Developing a Dynamic Model of Cascading Failure for High Performance Computing using Trilinos

Christopher Parmer
Dept of Electrical and Computer Engineering
McGill University
Montreal, QB, H3A 2A7
christopher.parmer@gmail.com

Eduardo Cotilla-Sanchez
School of Engineering
The University of Vermont
Burlington, VT 05405
eduardo.cotilla-sanchez@uvm.edu

Heidi K. Thornquist
Sandia National Laboratories
Albuquerque, NM 87185
hkthorn@sandia.gov

Paul D. H. Hines
School of Engineering
The University of Vermont
Burlington, VT 05405
paul.hines@uvm.edu

ABSTRACT

This paper describes work-in-progress toward the development of a dynamic model of cascading failure in power systems that is suitable for High Performance Computing simulation environments. Doing so involves simulating a power grid as a set of differential, algebraic and discrete equations. We describe the general form of the algorithm in use for this simulation and provide details about the implementation using the Trilinos software libraries. Several computational tests illustrate how the proposed approach can be leveraged to optimize the computational efficiency of cascading failure simulation.

Categories and Subject Descriptors

J.2 [**Computer Applications**]: Physical Sciences and Engineering—*Engineering*

General Terms

Algorithms, Performance

Keywords

Power Systems, Cascading Failures, High Performance Computing, Trilinos

1. INTRODUCTION

The $n-1$ security criterion is an effective heuristic for reliable power power grid operations so long as multiple contingencies do not occur simultaneously [18]. If multiple elements fail simultaneously (an $n-k$ contingency), NERC reliability standards [17] allow utilities to implement emergency control policies, such as load shedding, to reduce the risk of cascading outages that could result. However some control policies that would appear to reduce risk can have unexpected long-term outcomes [9]. Verifying that a given control policy is effective at reducing blackout risk over all possible $n-k$ contingencies requires a prohibitively large number of simulations. Exhaustively calculating the reaction of a system with $n = 10^4$ components to only $n-2$ and $n-3$ contingencies requires on the order of 10^{11} simulations. Furthermore, accurately calculating the effect of $n-k$ contingencies requires one to consider not only the immediate overcurrent and undervoltage conditions that result from the outages, but the potential cascading sequences that can occur after the initial overloads. This substantially increases the computational burden associated with the analysis of $n-k$ events.

There are a number of methods available that can reduce the computational requirements for cascading failure analysis [24]. For example, one can reduce the number of contingencies that need to be considered through careful contingency screening [8, 3]. However, for the $n = 10^4$, $k = 3$ case, reducing the number of simulations by, for example, 99.9%, still leaves 10^8 potential cascading outage sequences to run. Another approach is to reduce the computer time needed for individual simulations by using simplifying assumptions, such as developing models based on the dc power flow equations. However, doing so requires that some mechanisms of cascading failure, such as voltage collapse and dynamic instability, must be ignored. While the use of simplified models is often appropriate, little to no existing work has systematically compared the statistical differences between dynamic and sequential steady state models of cascading failure. Finally, high performance computing (HPC) can be harnessed by running many simulations in parallel. As with contingency screening, this approach also has limitations,

given the enormous size of the search space.

The long-term goal of this project is to use a combination of these approaches by reducing the model order to decrease simulation time, optimizing the numerical methods, carefully screening the number of simulations required using statistical methods (e.g., [7]) to gain insight from a limited number of simulations, and finally harnessing HPC to perform simulations in parallel. It is our conjecture that using dynamic models for cascading failure analysis, and comparing the results from dynamic models with results from steady-state contingency analysis methods will produce additional insight, over that which would come from steady-state methods alone. Future work will explore methods to combine these approaches. However, the goal of this paper is to present incremental results toward the development of a fully dynamic cascading failure model, which is optimized for use in a HPC environment.

The increased accessibility of HPC systems has led to a recent increase in effort to develop methods to improve the computational efficiency of time-domain power system simulators. One of the most significants efforts in this regard comes from the PEGASE Consortium in Europe, which has published a comprehensive report about the simulation of large power systems and the requirements with respect to algorithms and computing capabilities [2]. They propose a method that can simulate events in a pan-European dynamic grid model at 1/3 of real time. Among their results, they report that the linear solvers KLU [21] and Pardiso [20] are particularly efficient at solving power system matrix equations. Relatedly, Khaitan and McCalley [15, 14] report that multi-frontal pre-conditioners can be effectively applied to the solution of the differential algebraic equations (DAEs) that form the core of numerical power system simulation. Several US National Laboratories have leveraged experience with HPC to develop efficient algorithms for grid simulation. Chen et al. [1] and Jin et al. [11] improve the speed of contingency selection by developing dynamic load balancing schemes over parallel systems. Outside of the power systems field, the Xyce simulation code [12] exploits parallelism to efficiently and accurately solve the DAEs associated with large-scale integrated circuits. Much of the technology in Xyce is being harnessed in this project through the Trilinos software libraries [10].

This paper describes the software implementation (in-progress) of a dynamic cascading failure simulator using software packages from Trilinos [10]. Trilinos is a suite of packages for solving large-scale numerical problems within HPC environments. The software suite provides both serial and parallel computing capabilities. Our simulator makes direct use of Trilinos' interfaces to linear system solvers, the nonlinear algebraic system solver (NOX), and the DAE solver Rythmos, as well as a number of the other linear algebra and utility packages. Developing this simulation with Trilinos provides the project with access to a wide variety of numerical algorithms, facilitating the ability to perform numerous computational experiments.

2. MODELING CASCADING FAILURE AS A SYSTEM OF DIFFERENTIAL, ALGEBRAIC, AND DISCRETE EQUATIONS

It is well known that the power grid can be represented as a system of differential ($\mathbf{f}$) and algebraic ($\mathbf{g}$) equations [16]. When the important power system dynamics include discrete changes, such as is the case during cascading failure, a set of discrete equations ($\mathbf{h}$) should be added to this system, such that:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{f}(t, \mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t)) \quad (1)$$

$$\mathbf{0} = \mathbf{g}(t, \mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t)) \quad (2)$$

$$\mathbf{0} > \mathbf{h}(t, \mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t)) \quad (3)$$

where $\mathbf{x}(t)$ is a vector of continuous electro-mechanical state variables that change with time according to the differential equations, $\mathbf{y}(t)$ is a vector of continuous state variables that have purely algebraic relationships to other variables in the system, and $\mathbf{z}(t)$ is a vector of state variables that can take only binary or integer states ($z_i \in [0, 1, 2, ...]$). In our formulation each discrete variable $z_i(t)$ is governed by an equation (3) that indicates a discrete state change at the instant in time that $h_i(t)$ crosses the origin. When these discrete changes occur the electro-mechanical variables $\mathbf{x}(t)$ do not change across the small discrete time interval:

$$\lim_{\epsilon \to 0} \mathbf{x}(t + \epsilon) = \mathbf{x}(t)$$

whereas the algebraic variables are allowed to change instantaneously.

Assuming that we desire to simulate the impact of k branch (transmission line or transformer) outages occurring at time points $t_1 \leq t_2 \leq ... \leq t_k$ over the time interval $[t_0, t_e]$ such that $t_0 < t_1$ and $t_k < t_e$, the algorithm for simulating the potential cascading outage is the following:

1. Set the exogenous event number $r = 0$ and the current time: $t = t_0$.
2. If a discrete event occurs at time t:
   (a) Implement the event by changing the discrete vector from $\mathbf{z}(t)$ to $\mathbf{z}(t + \epsilon)$
   (b) Solve for the change in the algebraic variables across the infinitesimal time interval $t \to t + \epsilon$ by solving for $\mathbf{y}(t + \epsilon)$ in the non-linear system (2):
   $$\mathbf{0} = \mathbf{g}(t, \mathbf{x}(t), \mathbf{y}(t + \epsilon), \mathbf{z}(t + \epsilon))$$
   (c) Set $t = t + \epsilon$.
3. Simulate the DAE system ($\mathbf{fg}$) from t until either a discrete threshold is reached ($h_i = 0$) for any i, or $t = t_{r+1}$.
   (a) Return the simulation end time, $t \leq t_{r+1}$, the differential variable trajectory $\mathbf{x}([t_r, t])$, and the algebraic variable trajectory $\mathbf{y}([t_r, t])$.
   (b) If $t < t_{r+1}$ record the element (relay) affected by the endogenous discrete change.
   (c) If $t = t_{r+1}$ increment $r = r + 1$.

4. If $t < t_e$ repeat from 2.

Note that similar approaches are used in mid-term stability simulation [16] studies. Here we describe the algorithm to place careful attention on the process of handling discrete events. The general problem of solving Ordinary Differential Equations with discontinuous equations is a notably challenging one. The method proposed here essentially assumes that there exists a real-valued solution to the algebraic equations after change in the discrete variables (2). Clearly there are problems in which this is not the case. See [22] for a detailed discussion of degenerate cases for discontinuous ODEs. Additional research is needed to improve algorithms of this sort to robustly handle discontinuities in power system dynamic studies.

2.1 Power system model

To adapt this general formulation to the specific power system case, we model a power network as a system of transmission lines, transformers, simulated loads, and dynamic rotating generators, using the traditional ac network equations and a somewhat reduced order rotating machine model. The following subsections provide some detail about our formulation.

2.1.1 Algebraic equations

The algebraic equations in our model use the standard pair of ac active and reactive power flow equations, as is common in most power system models. However, note that the equations employ impedances that are based on the 60Hz frequency response of the network. Yet, during extreme events, the fundamental frequency commonly deviates from 60Hz. Additional research is needed to determine whether one would obtain more accurate results from frequency dependent impedance models. Given the constant impedance assumption, the active nodal power injection balance is:

$$\begin{aligned} &P_{g,i}(\delta_{m,i}, E'_{a,i}, |V_i|, t) - P_{d,i}(|V_i(t)|, t) \\ &= \sum_{k=1}^{n} |V_i||V_k|(g_{ik}\cos\theta_{ik} + b_{ik}\sin\theta_{ik}) \end{aligned} \tag{4}$$

where $P_{g,i}$ is the real power output of the generator at bus i and $P_{g,i}(t) = 0$ for non generator buses. $P_{g,i}$ for generator buses is given by:

$$\begin{aligned} P_{g,i}(t) = &\frac{E'_{a,i}|V_i|}{X'_{d,i}}\sin(\delta_{m,i})... \\ &+\frac{|V_i|^2}{2}\left(\frac{1}{X_{q,i}} + \frac{1}{X'_{d,i}}\right)\sin(2\delta_m)\forall i \in G \end{aligned} \tag{5}$$

and $P_{d,i}$ represents the load at bus i. Using standard notation, $\delta_{m,i}$ is the generator machine rotor angle relative to the terminal bus voltage phase angle θ_i, E'_a is the sub-transient machine voltage, $|V_i|$ is the per unit terminal bus voltage, and g_{ik} and b_{ik} are the corresponding real and imaginary elements of the system admittance matrix ($\mathbf{Y}_{BUS}$). $|V_i|$ and θ_i are algebraic variables in $\mathbf{y}$, whereas δ_m and E'_a are differential variables in $\mathbf{x}$.

In our model we extend the well-known ZIP (impedance, current, power) representation of electrical loads with a time invariant exponential term (resulting on a "ZIPE" load model). The exponential portion of the load has the following power-voltage relationship:

$$S_{d,i}(|V_i(t)|, t) = S_{d,i}(t)|V_i|^{\gamma}.$$

Using the exponential model, rather than constant power, ensures that power consumption approaches zero as $|V_i|$ approaches zero, which is important for the simulation of extreme events, such as cascading failure.

Similarly, (6) describes the reactive power balance for bus i.

$$\begin{aligned} &Q_{g,i}(\delta_{m,i}, E'_{a,i}, |V_i|, t) - Q_{d,i}(|V_i|, t) \\ &= \sum_{k=1}^{n} |V_i||V_k|(g_{ik}\sin\theta_{ik} - b_{ik}\cos\theta_{ik}) \end{aligned} \tag{6}$$

where Q_g comes from:

$$Q_{g,i}(t) = \begin{cases} \frac{E'_{a,i}|V_i|}{X'_{d,i}}\cos(\delta_{m,i})... & \\ -|V_i|^2\left(\frac{\cos(\delta_{m,i})^2}{X'_{d,i}} + \frac{\sin(\delta_{m,i})^2}{X_{q,i}}\right) & \forall i \in G \\ 0 & \forall i \notin G \end{cases}$$

2.1.2 Differential equations

In this formulation we use differential equations ($\mathbf{f}$) only to represent generator dynamics. Many have shown that loads also have dynamical behaviors, but in the current instantiation we neglect load dynamics. Our generator model consists of the standard second order swing equation, coupled with a second order salient pole generator model, second order droop-control governor model and a second order exciter/voltage control model. The exact details of this generator model are currently being finalized. With two variables in each of the generator, governor and exciter, the DAE index is 2.

2.2 DAE solution methods

A wide variety of methods exist for solving DAE systems, each with advantages and disadvantages in terms of computational complexity, numerical stability, and numerical accuracy [23]. Most modern solution methods divide the solution process into the the four numerical processes described below, each of which is dependent on the lower level process.

The highest level process is *Discrete time integration,* in which a time step size h is chosen, and $\mathbf{x}(t+h)$ and $\mathbf{y}(t+h)$ are calculated based on $\mathbf{x}(t)$, $\mathbf{y}(t)$, and $\mathbf{z}(t)$. This is handled in Trilinos by the Rythmos package, and requires the solution of non-linear systems of equations.

The *Non-linear system solution* step involves solving set of implicit non-linear equations to find $\mathbf{x}(t+h)$ and $\mathbf{y}(t+h)$ that are consistent with the DAE system and the state at time t. The form of these equations depends on the integration method (implicit vs. explicit, order, etc.). Typically the non-linear system is solved using a variant of Newton's method, which involves solving a series of linear systems, $\mathbf{Ax} = \mathbf{b}$[1] for $\mathbf{x}$ or $\mathbf{J}(\mathbf{x}_{n-1})(\mathbf{x} + \mathbf{x}_{n-1}) = -F(\mathbf{x}_n - \mathbf{x}_{n-1})$ for $\mathbf{x} + \mathbf{x}_{n-1}$ where $\mathbf{J}$ is the system's Jacobian matrix evaluated with $\mathbf{x}_{n-1}$, F is a vector of equation residuals evaluated

[1]Note that here $\mathbf{x}$ refers to a generic vector, whereas $\mathbf{x}$ refers to the differential variables above.

with $\mathbf{x}_{n-1}$. Section 4 describes results comparing different methods to solve the non-linear systems.

Linear system solution. The series of systems, $\mathbf{Ax} = \mathbf{b}$, generated through the linearization of the non-linear system are sparse, so both iterative and direct solution methods can be used to find the search direction $\mathbf{x}$. Direct methods require a matrix factorization, which generally does not scale well for large systems and suffers from limited parallelism. For very large linear systems, with more than 10^5 unknowns, it is generally preferable to use iterative solution methods, which have improved parallel scalability, provided an efficient preconditioner is available.

In general, an efficient preconditioner solves a linear problem $\hat{\mathbf{A}}\hat{\mathbf{x}} = \hat{\mathbf{b}}$, where $\hat{\mathbf{A}}$ is an inexpensive approximation of $\mathbf{A}$ that accelerates the convergence of the iterative method to compute $\mathbf{x}$. For power system simulation, one proposed approach to preconditioning is to occasionally factor the system Jacobian $\mathbf{A}$ with a sparse direct method and use these exact factors to compute $\hat{\mathbf{x}}$. This can be used to precondition $\mathbf{A}$ for some specified number of non-linear iterations or until the solution error ($||\mathbf{A}\hat{x} - \mathbf{b}||$) exceeds some threshold before recalculating the factorization.

For smaller linear systems, sparse direct methods are more often utilized, and require the factorization of the matrix A into components that can be applied through triangular solves. For non-symmetric matrices, a common numeric factorization is the LU factorization, that decomposes a matrix $\mathbf{A}$ into a lower triangular matrix, $\mathbf{L}$, and an upper triangular matrix, $\mathbf{U}$, such that $\mathbf{A} = \mathbf{LU}$. If $\mathbf{L}$ and $\mathbf{U}$ are sufficiently sparse, it is computationally inexpensive to solve $\mathbf{Ly} = \mathbf{b}$ and then $\mathbf{Ux} = \mathbf{y}$ for $\mathbf{x}$ with Gaussian elimination to solve $\mathbf{Ax} = \mathbf{b}$. To maximize the sparsity in $\mathbf{L}$ and $\mathbf{U}$, symbolic reordering algorithms can be applied to $\mathbf{A}$. Reducing fill-in in these factors is necessary as the computational cost of factoring $\mathbf{A}$ and the resulting triangular solves are related to the number of non-zeros in $\mathbf{L}$ and $\mathbf{U}$. Thus, a good symbolic reordering of $\mathbf{A}$ can speed up the numeric factorization if it allows for a smaller memory allocation for non-zero entries and less floating point operations (FLOPs) to compute $\mathbf{L}$ and $\mathbf{U}$. The symbolic reordering only needs to be done once, if the nonzero pattern (graph) of $\mathbf{A}$ is static, and reused in all subsequent numerical factorizations. For the cascading failure simulation case, the nonzero pattern will remain constant so long as the discrete state of the system ($\mathbf{z}$) remains unchanged (e.g., branches are not switched).

2.3 Methods to parallelize the DAE solution process

While this paper does not directly describe results from parallel simulation, there are a number of places in which parallel computer architectures might be employed to improve DAE solution speeds. The obvious starting point is to parallelize the linear system solution process. As discussed in [14], existing codes for iterative linear system solution can exploit parallel architectures. However, producing the pre-conditioner, generally requires a direct linear solver (like UMFPACK or KLU). Few codes exist for direct linear system solution that employ parallelism. And, our tests indicate that one code that is parallel (SuperLU) does not perform well for power system matrices. An effort to produce direct linear solvers that both work well for power systems matrices and exploit parallel architectures, would be notably valuable. It is important to note, however, that power system matrices (likely to be on the order of 10^5 variables) tend to be relatively small, when compared to those that result from many DAE problems, for which parallel methods have be successful. Given the relatively small size of the matrices, the classic tradeoff between interprocess communication and increased parallelism becomes particularly important.

Solving other components of an individual simulation in parallel is very difficult, since the solution process is trajectory dependent.

3. COMPARING NUMERICAL METHODS USED IN SOLVING POWER SYSTEM DIFFERENTIAL ALGEBRAIC EQUATIONS

As explained above, numerically integrating a system of differential-algebraic equations involves a hierarchical stack of numerical methods. The discrete time integration involves non-linear system solution methods, which subsequently require linear system solution methods, which further often involve matrix factorization. This section describes results from using Trilinos packages to interface the power system equations with a DAE integrator (the Rythmos Package via the Piro interface). We used the Rythmos/Piro tools to solve aspects of the DAE solution problem using a variety of numerical method packages and options, with the intention of optimizing the accuracy and speed performance of large power system simulations in the future. This section provides a number of results that illustrate how Trilinos, and packages like it, can be to compare different numerical solution methods to a variety of power system simulation problems.

3.1 Comparing direct linear solvers for efficient solution of power systems equations

Firstly we evaluated the relative merits different approaches to factorizing power system matrices. We do so using a power flow Jacobian matrix, which derives from the system algebraic equations (2). The power flow Jacobian differs somewhat from the Jacobian of the non-linear system that would be solved during DAE integration, but includes the most challenging portions of that matrix: the power flow constraints (4) and (6). Our power flow matrix is formulated from a model of the Polish power grid, which is publicly available with MATPOWER [25]. Figure 1 shows the locations of the nonzero elements (the sparsity structure) for this test matrix. This Jacobian is unsymmetric, extremely sparse (0.14% non-zero entries), and moderately sized with 4439 variables. For comparison purposes, power flow models of the US Eastern Interconnect contain on the order of 10^5 variables.

We factorized this test matrix using algorithms from four software packages, each of which was interfaced through Trilinos's Amesos package [19]. The four numerical packages that we tested were LAPACK, a library providing routines for solving linear systems with dense matrices; SuperLU, a library for the direct solution of large, sparse, non-symmetric

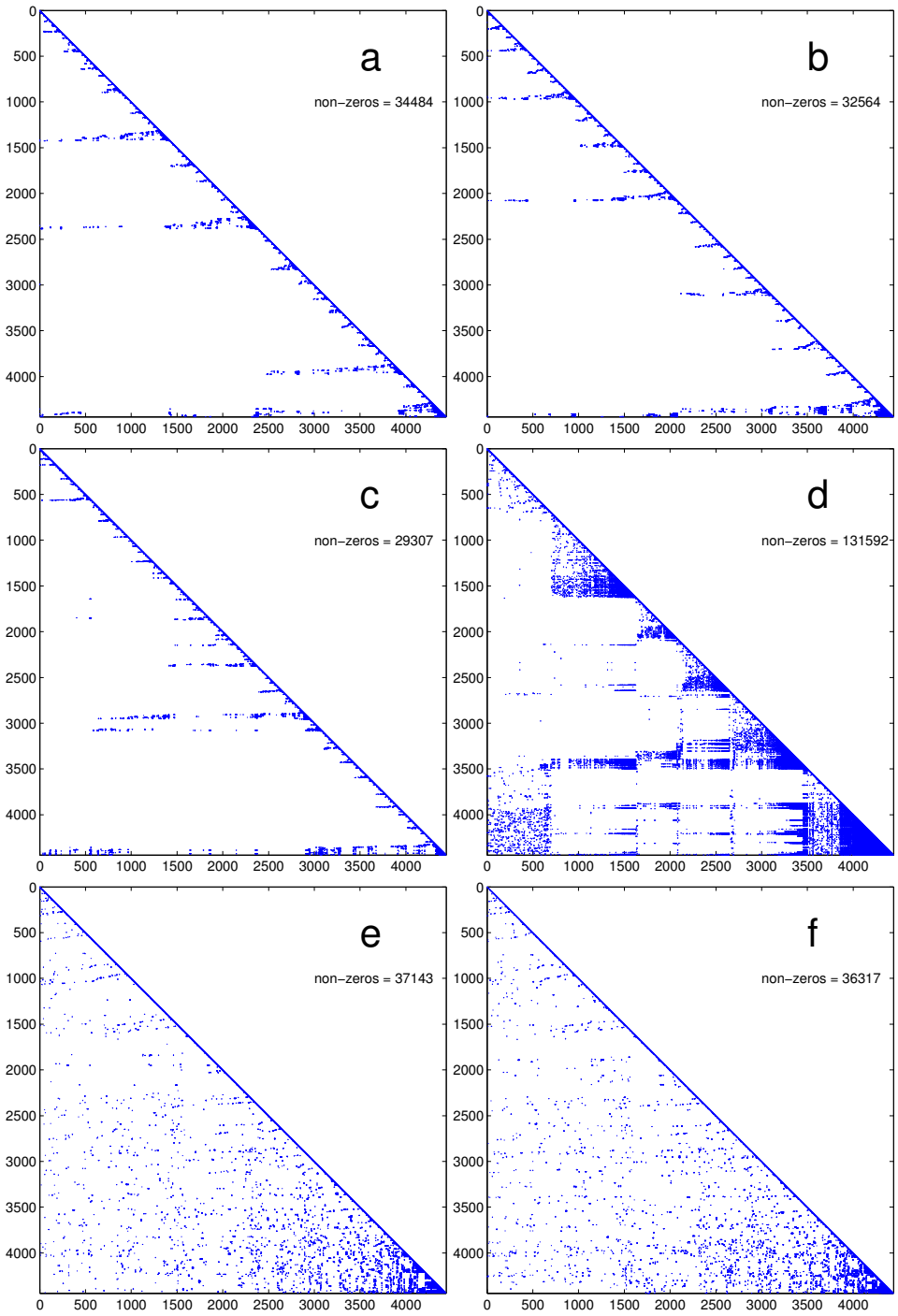

Figure 2: The lower triangular matrices (L) that result from UMFPACK's and KLU's symbolic reordering and numeric factorization. The symbolic reordering of the Polish power flow Jacobian (Fig. 1) for each method rearranges the Jacobian into a graph that is more efficient for numeric factorization. UMFPACK's decomposition after COLAMD (panel a) and METIS (panel b) ordering allocates fewer non-zero entries than three of KLU's methods. Although UMFPACK allocates a similar number of non-zeros as KLU's BTF AMD decomposition, we find UMFPACK be less efficient largely due to the fact that UMFPACK treats portions of the numeric factorization problem using dense matrix methods, and power system matrices are very sparse, even along the diagonal after reordering. Of the 4 methods available with KLU: AMD with (panel c) or without BTF (panel d) and COLAMD with (panel e) or without (panel f) BTF, we find that AMD ordering with BTF factorization allocates the least number of non-zeros and performs the fastest. The block triangular form does not contain true BTF blocks and therefore does not have the properties that would substantially simplify a linear solve, although the AMD ordering indirectly benefits from BTF factorization by allocating a sixth of the non-zero entries than without it.

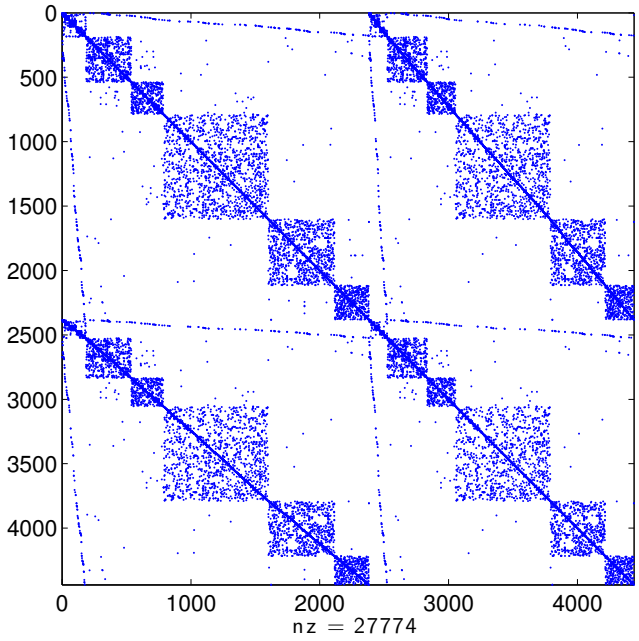

Figure 1: Sparsity Structure of the Polish power flow Jacobian matrix. The colored dots indicate the locations non-zero entries in the matrix. White spaces are zeros. This Jacobian is unsymmetric, extremely sparse, and moderately sized.

linear systems [6]; UMFPACK, a package for solving unsymmetric sparse linear systems with the Unsymmetric Multi-Frontal method [5]; and KLU, a sparse LU factorization algorithm designed for use in circuit simulation [4]. Table 1 compares solution times for each of these solvers. It is important to note that the symbolic ordering process typically would occur only once between discrete changes in a cascading failure model, whereas the numeric factorization would need to occur numerous times, making this stage particularly critical to the simulation. Our results indicate that KLU performs numeric factorization dramatically (at least four times) faster than all other packages. LAPACK performs poorly because it does not exploit the sparsity of the power system matrices.

We further analyzed the results from KLU and UMFPACK. KLU and UMFPACK together contain two symbolic reordering algorithms, Approximate Minimum Degree Ordering (AMD) and Column Approximate Minimum Degree Ordering (COLAMD), and an interface to a third-party symbolic reordering package, METIS (see Fig. 2 for illustrations of the sparsity patterns of the Jacobian after factorization). KLU contains algorithms that attempt to factorize the matrix into Block Triangular Form (BTF) which can be used to further reduce the computational complexity in a linear solve by allowing only a portion of permuted sub-matrices to be numerically factorized. However, the Polish power flow Jacobian matrix's block triangular form does not show evidence of significant block triangular structure, which means that this is not the source of KLU's relative efficiency. However, we do observe that the AMD ordering used by KLU and UMFPACK indirectly benefit from BTF factorization by allocating only a sixth of the non-zeros in the factorized matrix, relative to the case without AMD ordering (see Table 2). Although UMFPACK allocates a similar number of non-zeros as KLU in factorized matrices, after BTF and AMD ordering, it performs 3 times as many floating point operations (FLOPs) as KLU. This difference appears to be the result of the fact that UMFPACK uses dense routines from level-3 BLAS to process the frontal sub-matrices, whereas KLU handles the frontal sub-matrices using sparse codes. From Fig. 2, we note that UMFPACK's factorization does not contain dense sub-matrices, which indicates that significant amounts of CPU time and memory are being unnecessarily expended on zero-valued entries.

3.2 Comparing non-linear solvers for efficient solution of power systems equations

There are two cases in our simulator where it is necessary to solve non-linear algebraic equations. The first is to gain consistent algebraic conditions for a differential-algebraic integration by solving for $\mathbf{y}(t)$ (or $\mathbf{y}(t+\epsilon)$) in Eq. 2, given $\mathbf{x}(t)$ and $\mathbf{z}(t)$. This occurs most notably after discrete changes in the system. The second case is during differential-algebraic integration, when solving a system of equations to find $\mathbf{x}(t+h)$ and $\mathbf{y}(t+h)$ given the variables at time t. In this section we describe tests that compare the computational time required to find $\mathbf{y}(t+\epsilon)$ after a discrete change in the IEEE 39 bus system.

There are many algorithms for iteratively solving non-linear systems, however the general algorithm involves four steps to iteratively find some $\mathbf{x}$ that solves $\mathbf{g}(\mathbf{x}) = \mathbf{0}$. The first is to evaluate the error of the current solution estimate $\hat{\mathbf{x}}$: $\hat{g} = ||\mathbf{g}(\hat{\mathbf{x}})||$ and to stop if $\hat{g}$ is sufficiently small. The second is to choose a search direction $\mathbf{p}$. The familiar Newton-Raphson algorithm choses the search direction by linearizing $\mathbf{g}$ around the current estimate: $\mathbf{p} = -\left[\frac{d\mathbf{g}}{d\mathbf{x}}\right]^{-1}\mathbf{g}(\hat{\mathbf{x}})$. The third step is to choose a step size α such that $||\mathbf{g}(\hat{\mathbf{x}}+\alpha\mathbf{p})||$ is closer to zero than $||\mathbf{g}(\hat{\mathbf{x}})||$. Finally, once a sufficiently good step size is chosen the current solution estimate is updated: $\hat{\mathbf{x}} = \hat{\mathbf{x}} + \alpha\mathbf{p}$.

Different solution methods approach each of these stages differently, and have various advantages and disadvantages. For example, line search methods work to find a step size that minimizes $||\mathbf{g}(\hat{\mathbf{x}} + \alpha\mathbf{p})||$ at each iteration. Trust region methods limit the step size to be within a region that is known to be approximately linear. A detailed description of non-linear solution approaches is available in, for example, [13]. Through our interface with Trilinos' non-linear solver package, NOX, we test 3 types of nonlinear solvers (Line Search Based, Trust Region Based, and Inexact Trust Region Based) with 4 types of directional options (Newton, Steepest Descent, Nonlinear Conjugate Gradient, Broyden) and 4 types of line-search options (Full Step, Backtrack, Polynomial, More-Thuente).

Table 3.2 gives the average time for each solver option tested. The average time for many of the algebraic solves hovers around the timing resolution (0.01 seconds) of the machine running the tests. So, 300 consecutive algebraic equations were solved, timed, and then averaged. In future work we plan to run these tests over a wide variety of system con-

Table 1: Power system matrix factorization times (seconds) for several linear solver packages

	LAPACK	SuperLU	UMFPACK	KLU
Symbolic Reordering Time	NA	NA	0.005	0.0030
Numeric Factorial Time	6.2180	0.0154	0.0192	0.0037
Solve Time	0.0488	0.0009	0.0030	0.0003
Total Time	6.2668	0.0163	0.0272	0.007
Residual Norm	8.12e-12	1.05e-11	7.59e-12	9.64e-12

Table 2: UMFPACK and KLU Ordering Statistics.

Direct Linear Solver Package	KLU				UMFPACK	
Symbolic Ordering Method	AMD		COLAMD		COLAMD	METIS
Block Triangular Factorization	Yes	No	Yes	No	NA	NA
Symbolic and Numeric MFlops (CPU time)	8.72	56.15	1.69	1.64	28.61	27.75
Number of Non-zeros in L+U (in thousands)	58.73	388.73	82.55	80.92	62.15	64.56
Peak Memory Usage (MBytes)	1.73	15.07	7.30	7.30	NA	NA

ditions, with a much larger system. Despite the small size of our system, the results illustrate the utility of using a generic numerical solver package in understanding the relative merits of different numerical methods.

3.3 Integrator interface for efficient solution of power system equations

The next stage in our testing involves testing various methods for combining the results in Section 3.1 and 3.2 to solve DAE systems. Currently, we have interfaced our power system model with Trilinos's Rythmos integrator package through Trilinos's Piro interface. There are two integrators available through Rythmos: a simple fixed step backward Euler method and a backward differentiation formula method of orders 1 through 5 with variable-step size. However, only the simple backward Euler method is currently available through the Piro interface. Using this integrator we have completed a few preliminary tests of the DAE integrator, by simulating the response of the IEEE 39 bus system to a step decrease in load at bus 17. Fig. 3 illustrates these preliminary results.

Clearly, there are drawbacks to using backward Euler as an integration method. Backward Euler can, under some circumstances, suffer from numerical instability. Future work will investigate alternative integration approaches.

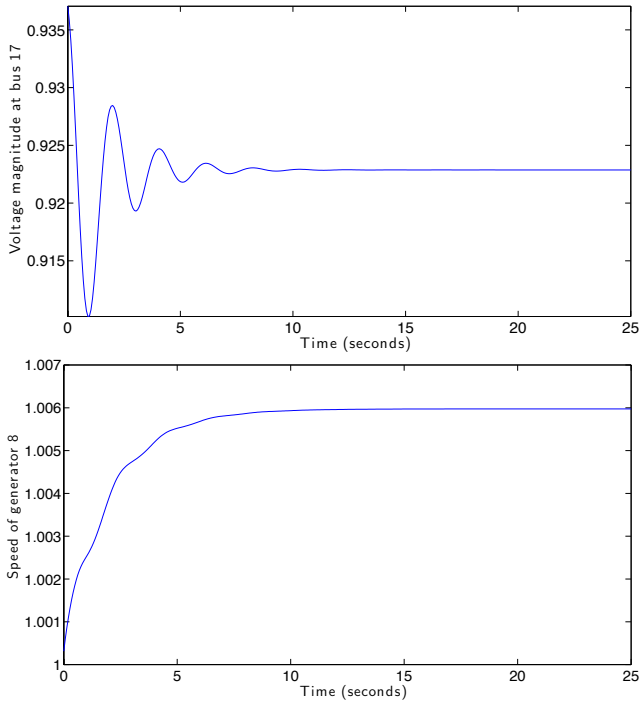

Figure 3: Illustrative DAE integration results for a small power system through our Trilinos Rythmos interface. This figure shows the bus voltage at bus 17, and the per unit machine speed at generator 8, integrated for 25 seconds after a step decrease in the (purely resistive) load at bus 17.

4. CONCLUSIONS AND FUTURE WORK

In this paper we describe the development of a dynamic power system simulator that has the ability to tune the underlying direct linear solver, non-linear solver, and the DAE integrator. Using a general purpose suite of numerical solvers (Trilinos) allows one to perform experiments involving the various methods that underly DAE solution. In the preliminary results described in this paper, we have found that the KLU linear solver package with AMD ordering and block triangular form dramatically outperforms UMFPACK, LAPACK, and SuperLU. We suggest that the reason for this increased efficiency is that power systems matrices remain highly sparse, without dense blocks on the diagonal, even after careful symbolic reordering. Because KLU uses sparse numerical routines exclusively, it appears to perform better for the solution of power grid linear systems. Secondly, we

Table 3: Nonlinear Solve Results

	Newton				Steepest Descent			
	Full Step	Backtrack	Polynomial	More-Thuente	Full Step	Backtrack	Polynomial	More-Thuente
Line Search	**0.0116**	0.0133	0.0131	-	UC	0.0155	0.0139	-
Trust Region	**0.0129**	**0.0129**	**0.0126**	**0.0129**	**0.0130**	0.0131	**0.0130**	**0.0128**
Inexact Trust Reg.	0.0140	0.0133	0.0135	0.0136	0.0137	0.0139	0.0134	0.0137
	Nonlinear Conjugate Gradient				Broyden			
	Full Step	Backtrack	Polynomial	More-Thuente	Full Step	Backtrack	Polynomial	More-Thuente
Line Search	UC	UC	UC	-	**0.0126**	**0.0127**	0.0145	-
Trust Region	0.0140	0.0135	0.0136	0.0140	-	-	-	-
Inexact Trust Reg.	0.0138	0.0139	0.0139	0.0135	0.0134	0.0136	0.0134	0.0140

All results are in seconds. UC = unconverged.

found that the Line Search nonlinear solver with the Newton direction method and the Full Step line-search method is the fastest non-linear solver for a small power system, though these results were for a single permutation of a small case, and therefore not particularly conclusive. Finally, we interface the power systems simulation problem to a simple backward Euler integrator and illustrate that the solver is stable.

Note that all of these tests were completed within a serial computing environment. Therefore, the testing of the parallel/HPC features available within Trilinos, as well as additional verification, remains for future work. In addition, we plan to further study the numerical methods associated with the non-linear system solution to identify approaches that are most appropriate for power systems under stress. It is well known that the system Jacobian matrices can become ill-conditioned under high-stress conditions (e.g., voltage collapse), and therefore work is needed to identify routines that maintain numerical stability for these highly stressed cases.

Acknowledgement

The authors wish to thank Trilinos developers who have provided helpful suggestions for this work, particularly Andy Salinger for assistance with the Piro package. Thanks are also due to Siva Rajamanickam for his assistance in interpreting the results from UMFPACK and KLU, and Juan Torres and Jeff Marshall for facilitating the research exchange between the University of Vermont and Sandia.

The authors also gratefully acknowledge financial support from the U.S. Department of Energy Award #DE-OE0000447 and a workforce development sub-award from Sandia National Laboratories, Inter-Entity Work Order #M610000767.

5. REFERENCES

[1] Y. Chen, Z. Huang, and D. Chavarría-Miranda. Performance evaluation of counter-based dynamic load balancing schemes for massive contingency analysis with different computing environments. In *Proceedings of the IEEE Power and Energy Society General Meeting*, pages 1–6. IEEE, 2010.

[2] CRSA, RTE, TE, and TU/e. Algorithmic requirements for simulation of large network. Technical report, PEGASE Consortium, 2011.

[3] C. M. Davis and T. J. Overbye. Multiple element contingency screening. *Power Systems, IEEE Transactions on*, 26(3):1294–1301, 2011.

[4] T. Davis. *Direct methods for sparse linear systems*, volume 2. Society for Industrial Mathematics, 2006.

[5] T. A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30:165–195, June 2004.

[6] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li, and J. W. H. Liu. A supernodal approach to sparse partial pivoting. *SIAM J. Matrix Analysis and Applications*, 20(3):720–755, 1999.

[7] I. Dobson, J. Kim, and K. R. Wierzbicki. Testing branching process estimators of cascading failure with data from a simulation of transmission line outages. *Risk Analysis*, 30(4):650–662, Apr. 2010.

[8] G. Ejebe and B. Wollenberg. Automatic contingency selection. *IEEE Transactions on Power Apparatus and Systems*, PAS-98:97 – 109, 1979.

[9] R. Fitzmaurice, A. Keane, and M. O'Malley. Effect of short-term risk-aversive dispatch on a complex system model for power systems. *IEEE Transactions on Power Systems*, 26(1):460–469, 2011.

[10] M. A. Heroux, R. A. Bartlett, V. E. Howle, R. J. Hoekstra, J. J. Hu, T. G. Kolda, R. B. Lehoucq, K. R. Long, R. P. Pawlowski, E. T. Phipps, A. G. Salinger, H. K. Thornquist, R. S. Tuminaro, J. M. Willenbring, A. Williams, and K. S. Stanley. An overview of the trilinos project. *ACM Transactions on Mathematical Software*, 31(3):397–423, 2005.

[11] S. Jin, Z. Huang, Y. Chen, D. Chavarría-Miranda, J. Feo, and P. Wong. A novel application of parallel betweenness centrality to power grid contingency analysis. In *Parallel & Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1–7. IEEE, 2010.

[12] E. Keiter, H. Thornquist, R. Hoekstra, T. Russo, R. Schiek, and E. Rankin. Parallel transistor-level circuit simulation. In *Simulation and Verification of Electronic and Biological Systems*. Springer, 2011.

[13] C. T. Kelley. *Iterative methods for linear and nonlinear equations*. SIAM, 1995.

[14] S. Khaitan and J. McCalley. A class of new preconditioners for linear solvers used in power system time-domain simulation. *Power Systems, IEEE Transactions on*, 25(4):1835–1844, 2010.

[15] S. Khaitan, J. McCalley, and Q. Chen. Multifrontal solver for online power system time-domain simulation. *Power Systems, IEEE Transactions on*,

23(4):1727–1737, 2008.

[16] P. Kundur. *Power System Stability and Control*. Electric Power Research Institute/McGraw-Hill, 1993.

[17] NERC. *Reliability Standards for the Bulk Electric Systems of North America*. North American Electric Reliability Corporation, May 2009.

[18] H. Ren, I. Dobson, and B. Carreras. Long-term effect of the n-1 criterion on cascading line outages in an evolving power transmission grid. *Power Systems, IEEE Transactions on*, 23(3):1217–1225, 2008.

[19] M. Sala, K. Stanley, and M. Heroux. Amesos: A set of general interfaces to sparse direct solver libraries. In *Proceedings of PARA'06 Conference, Umea, Sweden*, 2006.

[20] O. Schenk and K. Gärtner. Solving unsymmetric sparse systems of linear equations with PARDISO. *Fut. Gen. Comput. Sys.*, 20(3):475–487, 2005.

[21] K. Stanley and T. Davis. KLU: a 'Clark Kent' sparse LU factorization algorithm for circuit matrices. In *SIAM Conference on Parallel Processing for Scientfic Computing (PP04)*, 2004.

[22] D. Stewart. A high accuracy method for solving odes with discontinuous right-hand side. *Numerische Mathematik*, 58(1):299–328, 1990.

[23] L. Trefethen and D. Bau. *Numerical linear algebra*. Society for Industrial and Applied Mathematics, Philadelphia, 1997.

[24] M. Vaiman, K. Bell, Y. Chen, B. Chowdhury, I. Dobson, P. Hines, M. Papic, S. Miller, and P. Zhang. Risk assessment of cascading outages: Methodologies and challenges. *Power Systems, IEEE Transactions on*, 2011.

[25] R. Zimmerman, C. Murillo-Sánchez, and R. Thomas. Matpower: Steady-state operations, planning, and analysis tools for power systems research and education. *IEEE Transactions on Power Systems*, 26(1):12 –19, 2011.