yokohama, R., 'A parallel computation algorithm for static state estimation by means of matrix inversion lemma', IEEE Trans. Power Systems, Vol. 2, No. 3, August 1987.

[21] Torralba, A., Gomez, A. and Franquelo L.G., 'Three methods for the parallel solution of a large, sparse systems of linear equations by multiprocessors', Proc. IASTED Power High Tech'89, Valencia, Spain, 1989.

[22] Lau, K., Tylavsky, D.J. and Bose, A., 'Coarse grain scheduling in parallel triangular factorization and solution of power systems matrices', IEEE/PES Summer Meeting, Minneapolis, July 1990.

[23] Abur A., 'A parallel scheme for the forward/backward substitutions in solving sparse linear systems', IEEE Trans Power Systems, Vol. 3, No. 4, November 1988.

[24] Tinney, W.R. and Walker, J.W., 'Direct solutions of sparse network equations by optimallly ordered triangular factorization', Proc. IEEE, Vol. 55, pp. 1801-1809, 1967.

[25] Tinney W.F. , Brandwajn, V. and Chan S.M., 'Sparse vector methods', IEEE Trans. PAS, Vol. 104, February 1985.

[26] Van Ness, J.E. and Molina G., 'Multiple factoring in the parallel solution of algebraic equations', EPRI-EL-3893, 1983.

[27] Alvarado, F.L, Yu, D.C., Betancourt, R., 'Partitioned A^{-1} methods', IEEE Trans. Power Systems, Vol.5, No.2, May 1990

[28] Betancourt, R. and Alvarado F.L., 'Parallel inversion of sparse matrices', IEEE Trans. Power Systems, Vol. 1, No.1, February 1986.

[29] Enns, M.K., Tinney, W.F. and Alvarado, F.L, 'Sparse matrix inverse factors', IEEE Trans. Power Systems, vol.5, No. 2, May 1990.

[30] Padilha, A. and Morelato, A., 'A W-matrix methodology for solving sparse network equations on multiprocessor computers', IEEE Trans. Power Systems, vol. 7, No. 3, May 1992.

[31] Yu, D.C. and Wang, H., 'A new parallel LU decomposition method', IEEE Trans. Power Systems, vol. 5, No. 1, February 1990.

[32] Calahan, D.A., 'Vectorized solution of load flow problems', EPRI-EL 566-SR, 1977.

[33] Gomez, A. and Betancourt, R., 'Implementation of the fast decoupled load flow on a vector computer', IEEE/PES Winter Meeting, Atlanta, 1990.

[34] Hageman, L.A., Young, D.M.,'Applied iterative methods', Academic Press, 1981.

[35] Ortega, J.M. and Voigt, R.G., 'Solution of partial differential equations on vector and parallel computers' , SIAM Rev 27, 1985.

[36] Rice, J.R., Boisvert, R.F., 'Solving elliptic problems using ELLPACK', Springer Verlag, New York, 1985.

[37] Nieplocha, J., Mai, T. and Carroll, C.C.,'Asynchronous Algorithms for Solving Large Systems of Linear Equations on Parallel Computers', in Parallel Computing: From Theory to Sound Practice, W. Joosen and E. Milgrom, Eds, IOS Press, 1992

[38] Young, D.M. and Mai, T., 'ITPACK-3A User's Guide', CNA-197, University of Texas at Austin, October 1986.

[39] Oppe, T.C., Joubert, W.D. and Kincaid, D.R.,'NSPCG User's Guide ', CNA-216, University of Texas at Austin, April 1988.

[40] Jea, K.C., Young, D.M., 'On the Simplification of Generalized Conjugate-Gradient Methods for Nonsymmetrizable Linear Systems', Linear Algebra Appl. 52/53:399-417, 1983

[41] Hestenes, M.R. and Stiefel E.L., 'Methods of Conjugate Gradient for Solving Linear Systems', Nat. Bur. Std. J. Res. 49, 409-436, 1952.

[42] Tomlin, J.A., 'On scaling linear programming problems', Mathematical Programming Study 4, 1975.

[43] Abur, A. and Celik, M.K.,'A robust WLAV state estimator using transformations',IEEE Trans Power Systems, Vol. 7, No. 1, Nov. 1988.

[44] Greenbaum, A., Li, C. and Chao, H.Z., 'Parallelizing conjugate gradient algorithm', Comp. Phys. Comm., 53, 1989.

[45] Kershaw, D.S., 'The Incomplete Cholesky - conjugate gradient method for the iterative solution of systems of linear equations', J. Comput. Phys., Vol. 26, No. 1, Jan. 1978.

[46] Ashcraft, C.C. and Grimes, R.G., 'On vectorizing incomplete factorization and SSOR preconditioners', SIAM J. Sci. Stat. Comput., Vol. 9, No. 1, 1988.

[47] Henessy, J.L. and Patterson, D.A., 'Computer Architecture A Quantative Approach', Morgan Kaufmann Publishers, 1990.

[48] Bell, R.,'IBM RISC System/6000 performance tuning for numerically intensive FORTRAN and C programs', IBM GG24-3611,1990.

[49] Sangiovanni-Vincentelli, A., Chen, L.K. and Chua, L.O.,'An efficient heuristic cluster algorithm for tearing large-scale networks', IEEE Trans. Circ. Sys., Vol. 24, No. 12, 1977.

[50] Irving, M.R. and Sterling, M.J.H.,'Optimal network tearing using simulated annealing', IEE Proc., Pt.C. 137, No. 1, 1990.

DATE
FILMED
2/7/94
END