Electric Power Systems Research 141 (2016) 22–30

Contents lists available at ScienceDirect

Electric Power Systems Research

journal homepage: www.elsevier.com/locate/epsr

Power grid simulation applications developed using the GridPACK™ high performance computing framework

Shuangshuang Jin*, Yousu Chen, Ruisheng Diao, Zhenyu (Henry) Huang, William Perkins, Bruce Palmer

Pacific Northwest National Laboratory, Richland, WA 99352, USA

ARTICLE INFO

Article history:
Received 12 February 2016
Received in revised form 6 June 2016
Accepted 9 June 2016
Available online 30 July 2016

Keywords:
High performance computing
Parallel programming
Power system computation
Power system dynamics
Dynamic simulation
Contingency analysis

ABSTRACT

The need for accelerating power grid simulation through high performance computing (HPC) has long been recognized, and prior efforts have been devoted to developing one-off parallel computing applications for particular power grid functions. Non-transferable software codes and duplicated implementations in these prior efforts are a major barrier to more widespread HPC adoption in power grid applications. Modern HPC hardware and architecture require significant computing expertise for application development. The GridPACK™ software framework described in this paper provides an HPC-compatible software structure to access modern parallel solvers and HPC-ready modules for common components in power grid simulation applications. GridPACK hides the HPC details and enables power system developers to focus on applications instead of computational details. Several example applications of GridPACK are presented to demonstrate the capabilities of GridPACK and the performance of HPC simulations with large power grid networks. Examples discussed include: a dynamic simulation application capable of running a 17,156-bus Western Electricity Coordinating Council (WECC) system in a computational speed faster than real time (e.g., under 30 s for a 30-s simulation), a static contingency analysis application using a task manager, and a dynamic contingency analysis application utilizing two levels of parallelism. These example applications illustrate GridPACK's capabilities to support different types of simulations within a unified framework and to support reuse of transferable software codes across power grid applications. The computational results indicate strong performance improvements for power grid simulations with GridPACK.

1. Introduction

Despite the complexity of the modern power grid, commercial software for simulating its behaviors is mainly designed for sequential computation on single-processor workstations. High performance computing (HPC), which is capable of tackling much larger problems and solving them in much shorter time, has made few inroads into power grid simulation for real-world applications. A main reason that HPC is not more widespread in power grid simulation is that the complexity and cost of developing parallel algorithms and implementing them in parallel software are high. The need for high performance computing has been recognized for decades, and many researchers have devoted efforts to this topic. These prior efforts examined the HPC topic from individual application perspectives without a systematic view on how applications can benefit from a common set of software modules. For example, references [1–5] discuss parallel implementation of a state estimation application; [6–9] introduce massive static contingency analysis that utilizes a dynamic load-balancing scheme, and [10–14] discuss multiple implementations of dynamic simulation. As a result, each of these efforts required significant parallel programming skills as well as power engineering expertise, and the resulted outcome is a one-off implementation. The barrier to adopting these HPC applications is high. We recognize that developing HPC-based power grid applications typically requires strong parallel programming skills that are outside the focus of most power grid modelers and power grid modelers are more interested in developing their modeling capabilities than improving performance using complex programming techniques. It is therefore highly desirable to provide a common set of reusable parallel software modules on a unified framework supporting multiple applications and thus to lower the barrier to the adoption of HPC for power grid simulation.

* Corresponding author.
E-mail addresses: shuangshuang.jin@pnnl.gov (S. Jin), Yousu.chen@pnnl.gov (Y. Chen).

http://dx.doi.org/10.1016/j.epsr.2016.06.024
0378-7796/

Power system applications typically contain many common components such as the formulation of an admittance matrix (i.e. Y-bus matrix) [15], or contain component simulations that couple with each other. In addition, some common functions such as solving the power flow equations or integrating differential-algebraic equations for dynamic simulation, are used across multiple applications. For example, power flow solution is an important part of contingency analysis, dynamic simulation, and other applications. Reproducing these functions in different applications leads to duplication of efforts and reduces programmers' productivity. Efforts are underway to promote software reuse and simplify the process of developing parallel software codes for grid applications. The GridPACK, as a result of such efforts, provides open-source software for use by the power grid community to accelerate the adoption of high performance computing.

The goal of GridPACK is to simplify the task of developing parallel software for power grid applications that can run on high performance architectures. GridPACK accomplishes this goal by encapsulating many function modules of power grid computation in high level abstractions that hide from users the details of data communication and distribution, which are necessary for parallel software implementation. Application developers can focus on the physics and mathematics of framing and modeling their problems, without being overwhelmed by the bookkeeping required to distribute data over multiple processors and to access it in a distributed context. These abstractions also make it simpler for users to access high performance computing libraries, as well as promoting software reuse and reducing the costs of software development and maintenance.

This paper will describe the development and performance of several power system applications to illustrate the functionality and benefits of GridPACK. The rest of this paper is organized as follows: Section 2 provides a brief description of GridPACK. Section 3 discusses the example implementation of several applications using GridPACK. Section 4 concludes the paper with a discussion of future work.

2. Brief overview of GridPACK software framework

The applications described later in this paper have been developed using the GridPACK software framework so they can run on HPC platforms. GridPACK is designed to provide foundational functionality tailored for power grid simulation while also maintaining the flexibility to support a wide variety of applications. The framework contains modules for creating distributed representations of power grid networks, supporting distributed matrices and vectors, supplying linear and non-linear solvers, and mapping data between networks and matrices. In addition, GridPACK provides support for many additional computing functions, such as input/output modules, task management, error handling, and software profiling.

The framework is written in C++ and makes use of its object-oriented programming features. This enables GridPACK to encapsulate much of the bookkeeping associated with parallel programming in high-level modules, as well as promoting software reuse. Both the inheritance functionality and the software template features of C++ are used extensively in GridPACK.

GridPACK has been successfully used in developing software codes for power grid applications such as power flow solution, dynamic simulation, state estimation, contingency analysis, and real-time path rating calculations. Many of these applications use the same core functionality and libraries, demonstrating the key goal of developing GridPACK in the first place. Furthermore, software codes written for one application can readily be reused in other applications. One example is the code for computing the Y-bus matrix. Y-matrix appears in almost all power grid simulation applications and most implementations can share a common set of Y-bus matrix codes. With the GridPACK framework, improvements to the Y-bus matrix module in one application are automatically accessible and transferable for other applications.

The GridPACK framework is given in Fig. 1, with major modules indicated. The object-oriented programming principle is extensively followed in GridPACK implementation. Objects in this GridPACK context are defined as the components in a power grid such as buses, branches, and generators, such that it is a natural and easy way to provide and maintain input data to power grid applications. A brief description of the major modules in GridPACK is given below. More details of the overall design of the GridPACK framework and its modules are described in [16].

2.1. Network

The network module is a template class that allows users to include arbitrary models for the power grid bus and branch components in the network. Depending on the application, the bus and branch components can generate appropriate equations for power flow formulation, state estimation, and dynamic simulation, etc. The network module supports distribution of the power grid network over multiple processors through a partitioner. The partitioner divides the network in such a way as to maximize the number of connections between the buses allocated to the same processor, as well as minimizing the number of connections between buses allocated on different processors. This way minimizes the computational overhead introduced by data exchange between processors. The existing partitioner is based on the Parmetis partitioning software [17].

The network module also manages data exchanges between ghost buses and ghost branches (which are copies of buses and branches that are located on other processors, but are connected to locally-owned buses). Finally, the network module provides access to buses and branches on each processor. These capabilities are essential in defining the properties of the power grid model and implementing it on a parallel computer.

2.2. Mappers

One of the more complicated aspects of writing parallel codes for power grid applications is generating distributed matrices from a network that has been partitioned over multiple processors. The mapper modules and the matrix-vector interface in the network components are designed to simplify this task. Each bus and branch in the network is responsible for describing its contribution to a matrix or vector. This is achieved by implementing functions in the components that describe how many matrix elements each bus or branch is contributing, as well as functions that return a list containing the values of these elements. These functions also provide information on how the matrix elements are arranged. For example, a bus that contributes a square block to the diagonal of a matrix will return the dimensions of the block.

The mappers use the information returned from the buses and branches to build a complete distributed matrix. The mappers loop over all components on each processor and query the functions describing the number of elements and their values. This information can be used to determine the dimensions of the matrix and the locations of all matrix elements. The user is only responsible for supplying the matrix elements contributed by each network component. The index calculation for building the complete distributed matrix is handled by the mappers.

2.3. Math

The math module is responsible for supporting the creation and manipulation of distributed matrices and vectors. It also supplies

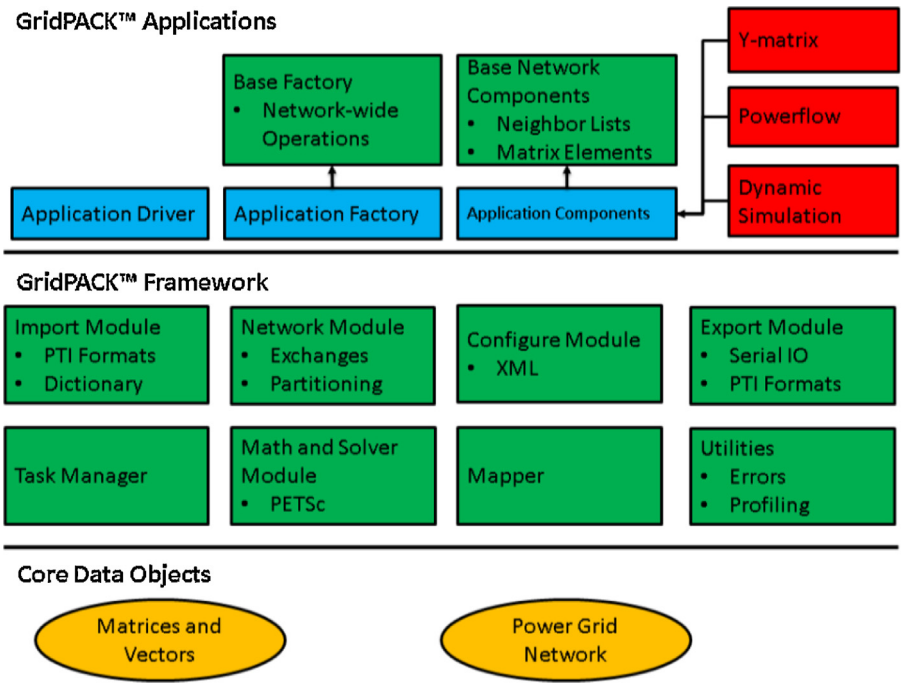

Fig. 1. A schematic diagram of the GridPACK software framework. Framework modules are colored green, user-supplied modules are colored blue. User modules that can be used across multiple applications are colored red.

the linear and non-linear solvers used by applications to implement their solution algorithms. The math module is currently built on top of the PETSc library [18], but it is designed to support a uniform interface that can be implemented using other libraries. Switching between different libraries would not require any changes to the application code, but could be accomplished by linking to a version of GridPACK that was built using different configuration options.

The matrix and vector classes in GridPACK are currently implemented on top of the corresponding classes in PETSc and work seamlessly with the PETSc linear and non-linear solvers. The different solvers available in PETSc are all accessible through their run-time options database. This simplifies user programming considerably since there is only one linear and one non-linear solver class in GridPACK. All the different solver options can then be accessed by specifying different options in the input deck. This makes the implementation of different solvers and preconditioners extremely straightforward.

2.4. Application Modules

Several common applications that have been written using GridPACK have been recast as modules and included in the GridPACK software package. These application modules are designed to be called as parts of other programs and allow developers to string several different types of calculations together. They are also useful for setting up contingency-type calculations where multiple instances of the same type of calculation are run concurrently. These types of calculations are supported by the network clone function, which allows users to copy a network for one application (e.g., power flow) to a network for another application (e.g., dynamic simulation). The availability of application modules simplifies the task of creating more complicated workflows in which the results of one type of calculation are fed into the input of another calculation. Examples include different types of contingency analysis, initialization of dynamic simulation using either power flow or state estimation calculations, and a real-time path rating calculation that combines power flow, contingency analysis, dynamic simulation, and voltage stability analysis into one workflow. Existing modules in GridPACK include power flow, state estimation and dynamic simulation. As more applications are developed using GridPACK, they can be converted to modules as well.

3. Examples of applications

This section discusses the parallel implementation and performance of example power grid applications using GridPACK, focusing on the broad categories of dynamic simulation and static and dynamic contingency analysis functions.

3.1. Dynamic simulation

Dynamic simulation is a core function in power system modeling and has the ability to evaluate transient trajectories that occur when there are abrupt changes in the system. It is important to understand these dynamic fluctuations, since instantaneous values of system variables can differ substantially from the values that are obtained during steady-state operation; parameters that may lie within safe operating ranges during steady-state operation can exceed safety margins under dynamic conditions. GridPACK has implemented a dynamic simulation module [19] that is capable of simulating the response to disturbances in a speed faster than real time for large-scale power systems (e.g., 30-s simulation accomplished in less than 30 s). Results shown below are for the 17,156-bus Western Electric Coordinating Council (WECC) system of North America, using typical 5-ms time steps without model reduction. The code was run on Constance supercomputer located in the Pacific Northwest National Laboratory's Institutional Computing (PIC) facility. The Constance supercomputer comprises about 300 Haswell nodes with an Infiniband QDR interconnect; each node is a dual socket with 16 AMD Interlagos cores per socket; and each core runs at 2.1 GHz with 64 GB of 1600 MHz memory (32GB/socket).

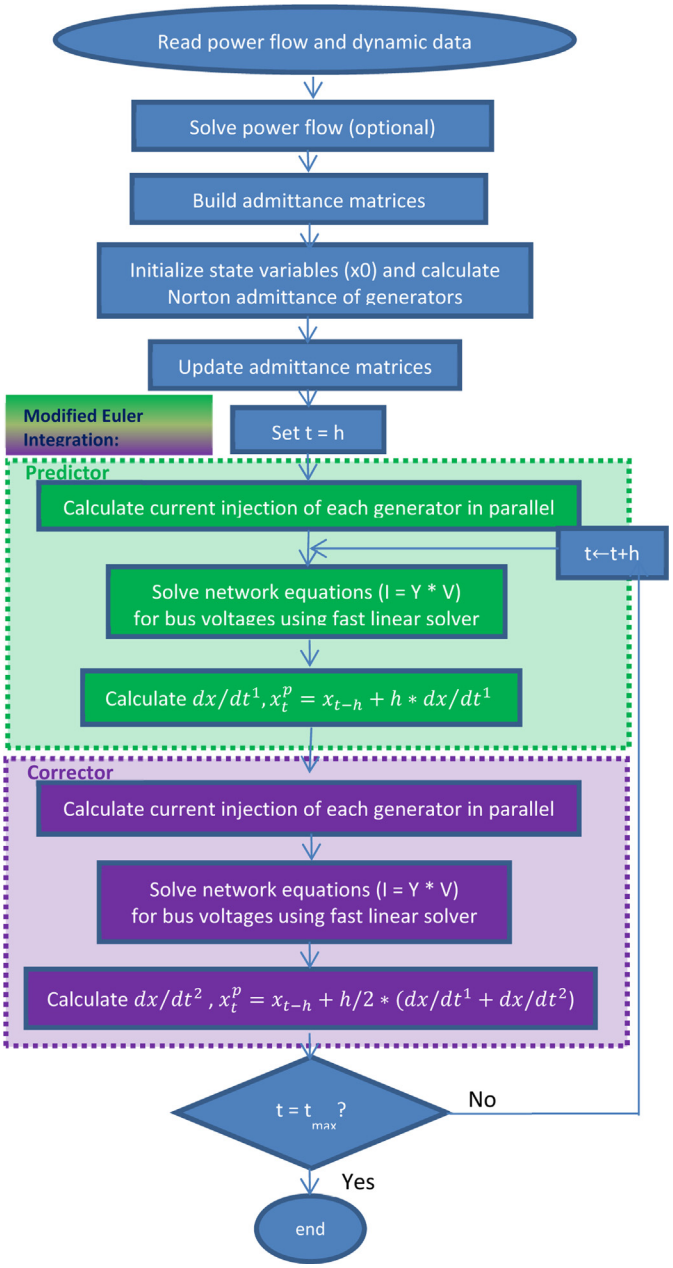

Fig. 2. Flowchart of dynamic simulation with "full Y-bus" matrix and modified Euler method integration.

3.1.1. Algorithm

The Y-bus matrix is a very sparse matrix representing the electrical connections between buses and describing the relationship between the voltages, currents, and power flows in a power system.

Most of today's existing commercial software uses the original full Y-bus matrix in dynamic simulation computations. Consistently, in this GridPACK application, the "full Y-bus" algorithm is implemented using the modified Euler method for time integration to create a parallel dynamic simulation module. Fig. 2 shows the main flowchart of the "full Y-bus" algorithm.

Given the property that the network module of GridPACK distributes the power system network over multiple processors and provides access to buses and branches on each processor, building the "full Y-bus" matrix in parallel is straightforward. This is accomplished by generating the equations for the bus and branch components in the network using the mapper module. The mapper extracts contributions to the Y-bus matrix from each bus and branch component (or object) and then assembles them into a fully distributed matrix that can be used in subsequent calculations. This process substantially simplifies building the matrix, since it eliminates the need for users to determine how to partition the matrix across processors, as well as the need to determine the global indices of each matrix element.

The modified Euler method is used to compute the system state variables at each step through time integration of differential equations. The differential equations describe the system dynamics of generating units and controllers, and they are naturally decoupled, suitable for easy distribution to different processors of the high performance computing resource. For example, the equations of a generator are distributed to the processor which contains the bus this generator connects to, given that the buses have already been distributed when the network is partitioned by GridPACK.

The implementation of the dynamic simulation application using the "full Y-bus" method uses a mixed strategy, where part of the algorithm is run in parallel and part of the algorithm is run sequentially. The target calculation for the dynamic simulation is to run a 30-s simulation of a fault on the 17,156-bus 1906-generator WECC system in less than 30 s. After carefully evaluating available linear solvers, a direct linear solver with fast lower and upper (LU) triangular decomposition was chosen to solve the network equations $\mathbf{Y}^*\mathbf{V} = \mathbf{I}$, where $\mathbf{V}$ is the voltage vector and $\mathbf{I}$ is the vector representing current injections to the buses. This LU solver, running sequentially on a single processor, exhibits the fastest performance with the same level of accuracy for this WECC system. Three setups of this solver, representing the pre-fault, fault-on and post-fault Y-bus matrices, are created within the application. If no non-linear loads are included in the model, these Y-bus matrices are constant during the simulation. Therefore, once a solver is set up in GridPACK, only the right-hand-side $\mathbf{I}$ vector needs to be updated at each time step.

The solution of the network equations at each time step takes the distributed right-hand-side $\mathbf{I}$ vector and performs an "all-gather" command. This "all-gather" command gathers the right-hand-side data held locally on each processor and creates a replicated version of the full right-hand-side vector on each processor. The matrix has already been replicated on each processor when the solver is originally set up and does not need to be updated again during the course of the simulation. Each process then solves the algebraic equations and copies the portion of the solution vector that is locally owned back into a distributed copy of the solution vector. The ability to configure the linear solver to run in this way has been added as a run-time option to the linear solvers in GridPACK, so there is no need to make any adjustments to the actual application code to access this functionality.

3.1.2. Validation

A WECC 179-bus system is run to compare the accuracy against the commercial PowerWorld Simulator (PWS). This model represents a simplified structure of the WECC system, consisting of 179 buses, 29 generators, 104 loads, 40 fixed shunt elements, 203 transmission lines, and 60 transformers. The Generator model is a GENSAL generator model (Salient pole generator represented by equal mutual inductance rotor modeling), with an exciter model EXDC1 (IEEE (1980) DCI excitation system model with added speed multiplier) and a governor model WSIEG1 (IEEE steam turbine/governor model with deadband and nonlinear valve gain added). Table 1 shows the type of 494 state variables being simulated in this model. (More detailed information about the state variables and control block diagram of the exciter and governor models can be found in [20]). The system is simulated for 20 s with a time step of 5 ms. A fault that induces a line trip between two buses is added at 1.00 s and cleared at 1.05 s.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Configuration>
3    <Powerflow>
4      <networkConfiguration>  WECC179_detailed.raw </networkConfiguration>
5      <maxIteration>50</maxIteration>
6      <tolerance>1.0e-3</tolerance>
7      ...
8    </Powerflow>
9    <Dynamic_simulation>
10     <generatorParameters>  WECC179_detailed.dyr </generatorParameters>
11     <simulationTime>20</simulationTime>
12     <timeStep>0.005</timeStep>
13     <faultEvents>
14       <faultEvent>
15         <beginFault> 1.00</beginFault>
16         <endFault>   1.05</endFault>
17         <faultBranch>5 6</faultBranch>
18         <timeStep>   0.005</timeStep>
19       </faultEvent>
20     </faultEvents>
21     <LinearSolver>
22       <ForceSerial>true</ForceSerial>
23       <InitialGuessZero>true</InitialGuessZero>
24       <SerialMatrixConstant>true</SerialMatrixConstant>
25       <PETScOptions>
26         -ksp_type preonly
27         -pc_type lu
28         -pc_factor_mat_ordering_type amd
29       </PETScOptions>
30     </LinearSolver>
31   </Dynamic_simulation>
32 </Configuration>
```

Fig. 3. Input deck for dynamic simulation application on GridPACK.

Fig. 3 shows the input deck for the configuration options of performing this specific dynamic simulation using GridPACK. The GridPACK module used for reading input decks is designed to work with XML-formatted inputs and supports a simple hierarchical data format [21]. This format is compatible with many other types of software tools, and also provides great flexibility and extensibility. New features can be easily added without requiring significant modifications to the input format or existing parsers. Note that the choice of solver is specified in the PETScOptions block (lines 25–29) and can be changed by modifying these lines without changing any of the underlying source codes. This enables users to experiment with a variety of solvers.

Figs. 4 and 5 show a comparison between the dynamic simulation using the GridPACK and PowerWorld simulators for generator rotor angle and speed on the faulted bus of the WECC 179-bus system. The pre-disturbance and during-fault periods showed perfect match between the two curves, indicating the network interface calculation of generator models and network solutions are accurate. The difference is accumulating after the fault is cleared. There are a few factors identified, indicating (1) in GridPACK implementation, we are using modified Euler method for numerical integration while PowerWorld uses forward Euler and 2nd order Runge–Kutta methods; (2) the saturation function of nonlinear

Table 1
State variables being simulated in the WECC 179-bus system.

State variable	Generator	Exciter	Governor
1	Rotor Angle	VE	X1_gov
2	Rotor Speed	Terminal Voltage	X2_gov
3	Transient Q Axis Eq	X3_exc	X3_gov
4	Transient D Axis Flux	X4_exc	X4_gov
5	Subtransient Q Axis Flux	X5_exc	X5_gov
6			X6_gov

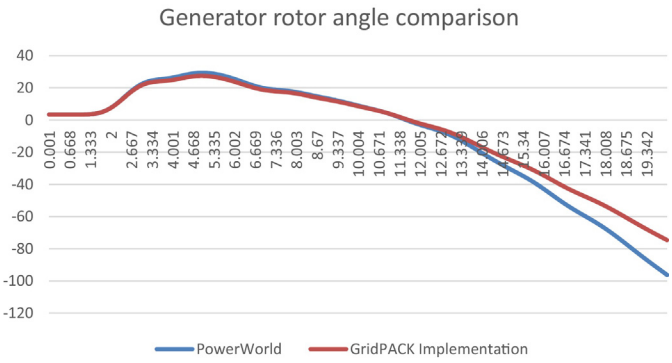

Fig. 4. Comparison of generator rotor angle on the faulted bus of the WECC 179-bus system between dynamic simulation using PowerWorld simulator and GridPACK.

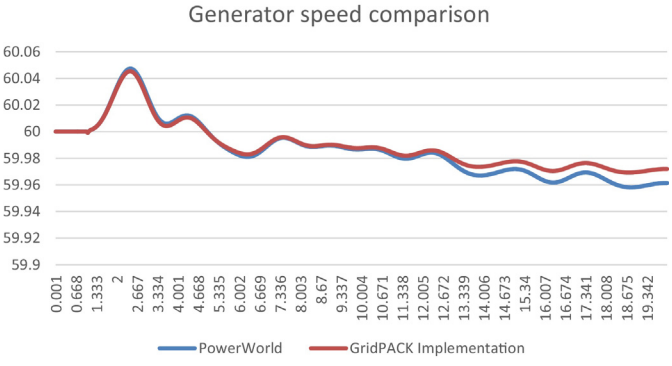

Fig. 5. Comparison of generator rotor speed on the faulted bus of the WECC 179-bus system between dynamic simulation using PowerWorld simulator and GridPACK.

Table 2
Total computation time of the 30 s simulation of WECC 17,156-bus system.

Time spent (s)	1p	2p	4p	8p	16p
Total time	64.47	46.85	33.82	26.82	24.80
Initial power flow	6.53	8.26	6.9	6.62	6.65
Dynamic simulation time integration	56.16	37.37	26.32	19.86	17.96
Linear solver in Modified Euler prediction step	3.76	5.74	6.38	6.73	7.39
Linear solver in Modified Euler correction step	3.75	5.69	6.35	6.64	7.30

blocks in generator, exciter and governor models may be different between GridPACK and PowerWorld; (3) the precision of the codes is different–GridPACK is using double precision while PowerWorld uses single precision. In the future work, we plan to reduce the difference by consulting with commercial vendors. We envision a close match after the implementation details are made consistent with PowerWorld.

3.1.3. Performance

A WECC system consisting of 17,156 buses and 1906 in-service generators (30,496 state variables) is simulated in GridPACK with a fixed time step of 5 ms for 30 s. A 3-phase-to-ground fault occurs at 1.0 s and is cleared at 1.05 s on a major transmission line during the simulation. The total computation time of the simulation with different numbers of processors is shown in Table 2. The scaling behavior of the GridPACK-based dynamic simulation application for the WECC system is shown in Fig. 6. The figure shows the total simulation time and time breakdown by algebraic solution using the linear solver, integration of differential equations, and one-time power flow calculation for initializing the dynamic simulation. The integration time includes the linear solver time. The time for the power flow calculation also includes the time to read the network configuration file and to distribute the data across processors.

Typically, it takes 150 s or more to run a 30-s WECC-size no-fault simulation using commercial tools on a workstation with 2.4 GHz Duo Core and 4GB of RAM. In the GridPACK implementation, the total simulation time has a minimum at 16 processors and takes about 24 s, which is about 6 s faster than real time. The integration time only takes 17 s. The linear solver is very fast on one processor and takes 7 s. On two processors, the linear solver time jumps up to 13 s and then slowly rises. The initial jump reflects the cost of replicating the right hand side vector on all processors and then copying the solution back to a distributed vector. Additional increases reflect the cost of communicating over more processors.

The gap between the time spent in the solver and the time spent in the integration loop decreases steadily up to 16 processors,

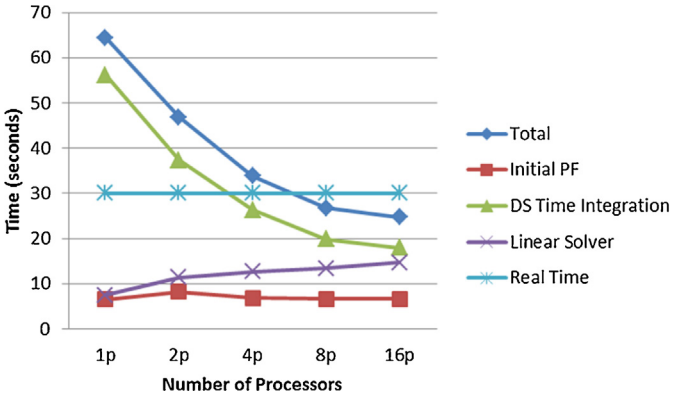

Fig. 6. Performance of dynamic simulation application for the 17,156-bus WECC system.

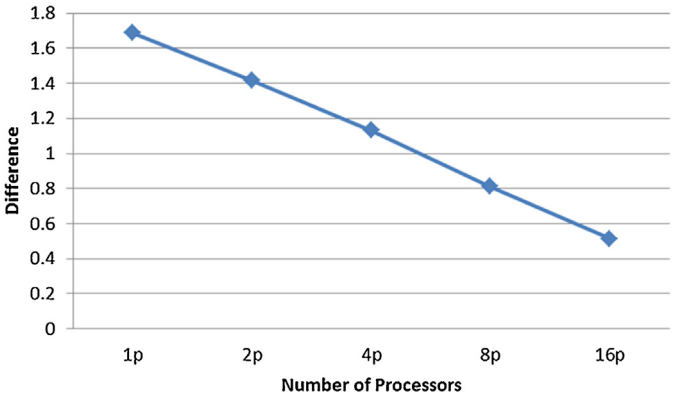

Fig. 7. Difference between time spent in integrating dynamical equations and time spent in linear solver.

showing that the non-solver portions of the time integration loop are highly scalable. This gap represents the time spent in functions such as calculating the Norton impedance, which are completely decoupled from each other and require no interprocessor communication.

The difference between the time spent in integrating the dynamic simulation and the time spent in the linear solver is plotted in Fig. 7, which shows near perfect scalability up to 16 processors.

The linear solver calls are not parallelized and do not scale, but even with the overhead of the "all-gather" operation, they are faster than any parallel solver for problems of this size. As shown in Fig. 7, the remaining parts of the algorithm mostly consist of loops over network components that do scale quite well, since the number of network components per processor is proportional to the size of the network divided by the number of processors. Overall, the combination of using a very fast sequential solver, coupled with parallelizing the non-solver portions of the calculation in GridPACK, is enough to achieve faster-than-real-time performance on 8–16 processors.

3.2. Static and dynamic contingency analysis

Contingency analysis is an important function in power system operation and security analysis. It is used to check physical limitations and electrical stability security constraints in a number of scenarios with power system component failures, i.e. contingencies. Contingency analysis takes input from the results of state estimation, which is formulated around power flow equations. Power system operation is trending towards maximizing utilization of existing infrastructure with leaner stability margins. Because of this, contingency analysis is becoming increasingly important in modern power system operation.

Contingency analysis has two categories: static contingency analysis and dynamic contingency analysis. In general, static contingency analysis only considers the system's behavior in a certain snapshot under different contingencies. It requires solving a large set of power flow equations, which is typically achieved by solving linear equations iteratively based Newton's method. Because solving such linear equations for current realistic models is not very scalable as shown in the dynamic simulation example, a sequential solver is typically applied. But the parallelism comes from the fact that contingency scenarios can be distributed across multiple processors so they can be performed simultaneously. This is the task-level parallelization GridPACK is capable of.

Dynamic contingency analysis is used to study system's behaviors over a certain period of time under different contingencies. It requires the solution of a large set of dynamic simulation problems, one for each contingency, to determine the dynamic stability of the

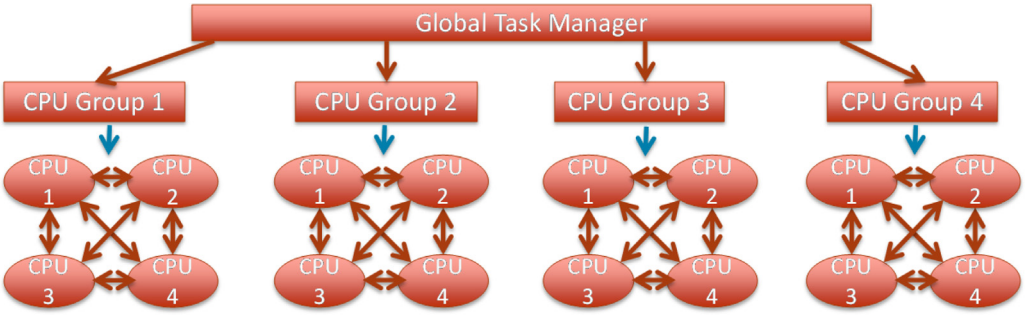

Fig. 8. Two-level parallelism concept. Individual tasks are distributed to groups (first level of parallelism) and run in parallel within the group (second level of parallelism).

system. As discussed in Section III.A, parallel computing can reduce the computational time for individual dynamic simulation runs. At the task level, dynamic contingency analysis is parallelized in the same way as static contingency analysis. The following subsections describe the GridPACK task manager module that is designed to manage tasks across multiple processors.

3.2.1. Task manager and two-level parallelism

The contingency analysis functions are built with the help of a task manager module implemented in the GridPACK framework to support load-balanced task distribution and two-level parallelism. The task manager is designed to assign tasks to processors based on the availability of processors and is implemented via a global atomic read-modify-write operation [6,7]. It can be used to efficiently manage large-scale, scenario-based simulations where minimum communication is required among processors, e.g., "embarrassingly parallel" simulations.

Contingency analysis consists of a set of simulations for relatively independent scenarios. The computational process of contingency analysis can be sped up in two approaches: the task-level parallelization as mentioned above, and the parallelization of individual simulation as described in Section III.A. Combining these two approaches is referred to as two-level parallelism. GridPACK supports this type of programming through two mechanisms. The first is through the availability of the task manager module. The second implementation is a result of how all GridPACK modules are based, either explicitly or implicitly, on a communicator that defines a group of interacting processors.

Fig. 8 shows a schematic diagram illustrating the concept of two-level parallelism supported by GridPACK. At the top level, individual contingencies are distributed to different groups of processors. At the lower level, each contingency is further parallelized among multiple processors. This can be accomplished easily by splitting the global communicator for the system into smaller groups, and then creating separate GridPACK applications on each subgroup. If the application already exists, then it is usually straightforward to develop the small amount of driver codes necessary for the contingency analysis application.

Two-level parallelism can be applied if individual contingency simulations are themselves scalable. This is the case for dynamic contingency analysis with multi-level parallelism. The power flow application is much less scalable, so static contingency analysis based on power flow analysis makes use of a single level of parallelism in this study. For reasons outlined in Section III.A, individual power flow calculations are run on a single processor.

The GridPACK task manager has a relatively simple interface and has only one or two important operations. The first is to specify the number of tasks to be executed. The second is a function, called nextTask, which returns an integer index for the next task. The value of the index runs between 0 and $N-1$, where N is the number of tasks. The atomic read-modify-write operation in the task manager guarantees that all task indices will be selected at least once and only once. The index is then translated into a unit of work. Any processor can call the nextTask function and receive the index of the next task on the task list. Processors that complete their tasks will immediately ask for another task, rather than waiting for processors that have selected slower tasks to complete. This results in automatic load balancing and minimizes processor idle time.

In GridPACK, a variant of the nextTask function that enables multi-level parallelism contains a parallel communicator defining a group of processors. All processors in the communicator receive the same task index. A parallel application can then be launched on this group of processors. The nextTask function also returns a Boolean value that is false if the number of tasks has been exceeded. This can be used by the application to shut down the loop over tasks and go to the next phase of the calculation.

For contingency analysis calculation, the global task manager allocates contingencies to processor groups, each containing multiple CPUs. Each group is responsible for performing a single parallel dynamic simulation, which can lower the time required for a single simulation compared to running on a single processor. This enables additional performance gains compared to running a single task on a single processor, and allows users to further decrease time-to-solution if large numbers of processors are available.

3.2.2. Static contingency analysis application

The static contingency analysis uses a power flow simulation to calculate the impact of different contingencies. Highly optimized sequential implementation of the power flow codes is utilized, as parallelization of individual power flow calculations does not provide additional speedup for this problem. For this analysis, only a single level of parallelism provides any benefit.

A logarithmic plot of the time GridPACK required to simulate 1024 contingencies using the WECC system is shown in Fig. 9. This plot shows good scalability out to about 128 processors, and then starts flattening out. The overall solution time improved from 2 h

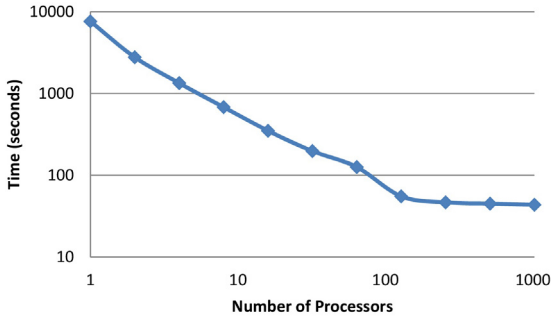

Fig. 9. Scaling behavior of static contingency analysis of 1024 contingencies using the WECC system.

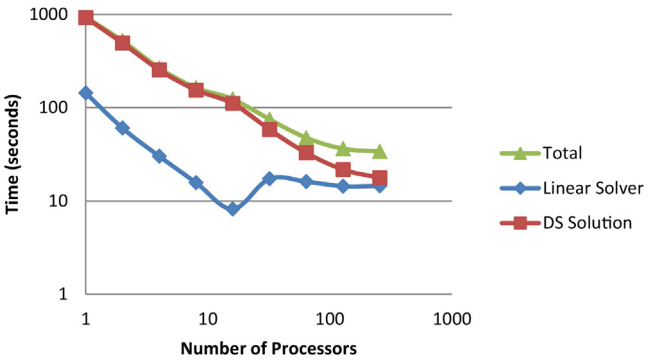

Fig. 10. Simulation of 16 contingencies on the WECC system.

to less than a minute. Additional improvements may be possible by analyzing the behavior of individual simulations. At small processor counts, each contingency requires about 7 s, on average.

As the number of processors increases and the number of tasks executed by each process decreases, the effect on the total execution time of tasks that take an unusually long time to complete becomes more pronounced. In this case, it was found that simulations where the Newton-Raphson iterations converged tended to use only a few iterations to reach a solution. Simulations that did not converge used the maximum allowable number of iterations before returning, and generally took much longer than the converged calculations. If more stringent criteria for shutting these tasks down sooner were available, then the elapsed time for the entire contingency analysis at large processor counts should improve. Reducing the maximum number of Newton-Raphson iterations in the GridPACK power flow solution from 50 to 10 resulted in a 30% reduction in the amount of time required to run the simulation at 128 processors. Of course, this strategy must be pursued carefully, because reducing the limit too much may result in calculations that don't converge, even when they should.

3.2.3. *Dynamic contingency analysis application*

Contingency analysis can also be applied to dynamic simulation of the power grid and is typically used to study system dynamic behaviors. It can determine system transient stability subject to a selected set of contingencies, where system violations may show up in the transients that occur when the system moves from one steady-state to the next, even if the steady-states themselves show no violations. Examples using dynamic contingency analysis are path rating studies and system planning studies.

A dynamic contingency analysis application using two-level parallelism has been implemented using the GridPACK framework. A test problem consisting of 16 30-s long simulation contingencies based on the WECC system was constructed to investigate the scaling properties of two-level parallelism. Note that for larger numbers of contingencies, the potential performance gains are also larger if sufficient processors are available. The total time taken for different processor counts to simulate 16 contingencies are shown in Fig. 10. The figure shows good scaling up to about 64 processors and then starts to flatten out, although small decreases in execution time are seen all the way out to 256 processors. The important feature of these results is that scaling continues beyond 16 processors (one core per simulation) and additional benefits from parallelizing individual simulations are achieved up to 256 processors. At this point, the execution time for all 16 contingencies is about 33 s.

Fig. 10 also shows the amount of time spent in the solver portion of the code and the time spent solving the dynamic simulation equations. The GridPACK solver portion of the code scales perfectly to 16 processors and then jumps up and plateaus. This is the point at which individual simulations start running on more than 1 core, so the jump reflects the overhead of distributing the right-hand-side vector to all processors. The time spent in solving the dynamic simulation equations track the total time fairly closely until 16–32 processors and then it starts to diverge. Similar to the results shown in Section III.A, the difference between the total time and the dynamic simulation time reflects overhead associated with distributing task to different processors. It may also reflect load imbalance that becomes more significant at large processor counts. However, the magnitude of the gap is fairly large and may warrant additional investigation, with the hope of realizing additional performance gains.

4. Conclusions

The GridPACK software encapsulates parallel programing concepts using high-level abstractions and provides a modularized framework for simplifying the development of power system applications on high performance computing (HPC) platforms. Facing the dilemma between the strong desire for HPC applications and the growing complexity of HPC programming, GridPACK provides a solution by hiding the HPC programming details from application developers. The main benefit of the GridPACK framework is that power system developers can focus on applications instead of being overwhelmed by HPC programming details, and thus the HPC applications can be developed more efficiently and maintained more cost-effectively. Dynamic simulation and contingency analysis applications are presented in this paper as examples of the use of GridPACK. These example applications successfully demonstrated the capabilities of GridPACK in the following key aspects: ready support to multiple applications via an HPC-compatible unified framework; reuse of common HPC-ready functional modules across applications; easy access to modern mathematical solvers in parallel software; two-level parallelism implementation for additional scalability and performance improvement; and strong scalability and performance improvements enabled by the HPC modules provided by GridPACK. In particular, the large-scale dynamic simulation with a 17,156-bus power system achieved excellent faster-than-real-time performance; and the dynamic contingency analysis application – a key part of real-time dynamic security assessment – can extract additional performance improvement through the two-level parallelization approach. These demonstrate GridPACK's core framework functionalities to enable parallel programming for power grid simulations on parallel computing architectures. Work is ongoing to build additional applications using GridPACK and to extend the GridPACK framework to support additional types of power grid applications.

GridPACK is open-source software and is freely available for download at https://www.gridpack.org.

Acknowledgements

Funding for this work was provided by the U.S. Department of Energy's Office of Electricity Delivery and Energy Reliability through its Advanced Grid Modeling Program. Additional funding was provided by the Future Power Grid Initiative at Pacific Northwest National Laboratory through the Laboratory Directed Research and Development program. The authors gratefully acknowledge the support of Gilbert Bindewald with the U.S. Department of Energy.

References

[1] J.B. Carvalho, F.M. Barbose, A parallel algorithm to power system state estimation, in: Proc. POWERCON, 1998.
[2] A. Abur, P. Tapadiya, Parallel state estimation using multiporcessors, Electr. Power Syst. Res. 18 (1990) 67–73.

[3] D.M. Falcao, F.F. Wu, L. Murphy, Parallel and distributed state estimation, IEEE Trans. Power Syst. 10 (2) (1995).
[4] Y. Chen, M. Rice, Z. Huang, SCADA-rate parallel state estimation assessment with utility data, in: Proc. IEEE PES General Meeting, National Harbor, MD, 2014.
[5] Y. Chen, K.R. Glaesemann, M. Rice, Z. Huang, Sub-second state estimation implementation and its evaluation with real data, in: Proc. IEEE PES General Meeting, Denver, CO, 2015.
[6] Z. Huang, Y. Chen, J. Nieplocha, Massive contingency analysis with high performance computing, in: PES'09 IEEE, 2009, pp. 1–8.
[7] Y. Chen, Z. Huang, M. Rice, Evaluation of counter-based dynamic load balancing schemes for massive contingency analysis on over 10,000 cores, in: High Performance Computing, Networking, Storage and Analysis (SCC), SC Companion IEEE, 2012, pp. 341–346.
[8] S.K. Khaitan, J.D. McCalley, Dynamic load balancing and scheduling for parallel power system dynamic contingency analysis, in: High Performance Computing in Power and Energy Systems, Springer, Berlin/Heidelberg, 2013, pp. 189–209.
[9] A. Mittal, J. Hazra, N. Jain, Real time contingency analysis for power grids, in: Euro-Par 2011 Parallel Processing, Springer, Berlin/Heidelberg, 2011, pp. 303–315.
[10] Z. Huang, S. Jin, R. Diao, Predictive dynamic simulation for large-scale power systems through high-performance computing, in: High Performance Computing, Networking, Storage and Analysis (SCC), SC Companion, 2012, pp. 347–354.
[11] S. Jin, Z. Huang, R. Diao, D. Wu, Y. Chen, Parallel implementation of power system dynamic simulation, in: Power and Energy Society General Meeting (PES), IEEE, 2013, pp. 1–5.
[12] X. Zhou, Z. Han, F. Tian, Y. Li, F. Li, X. Tian, Concept and mechanism on full-process dynamic real-time simulation of power system with parallel-in-time-space, Power Syst. Technol. (POWERCON) (2010) 1–7.
[13] W. Xue, J. Shu, Y. Wu, W. Zheng, Parallel algorithm and implementation for realtime dynamic simulation of power system, in: Parallel Processing, ICPP, 2005, pp. 137–144.
[14] S. Abhyankar, A.J. Flueck, Real-time power system dynamics simulation using a parallel block-jacobi preconditioned Newton-GMRES scheme, in: High Performance Computing, Networking, Storage and Analysis (SCC), SC Companion, 2012, pp. 299–305.
[15] Ybus, Available online: http://en.wikipedia.org/wiki/Nodal_admittance_matrix.
[16] B. Palmer, W. Perkins, Y. Chen, S. Jin, D. Callahan, K. Glass, R. Diao, M. Rice, S. Elbert, M. Vallem, Z. Huang, GridPACK™: a framework for developing power grid simulations on high performance computing platforms, in: Proceedings of the Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC), 2014, pp. 68–77.
[17] G. Karypis, V. Kumar, Parallel algorithm for multilevel graph partitioning and sparse matrix ordering, J. Parallel Distr. Comput. 3 (1998) 71–95.
[18] PETSc: Portable, Extensible Toolkit for Scientific Computation, Available online: http://www.mcs.anl.gov/petsc.
[19] Z. Huang, B. Palmer, S. Jin, G. Bindewald, An open-source approach to accelerating power system dynamic simulation, abstract for panel session "Faster than Real-time dynamic simulation", in: Power and Energy Society General Meeting (PES), IEEE, 2014, 1-1.
[20] J. Weber, Implement a Transient Stability Calculation in Software, vol. 31, 2015, pp. 38.
[21] XML, Available online: http://www.w3.org/XML/.