Development of a Parallel Three-phase Transient Stability Simulator for Power Systems

Shrirang Abhyankar
Department of Electrical and Computer Engineering
Illinois Institute of Technology
3301 South Wabash Avenue
Chicago, Illinois - 60616
abhyshr@iit.edu

Alexander Flueck
Department of Electrical and Computer Engineering
Illinois Institute of Technology
3301 South Wabash Avenue
Chicago, Illinois - 60616
flueck@iit.edu

Xu Zhang
Department of Electrical and Computer Engineering
Illinois Institute of Technology
3301 South Wabash Avenue
Chicago, Illinois - 60616
xzhang70@iit.edu

Hong Zhang
Mathematics and Computer Science Division
Argonne National Laboratory
9700 South Cass Avenue
Argonne, Illinois
hzhang@mcs.anl.gov

ABSTRACT

This paper discusses the development of a parallel three-phase transient stability simulator that is built using the high performance computing library PETSc. Unlike the existing transient stability simulators that use a balanced per phase transmission network, the authors use a three phase transmission network. This three phase representation allows a more realistic analysis of unbalanced conditions due to untransposed transmission lines, widespread single phase loads and unbalanced disturbances. The goal of this paper is not to delve into the unbalanced analysis but rather present the scalability results of the newly developed parallel three-phase transient stability simulator. The performance of the proposed parallel three-phase transient stability simulator using Krylov subspace based iterative solver GMRES with two variants of Block-Jacobi preconditioner and a very dishonest preconditioning strategy is presented.

Categories and Subject Descriptors

I.6 [**Computing Methodologies**]: Simulation and Modeling; J.2 [**Computer Applications in Physical Sciences and Engineering**]: Engineering

General Terms

Algorithms, Design, Performance

Keywords

Transient stability, Three-phase network, Parallel processing, GMRES

HiPCNA-PG'11, November 13, 2011, Seattle, Washington, USA.

1. INTRODUCTION

Dynamic security assessment (DSA) of large-scale electrical power systems is done by existing transient stability simulators that use a balanced per phase model of the transmission network. Due to the balanced per phase assumption, unbalanced phenomena due to large single phase loads, unbalanced disturbances, untransposed transmission lines, single phase feeders, and single phase switching operations cannot be studied realistically.

A balanced transmission network model was used in the first developed transient stability analysis tools to make the computational problem tenable. Over the last decade there has been tremendous development in computing architectures enabling faster processor speed, bigger memory, and the ability to solve the problems using multiple processors. Moreover, there has been a lot of research on speeding up the solution of linear systems.

In recent years, power systems have grown in size and complexity due to network expansion and addition of more components to meet the ever-increasing demand. The deregulation of the power industry is motivating economic operation and thus operating the electrical network to its fullest potential and hence closer to stability limits. Under such circumstances the analysis of single phase disturbances becomes even more important.

We propose a three phase transient stability simulator which uses a full three phase network model of the transmission network. The ability to model all three phases can provide a more realistic picture of the dynamics of the individual three phases. When power systems operate at their limits, the margin between stability and instability is small and even single phase disturbances could trigger dynamic instability. Hence, such single phase disturbances need to be analyzed and a three phase transient stability simulator can provide details of the individual phase dynamics. Moreover, due to modeling of all the three phases, relay operations on individual phases can be analyzed and the relay settings for single phase tripping can be better ascertained.

The proposed three-phase transient stability simulator uses a spatial decomposition, or parallel-in-space, approach for distributing the differential-algebraic equations to different processors. The goal of this paper is to discuss the scalability results of this parallel implementation. Results obtained by using iterative solver GMRES, with two preconditioning schemes, are presented for three large-scale systems.

2. LITERATURE REVIEW

Transient stability simulation for large-scale power systems is a computationally onerous task due to the need to solve a large number of equations at each time step. A natural way to speed up this computation is to use parallel computing techniques, i.e., share the work load amongst multiple processors. Three algorithms for the parallel transient stability simulation have been proposed so far: parallel-in-space, parallel-in-time, and parallel-in space and time.

The parallel-in-space algorithms partition the given network into loosely coupled or independent subnetworks. Each processor is then assigned equations for a subnetwork. The partitioning strategy for the network division is critical for parallel-in-space algorithms to minimize the coupling between the subnetworks, i.e., to reduce the inter-processor communication, and balance the work load. Once a suitable partitioning strategy is selected the next key thing is the solution of the linear system in each Newton iteration. Several linear solution schemes have proposed in the literature, of which the prominent are the *Very Dishonest Newton Method* and *Conjugate gradient*. Reference [1] uses the very dishonest Newton method in which the factorization of the Jacobian matrix is done only when a certain fixed number of iterations are exceeded. Reference [2] decomposes the network equations in a Block Bordered Diagonal Form (BBDF) and then uses a hybrid solution scheme using LU and Conjugate gradient. Reference [3] solves the network by a block-parallel version of the preconditioned conjugate gradient method. The network matrix in [3] is in the Near Block Diagonal Form (NBDF).

The Parallel-in-time approach was first introduced in [4]. The idea of this approach was to combine the differential and algebraic equations over several time steps, create a bigger system and solve them simultaneously using the Newton method. All the equations for several integration steps are assigned to each processor.

Waveform relaxation methods [5]-[7] involve a hybrid scheme of space and time parallelization in which the network is partitioned in space into subsystems and then distributed to the processors. Several integration steps for each subystem are solved independently to get a first approximation [8]. The results are then exchanged and the process is repeated. The advantage of this scheme is that each subsystem can use a different integration step (multi-rate integration).

3. PROPOSED THREE-PHASE TRANSIENT STABILITY SIMULATOR (TS3PH)

The equations for the proposed three phase transient stability simulator with the network equations written in current balance form and the three phase network voltages and currents in rectangular form are as follows:

$$\frac{dx_{gen}}{dt} = f(x_{gen}, I_{dq}, V_{DQ,abc}) \tag{1}$$

$$0 = h(x_{gen}, I_{dq}, V_{DQ,abc}) \tag{2}$$

$$\begin{bmatrix} G_{3ph} & -B_{3ph} \\ B_{3ph} & G_{3ph} \end{bmatrix} \begin{bmatrix} V_{D,abc} \\ V_{Q,abc} \end{bmatrix} =$$

$$\begin{bmatrix} I_{genD,abc}(x_{gen}, I_{dq}) \\ I_{genQ,abc}(x_{gen}, I_{dq}) \end{bmatrix} - \begin{bmatrix} I_{loadD,abc}(x_{load}, V_{DQ,abc}) \\ I_{loadQ,abc}(x_{load}, V_{DQ,abc}) \end{bmatrix} \tag{3}$$

$$\frac{dx_{load}}{dt} = f_2(x_{load}, V_{DQ,abc}) \tag{4}$$

(1) and (2) represent the generator dynamics and the stator current equations. The three-phase network equation, in current balance form, is given by (3). Here, G_{3ph} and B_{3ph} are the real and imaginary parts of the complex three phase $\bar{Y}_{bus}$ matrix. The three phase complex $\bar{Y}_{bus}$ matrix can be built in a similar way as the per phase Y matrix. Each 3X3 block of $\bar{Y}_{bus}$ can be built either from the conductor geometry and line parameters, if available, or from the positive, negative and zero sequence line parameters. The later approach was used for building the three phase $\bar{Y}_{bus}$ matrix. The load equations can be either represented by differential equations for induction motor or dynamic load models, or by algebraic equations for static load models. If static load models are used then there are no separate variables for the loads but rather the load current injections are incorporated into the network equations.

Grouping all the dynamic variables together in one set and all the algebraic variables in another set, the TS3ph equations can be described by the differential-algebraic model:

$$\begin{aligned} \frac{dx}{dt} &= f(x, y) \\ 0 &= g(x, y) \end{aligned} \tag{5}$$

where

$$\begin{aligned} x &\equiv [x_{gen}, x_{load}]^t \\ y &\equiv [I_d, I_q, V_{D,abc}, V_{Q,abc}]^t \end{aligned}$$

The proposed three-phase transient stability simulator uses a fixed step implicit trapezoidal scheme for discretization as it is easy to implement and has numerical A-stablility properties. A literature survey on transient stability simulators also reveals that the implicit trapezoidal scheme is the prefered discretization scheme [1, 2, 3, 4, 8, 15]. One of the future developments for the three-phase transient stability simulator involves implementing more numerical integration schemes, both implicit and explicit with variable time stepping.

Using the implicit trapezoidal scheme the nonlinear algebraic equations to be solved are

$$\begin{aligned} &x(t + \Delta t) - x(t) - \\ &\frac{\Delta t}{2} \left(f(x(t + \Delta t), y(t + \Delta t)) + f(x(t), y(t)) \right) = 0 \end{aligned} \tag{6}$$

$$g(x(t + \Delta t), y(t + \Delta t))) = 0 \tag{7}$$

Equation (7) is then solved iteratively using Newton's method at each time step. The linear system to be solved in each Newton iteration is

$$\begin{bmatrix} J_{xx} & J_{xy} \\ J_{yx} & J_{yy} \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} = - \begin{bmatrix} F_x \\ F_y \end{bmatrix} \tag{8}$$

4. TS3PH PARALLEL IMPLEMENTATION

For the proposed three-phase transient stability simulator, we adopt a parallel-in-space approach for the decomposition of differential and network equations. The decomposition of the equations is based on the partitioning of the network equations since the network forms the only coupling. The generator and the load equations are naturally decoupled as they are only incident at a bus.

4.1 Partitioning

The ParMetis package [18] was used for doing the network partitioning. The ParMetis package [18] is available as a plug-in with the PETSc library. PETSc provides an interface for using the ParMetis package where the user calls PETSc interface routines to use the ParMetis package.

The Parmetis package requires an *Adjacency matrix* whose elements are 1s and 0s, where an element $A_{i,j}$ is 1 if vertex i is connected to vertex j. Along with the *adjacency matrix*, a weight can be also can be assigned to each vertex to account for different computational requirement. With vertex weights, ParMetis tries to minimize the edge cuts and also have the same amount of vertex weights in each sub-domain.

For TS3ph, the connection information from the single-phase Y_{bus} matrix of the network was used as the adjacency graph. Larger weights were assigned to the vertices having generators to account for the extra computation involved for the generator differential and algebraic equations.

4.2 Equations for each processor

Using the parallel-in-space approach, or spatial decomposition, each processor gets a subset of the generator, network and load equations of the complete network. The generator and the load equations are naturally decoupled since they are incident only on the local network bus and do not require communication with other processors. The network equations requires communication with other processors to compute the current mismatch equations.

The equations assigned to each processor at each time step are

$$F \equiv \begin{bmatrix} f^p(x^p, I^p_{dq}, V^p) \\ g^p(x^p, I^p_{dq}, V^p, V^{offproc}) \end{bmatrix} \tag{9}$$

where the superscript p denotes the processor number and $V^{offproc}$ means the network voltages required from other processors for computing g^p. The variables for each processor are

$$X^p \equiv \begin{bmatrix} x^p \\ I^p_{dq} \\ V^p \end{bmatrix}$$

4.3 Speeding up the linear solution process

4.3.1 Linear solver

Newton's method requires the solution of the linear system

$$J(x^i)\Delta x^{i+1} = -F(x^i) \tag{10}$$

where i is the iteration count. Solution of (10) can be either done by direct or iterative methods. From our experiments with direct and iterative solvers, we found that iterative solvers tend to be more scalable. Hence, we used Krylov subspace based iterative method GMRES (Generalized Minimum Residual) as the linear solver.

4.3.2 Preconditioning

The convergence of the Krylov subspace linear solvers depends on the eigenvalues of the operating matrix A and can be slow if the matrix has widely dispersed eigenvalues. Hence, to speed up the convergence, a preconditioner matrix P^{-1}, where P^{-1} approximates A^{-1}, is generally used. PETSc provides a variety of preconditioners and we experimented with two variations of the parallel Block-Jacobi preconditioner to optimize the iterative linear solution process.

The first preconditioning scheme tested was to use a single diagonal Jacobian block on each processor as the preconditioner. As an example, if the Jacobian matrix is distributed to two processors (0 and 1) as follows

$$\begin{matrix}[0] \\ [1]\end{matrix} \left[\begin{array}{cc} J_1 & J_2 \\ \hline J_3 & J_4 \end{array}\right]$$

then the parallel Block-Jacobi preconditioner is

$$\begin{matrix}[0] \\ [1]\end{matrix} \left[\begin{array}{cc} J_1^{-1} & \\ \hline & J_4^{-1} \end{array}\right]$$

The factorization of J_1 and J_4 can be done independently on each processor and no communication is required for building the preconditioner.

The second preconditioner that we experimented with is essentially a multiple-level Block-Jacobi preconditioner. The diagonal Jacobian block on each processor is further divided into smaller weakly coupled diagonal blocks. For example, if the Jacobian matrix on two processors is

$$\begin{matrix}[0] \\ [1]\end{matrix} \left[\begin{array}{ccc} J_{1a} & J_{1b} & \\ J_{1c} & J_{1d} & J_2 \\ \hline & J_3 & J_{4a} \quad J_{4b} \\ & & J_{4c} \quad J_{4d} \end{array}\right]$$

where the off-diagonal blocks J_{1b}, J_{1c} , J_{4b}, and J_{4c} represent the Jacobin part for weak coupling within each subnetwork, then the preconditioner with 2 blocks/processor would be

$$\begin{matrix}[0] \\ [1]\end{matrix} \left[\begin{array}{ccc} J_{1a}^{-1} & & \\ & J_{1d}^{-1} & \\ \hline & & J_{4a}^{-1} \quad \\ & & \quad J_{4d}^{-1} \end{array}\right]$$

For problems based on structured grids, as found in PDE simulations. a row-wise partitioning strategy would suffice. However, for the transient stability problem, we found a row-wise division to be inefficient since each processor has the generator equations set up first followed by the network equations. We used ParMetis to partition the diagonal Jacobian block into a larger number of weakly connected smaller diagonal blocks on each processor. We didn't extract the partitioning information but our conjecture is that the partitioning of the diagonal block further creates weakly connected subsystems, a topic which needs further exploration.

Building the preconditioner requires a factorization process, to compute the L and U matrices, and subsequent triangular solves. The numerical factorization phase, for large scale systems, is the most dominant operation and can constitute a large proportion of the total execution time. Hence, we use a strategy, called *Very Dishonest Preconditioner*, proposed in [10], of only updating the numerical factorization during the fault on/off time steps and reusing the preconditioner for all other time steps.

4.3.3 *Reordering scheme*

By reordering the rows and columns of a matrix, it may be possible to reduce the amount of fill-in created by LU factorization, thereby reducing time and storage cost. We experimented with various reordering strategies, available with PETSc, on the test systems to determine the optimal reordering strategy, i.e., the ordering scheme resulting in the least number of nonzeros in the factored matrix. The *Quotient Minimum Degree* ordering was found to be the most optimal approach for the systems that we tested.

5. BENCHMARKING AND PERFORMANCE RESULTS

The developed three-phase transient stability simulator was benchmarked on the WECC 9-bus system with the commercial positive sequence TS simulator PSS/E version 30 on a Windows Vista PC. The benchmarking scenario was a balanced three-phase fault applied on bus 5. The generators at buses 1, 2, and 3 are modeled as GENROU with an IEEET1 exciter model and all the loads are modeled as constant impedance loads. The benchmarking simulations were run for 3 seconds and the step size used for numerical integration was 1 cycle, i.e., 0.01667 seconds. Figures 1 and 2 show the TS3ph results match up nicely with the PSS/E results.

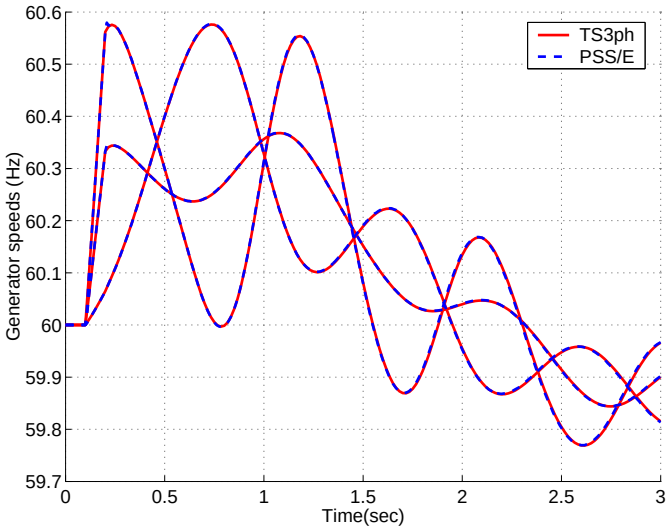

Figure 1: Generator speeds for a three phase fault on bus 5 from 0.1 sec to 0.2 sec

Three large power systems were created for testing TS3ph by duplicating the 118 bus system. To ensure that the individual 118 bus areas are connected, we used 5 randomly chosen tie lines between each area. Thus each 118 bus system area is connected to every other by 5 tie lines.

Table 1: TS3ph large-case test system inventory

Scale	Buses	Generators	Branches
10x	1180	540	2085
20x	2360	1080	4670
40x	4720	2160	11340

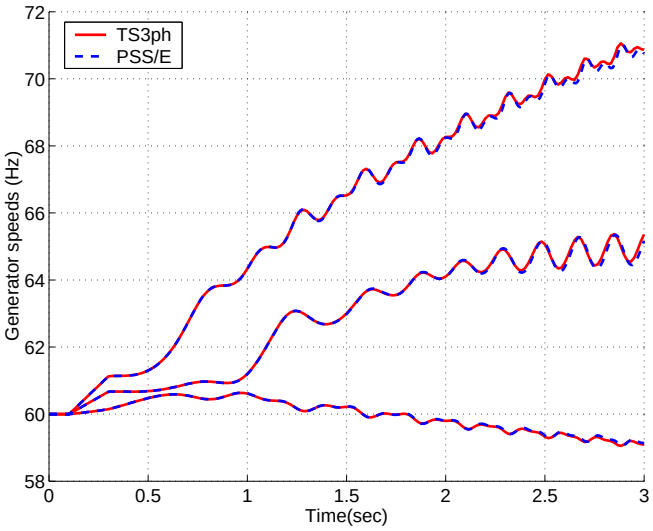

Figure 2: Generator speeds for a three phase fault on bus 5 from 0.1 sec to 0.3 sec

All the generators were modeled using the GENROU model with an IEEE Type 1 exciter model [19] with the loads modeled as constant impedance loads.

The parallel performance runs were done on a 16 node cluster with 2 processors on each node. Each processor is an Intel Xeon 2.53 GhZ with 4 cores per processor. Each core has 3 GB RAM giving a total of 24 GB RAM per node. The interconnecting network for this cluster is InfiniBand.

To compare the parallel performance results, the optimal solution strategy for simulation on a single processor was first determined. We found very dishonest preconditioned GMRES to be more efficient than a direct linear solver. We further experimented with LU and incomplete level-based LU factorization, ILU(p), as the preconditioners. For large-scale systems a full LU factorization requires more memory due to a larger number of extra fill-in elements. As such, most of the execution time is spent in fetching data from the main memory, rather than doing the computation, resulting in degrading the performance.

Hence, we used incomplete level-based LU factorization, ILU(p)) as the linear solver, which requires fewer fill-ins and subsequently less memory than complete LU. Fewer levels, with ILU(0) the cheapest, require less memory but could result in more triangular solves if the convergence is poor. We tested various levels and found ILU(6) as the optimal for our test cases. The reader is referred to [12], Chapter 6, for a detailed discussion and results for the simulations on a single core.

For all the parallel runs, we used GMRES as the iterative solver with very dishonest Block-Jacobi preconditioner (with 1 or more diagonal blocks/core), and a quotient minimum degree reordering for each block. ILU(6) on each block was used as the factorization scheme.

As seen from the results in tables 2 - 4, for the 1180 bus system, the maximum speed up obtained with Block-Jacobi preconditioner, with 1 block/core, was 1.33 on 4 cores, while the Block-Jacobi scheme with two 2 blocks/core on 3 cores

Table 2: TS3ph scalability results for 1180 bus system

Preconditioning strategy	# of cores	Total time (sec)	Speed up
ILU(6)	1	8.61	1.00
Block-Jacobi	2	9.75	0.88
Block-Jacobi	3	6.74	1.28
Block-Jacobi	**4**	**6.46**	**1.33**
Block-Jacobi	6	7.14	1.21
Block-Jacobi+2 blocks/core	2	9.93	0.87
Block-Jacobi+2 blocks/core	**3**	**6.39**	**1.35**

Table 3: TS3ph scalability results for 2360 bus system

Preconditioning strategy	#of cores	Total time (sec)	Speed up
ILU(6)	1	44.9	1.00
Block-Jacobi	2	37.1	1.21
Block-Jacobi	3	25.1	1.79
Block-Jacobi	**4**	**18.2**	**2.47**
Block-Jacobi	6	18.5	2.43
Block-Jacobi	8	22.5	2.00
Block-Jacobi+2 blocks/core	2	37.9	1.19
Block-Jacobi+2 blocks/core	3	25.3	1.78
Block-Jacobi+2 blocks/core	**4**	**19.0**	**2.36**

Table 4: TS3ph scalability results for 4720 bus system

Preconditioning strategy	# of cores	Total time (sec)	Speed up
Block-Jacobi	1	404	1.00
Block-Jacobi	2	216	1.87
Block-Jacobi	3	126	3.21
Block-Jacobi	4	62.5	6.46
Block-Jacobi	**6**	**44.5**	**9.08**
Block-Jacobi	8	60.3	6.69
Block-Jacobi+2 blocks/core	2	115	3.52
Block-Jacobi+2 blocks/core	3	411	0.98
Block-Jacobi+2 blocks/core	**4**	**42.0**	**9.61**

gave a speed up on 1.35. For the 2360 bus system, the Block-Jacobi preconditioner with 1 diagonal block/core yielded the best result with a speed up of 2.47 on 6 cores while a comparable speed up, 2.36, was attained with 2 diagonal blocks/core on only 4 cores.

The 4720 bus test case, which is the biggest one, shows considerable speed up with a superlinear speed up of 9 times with Block-Jacobi on 6 cores and 9.6 on 4 cores but with 2 blocks/core. This superlinear speed up can be attributed to the slow down in the sequential performance due to the need to access data from the main memory. For the 1180 and 2360 bus system all or most of the data can fit into the cache, for single core runs, and hence can be accessed much faster. The accessing of data, for the 4720 bus test case, requires fetching from the main memory since all the data cannot fit into the cache. This fetching of data from the main memory degrades the performance for the sequential run for the 4720 bus test case. Figure 3 shows the speed up performance versus the number of cores for each of the test systems.

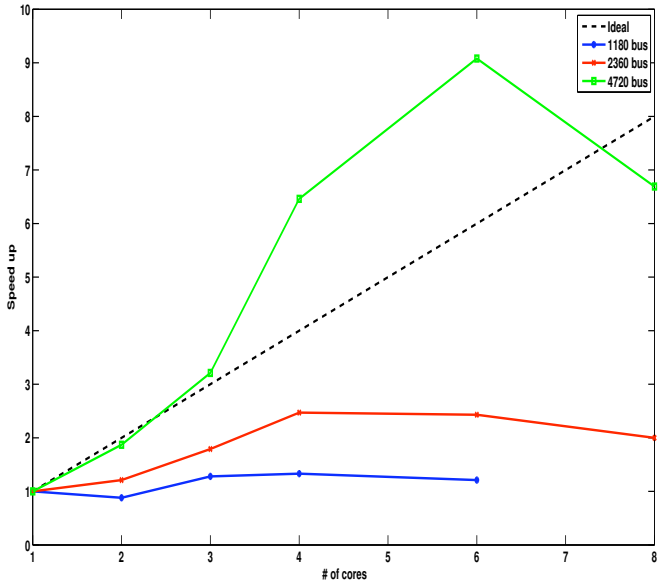

Figure 3: Speed up with Block-Jacobi preconditioner having 1 diagonal block per core

6. PETSC: PORTABLE EXTENSIBLE TOOLKIT FOR SCIENTIFIC COMPUTATION

The Portable Extensible Toolkit for Scientific Computation (PETSc)[13] is a high performance computing library developed in the Mathematics and Computer Science Division at Argonne National Laboratory. It includes a suite of data structures and routines for the scalable (parallel) solution of large-scale scientific applications.

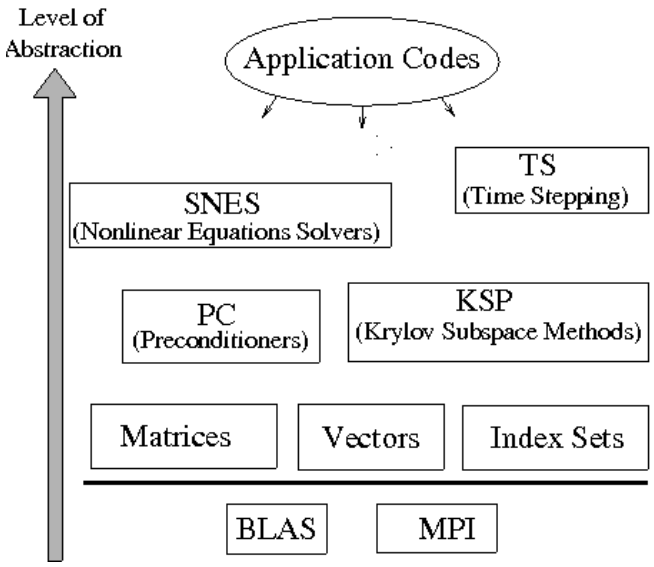

Figure 4: Organization of the PETSc library [13]

PETSc utilizes the Message Passing Interface (MPI) standard for all message passing communication. It is portable to various operating systems such as Unix, Linux, Windows,

Macintosh etc., and the application code can be written either in C, C++, Fortran, or Python. Various linear solution schemes, direct and iterative methods with a variety of precondtioners, are available for optimizing the linear solution process. Additionally, various third-party packages, such as ParMetis, SuperLU, UMPACK etc., are can be downloaded with PETSc installation. PETSc consists of a set of libraries that contain routines for creating vectors, matrices, and distributed arrays, both sequential and parallel, as well as libraries of linear and nonlinear numerical solvers. PETSc also incorporates time-stepping methods and graphics.

The wide range of sequential and parallel linear solvers, preconditioners, reordering strategies, flexible run-time options, ease of code implementation, debugging options, and a comprehensive source code profiler make PETSc an attractive experimentation platform for developing the three-phase transient stability simulator. The proposed parallel three-phase transient stability simulator has been developed using the recently release version, version 3.2, of PETSc. The organization of the PETSc library is illustrated in Figure 4.

7. CONCLUSIONS

This paper discussed the development of the parallel three-phase transient stability simulator and presented the details and the results of its parallel implementation. The scalability results of the proposed simulator were presented using GMRES with two variants of the Block-Jacobi preconditioner and the very dishonest preconditioning strategy. The parallel performance shows good scalability with both the preconditioners for the largest test case. The slow down of the sequential performance, for large systems, due to slow fetching of data from the main memory was identified.

To make the proposed parallel three-phase transient stability simulator a practical application a lot of development still needs to be done. In the future we propose to further develop the proposed simulator by testing its accuracy, for both unbalanced and balanced cases, with commercially accepted packages, incorporating different equipment models, and exploring strategies to speed up the numerical solution for both serial and parallel implementation.

8. REFERENCES

[1] Chai., J. S., N. Zhu, A. Bose, and D.J. Tylavsky. "Parallel Newton type methods for power system stability analysis using local and shared memory multiprocessors", *IEEE Transactions on Power Systems*, 6.4 (November 1991): 9-15.

[2] Decker, I. C., D. M. Falcao, and E. Kaszkurewicz. " Parallel implementation of a power system dynamic simulation methodology using the conjugate gradient method", *IEEE Transactions on Power Systems*. 7.1 (February 1992): 458-465.

[3] Decker, I. C., D. M. Falcao, and E. Kaszkurewicz. "Conjugate gradient methods for power system dynamic simulation on parallel computers", *IEEE Transactions on Power Systems*, 9.2 (May 1994): 629 - 636.

[4] Alvarado, F. L., "Parallel solution of transient problems by trapezoidal integration", *IEEE Transactions on Power Apparatus and Systems*, PAS-98 (May/June 1979): 1080-1090.

[5] Ilic., M., M. L. Crow, and M. A. Pai., "Transient stability simulation by waveform relaxation methods", *IEEE Transactions on Power Systems*, 2.4 (November 1987): 943-952.

[6] Crow., M. L., and M. Ilic., "The parallel implementation of waveform relaxation methods for transient stability simulations", *IEEE Transactions on Power Systems*, 5.3 (August 1990): 922-932.

[7] Hou L., and A. Bose., "Implementation of the waveform relaxation algorithm on a shared memory computer for the transient stability problem", *IEEE Transactions on Power Systems*, 5.3 (August 1991).

[8] Falcao., D., "High performance computing in power system applications", *Invited Lecture at the 2nd International Meeting on Vector and Parallel Processing (VECPAR '96)*, Porto, Portugal, (September 1996).

[9] Jalili-Marandi V., and V. Dinavahi., "SIMD-based large scale transient stability simulation on the graphics processing unit", *IEEE Transactions on Power Systems*, 20.3 (August 2010): 1589-1599.

[10] Kulkarni, A. Y., M. A. Pai, and P. W. Sauer., "Iterative solver techniques in fast dynamic calculations of power systems", *International Journal of Electrical Power and Energy Systems*, 23.2 (March 2001), 237-244.

[11] Makram., E. B., V. Zambrano, R. Harley, and J. Balda., "Three-phase modeling for transient stability of large scale unbalanced distribution systems", *IEEE Transactions on Power Systems*, 4.2 (May 1989): 487-493.

[12] Abhyankar, S. G. *An implicitly coupled electromechanical and electromagnetic transients simulator for power systems*, Ph.D. Dissertation, Illinois Institute of Technology, Chicago. (2011).

[13] Balay, S., J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, and B. F. Smith, and H. Zhang, *PETSc users manual*. ANL-95/11-3.1 (2010).

[14] Kundur, P., *Power System Stability and Control*, New York: McGraw-Hill,1994.

[15] Sauer, P.W., and M.A.Pai., *Power System Dynamics and Stability*, New Jersey: Prentice Hall Inc., 1998.

[16] Balay, S., J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, and B. F. Smith, and H. Zhang, *PETSc users manual*. ANL-95/11-3.2 (May 2011).

[17] Balay, S., J. Brown, K. Buschelman, V. Eijkhout, W. Gropp, D. Kaushik, M. Knepley, L. McInnes, and B. F. Smith, and H. Zhang, *PETSc Web page*. (2011) http://www.mcs.anl.gov/petsc.

[18] Karypis, G., K. Schloegel, and V. Kumar., *PARMETIS Parallel Graph Partitioning and Sparse Matrix Ordering Library*. Department of Computer Science and Engineering, University of Minnesota, (August 2003).

[19] Siemens PTI Inc. *PSS/E Operation Program Manual: Volume 2* ver 30.2.

[20] Siemens PTI Inc. *PSS/E Application Guide, Volume 2* ver 30.2.