The GridPACK™ Toolkit for Developing Power Grid Simulations on High Performance Computing Platforms

Bruce Palmer, Bill Perkins, Kevin Glass, Yousu Chen, Shuangshuang Jin, Ruisheng Diao, Mark Rice, David Callahan, Steve Elbert, Henry Huang
Objective

Develop a framework to support the rapid development of power grid software applications on HPC platforms

- Extend the penetration of HPC in power grid modeling
- Provide high level abstractions for often-used motifs in power grid applications
- Reduce the amount of explicit communication that must be handled by developers
- Allow power grid application developers to focus on physics and algorithms and not on parallel computing
Power Grid Computing Requirements

- Must be able to manage distributed graphs that represent the power grid network topology
- Support distributed matrices and vectors and implement parallel solvers and preconditioners. Solution algorithms are usually expressed in terms of linear or non-linear algebraic equations
- Map elements from distributed network to matrices and vectors and back to network
GridPACK™ Software Stack

Applications

Network Components
- Neighbor Lists
- Matrix Elements

Application Factory

Solver

GridPACK™ Framework

Import Module
- PTI Formats
- Dictionary

Math Module
- PETSc

Math Module
- PETSc

Configure Module
- XML

Mapper

Network Module
- Ghost Exchanges
- Partitioning

Export Module
- Serial IO
- PTI Formats

Matrix and Vector Module
- PETSc

Utilities
- Errors
- Logging
- Profiling

GridPACK™™ Framework

Matrices and Vectors

Power Grid
Network and Fields

Core
Data
Objects
Approach

Develop software modules that encapsulate commonly used functionality in HPC power grid applications

► Setup and distribution of power grid networks
► Input/Output
► Mapping from grid to distributed matrices
► Parallel solvers
► Incorporate advanced parallel libraries whenever possible
  ▪ PETSc, ParMETIS
GridPACK™ Modules

- Network: Manages the topology, neighbor lists, parallel distribution and indexing. Acts as a container for bus and branch components and manages data exchanges between network components.
- Math: Create distributed matrices and vectors and implement parallel solvers and preconditioners. Math module is a thin wrapper on top of existing solver libraries.
- Network Components: Provide base functionality for describing behavior of buses and branches and define functions needed to set up matrices and vectors.
- Mapper: Create map between distributed network components and distributed matrices and vectors. Enable construction of matrices and vectors directly from network components.
Network Component

- Responsible for managing network topology
- Distributed components over processors
- Set up data exchanges between ghost elements and data located on other processors
Partitioning the Network
Process 0 Partition

Ghost Buses and Branches
Process 1 Partition

Ghost Buses and Branches
Math Module

- Currently built on top of PETSc libraries
- Supports distributed matrices and vectors
  - Matrices can be stored in either sparse or dense formats
- Implements parallel linear and non-linear solvers as well as preconditioners
- Supports basic algebraic operations
  - Matrix-vector multiply
  - Scaling
  - Vector addition
  - Etc.
Network Components

- Most of the application-specific functionality is implemented in the network components.
- Application components are derived from component base classes that define functions that must be implemented in order for other GridPACK™ modules to function.
- Separate application components are defined for branches and buses.
Component Class Hierarchy

MatVecInterface

BaseComponent

BaseBusComponent
BaseBranchComponent

AppBusComponent
AppBranchComponent
The MatVecInterface

- Designed to allow the GridPACK™ framework to generate matrices and vectors from individual bus and branch components.
- Buses and branches are responsible for describing their individual contribution to matrices and vectors.
- Buses and branches are NOT responsible for determining location in matrix or vector and are NOT responsible for distributing matrices or vectors.
Base Network Components

- BaseComponent provides a few methods that are needed by all network components (bus or branch). Sets up buffers used for ghost bus and ghost branch exchanges and defines methods for initializing components and changing component behavior.
- BaseBusComponent provides access to branches attached to bus and keeps track of reference bus.
- BaseBranchComponent provides access to buses at either end of branch.
Typical Component Calculation

\[ Y_{\text{branch}}_{mn} = \sum_k \frac{-1}{r_{mnk} + jx_{mnk}} \]

Ignore values of indices \( m \) and \( n \). These are evaluated by mapper.

\[ Y_{\text{bus}}_{mm} = S_{\text{bus}}_{mm} - \sum_{n \neq m} Y_{\text{branch}}_{mn} \]
Provide a flexible framework for constructing matrices and vectors representing power grid equations

Hide the index transformations and partitioning required to create distributed matrices and vectors from application developers

Developers can focus on evaluating contributions to matrices and vectors coming from individual network elements

- These are local calculations that only depend on neighboring network elements
Matrix Contributions from Network Components

No matrix contribution

No matrix contribution

No matrix contribution

Proudly Operated by Battelle Since 1965
Matrix Generation

Initial Placement

Final Matrix
typedef BaseNetwork<PFBus, PFBbranch> PFNetwork;

Communicator world;

shared_ptr<PFNetwork> network(new PFNetwork(world));

PTI23_parser<PFNetwork> parser(network);
parser.parse("network.raw");
network->partition();

PFFactory factory(network);
factory.load();
factory.setComponents();
factory.setExchange();

network->initBusUpdate();
factory.setYBus();
factory.setMode(YBus);

FullMatrixMap<PFNetwork> mmap(network);
shared_ptr<Matrix> Y = mmap.mapToMatrix();

factory.setSBus();
factory.setMode(RHS);

BusVectorMap<PFNetwork> vmap(network);
shared_ptr<Vector> PQ = vmap.mapToVector();
Powerflow Application using GridPACK™

```cpp
26  factory.setMode(Jacobian);
27  FullMatrixMap<PFNetwork> jMap(network);
28  shared_ptr<Matrix> J = jMap.mapToMatrix();
29  shared_ptr<Vector> X(PQ->clone());
30
31  double tolerance = 1.0e-6;
32  int max_iteration = 100;
33  ComplexType tol = 2.0*tolerance;
34  LinearSolver isolver(*J);
35
36  int iter = 0;
37
38  // Solve matrix equation J*X = PQ
39  isolver.solve(*PQ, *X);
40  tol = X->norm2();
41
42  while (real(tol) > tolerance && iter < max_iteration) {
43     factory.setMode(RHS);
44     vMap.mapToBus(X);
45     network->updateBuses();
46     factory.setMode(RHS);
47     vMap.mapToVector(PQ);
48     factory.setMode(Jacobian);
49     jMap.mapToMatrix(J);
50     LinearSolver solver(*J);
51     solver.solve(*PQ, *X);
52     tol = X->norm2();
53     iter++;
54  }
```

Newton-Raphson loop

- Create Jacobian and solution vector
- Create linear solver and evaluate initial solution
- Project solution back onto network
- Exchange data between network components
Summary

- Major modules that are currently available
  - Network and partitioner
  - Import module for PTI v23 format files
  - Math module for distributed matrices, vectors and solvers
  - Mapper for generating matrices and vectors from network components
  - Export of data from network to standard out
  - Base component and factory classes that support application development
  - Configuration module for managing input from external user files
  - Timer module for profiling

- Power flow application completed
- Download from https://gridpack.org
Acknowledgments

- U.S. Department of Energy’s Office of Electricity through its Advanced Grid Modeling Program.

- Future Power Grid Initiative Lab Directed Research and Development Program at Pacific Northwest National Laboratory