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

Pacific Northwest
NATIONAL LABORATORY

Proudly Operated by Battelle *Since 1965*

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
- Mapper
- Export Module
  - Serial IO
  - PTI Formats
- Matrix and Vector Module
  - PETSc
- Configure Module
  - XML
- Network Module
  - Ghost Exchanges
  - Partitioning
- Utilities
  - Errors
  - Logging
  - Profiling

Core Data Objects

- Matrices and Vectors
- Power Grid Network and Fields

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

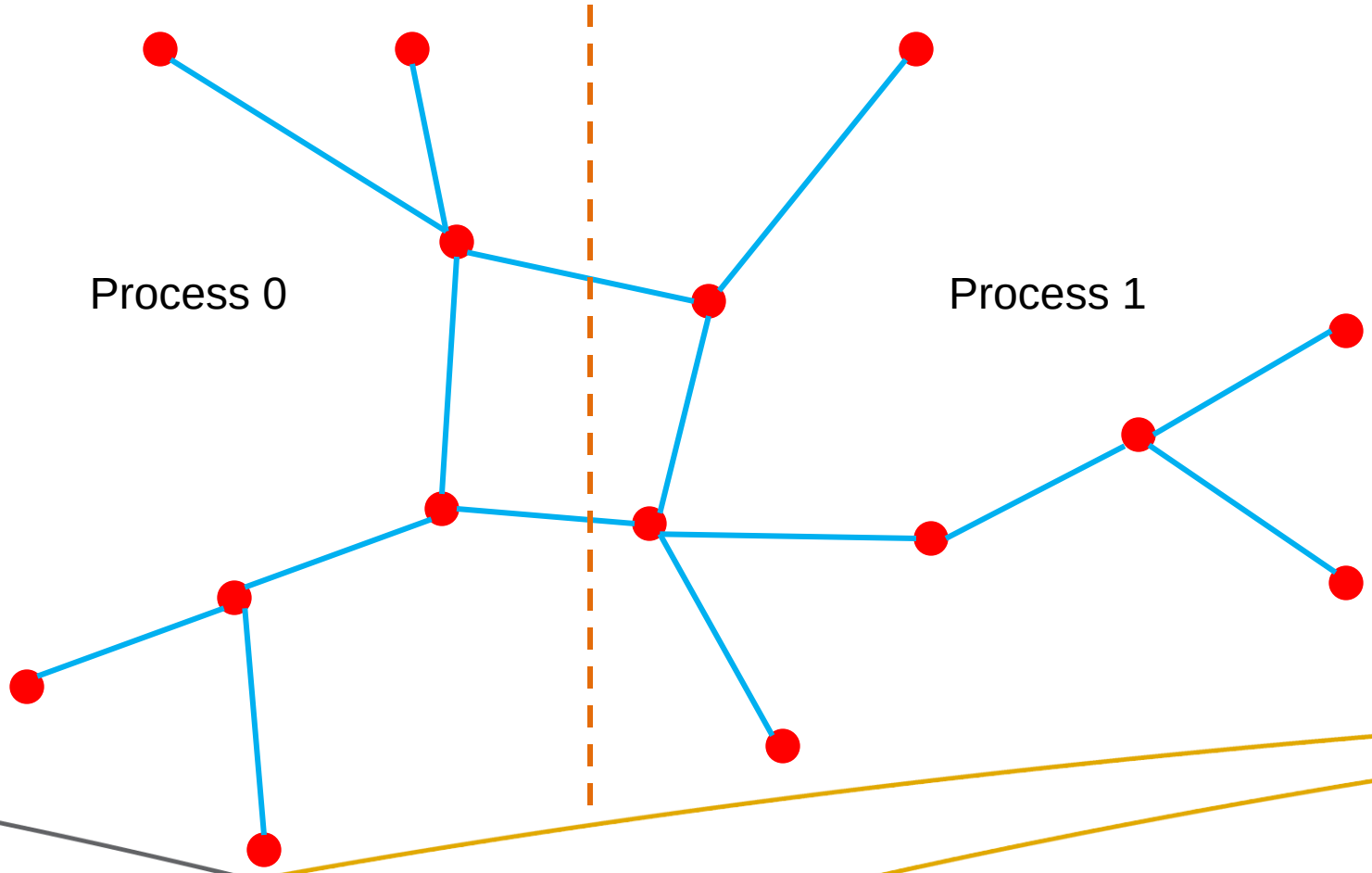
GridPACK™ Modules

- 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

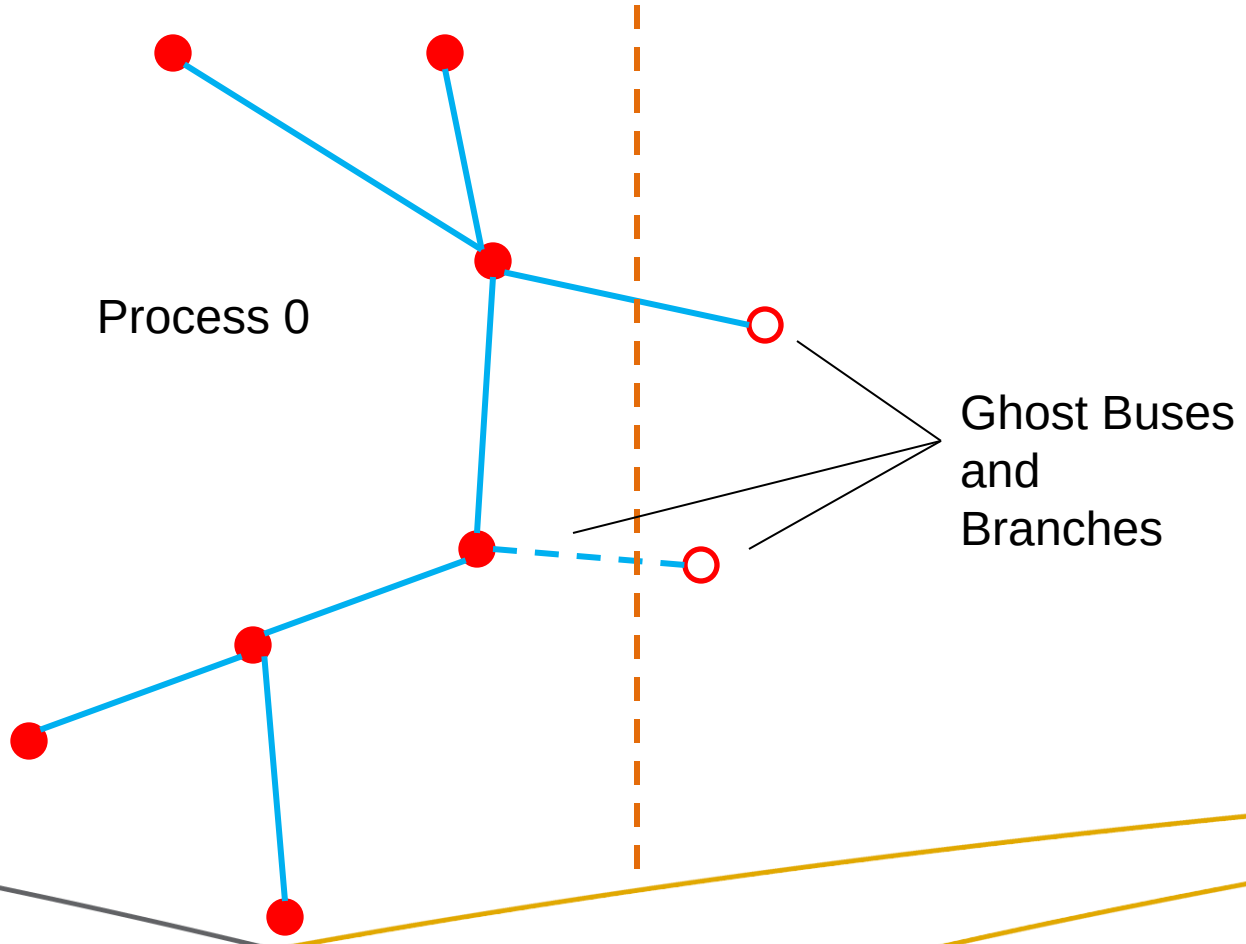
Network Component

- Responsible for managing network topology
- Distributed components over processors
- Set up data exchanges between ghost elements and data located on other processors

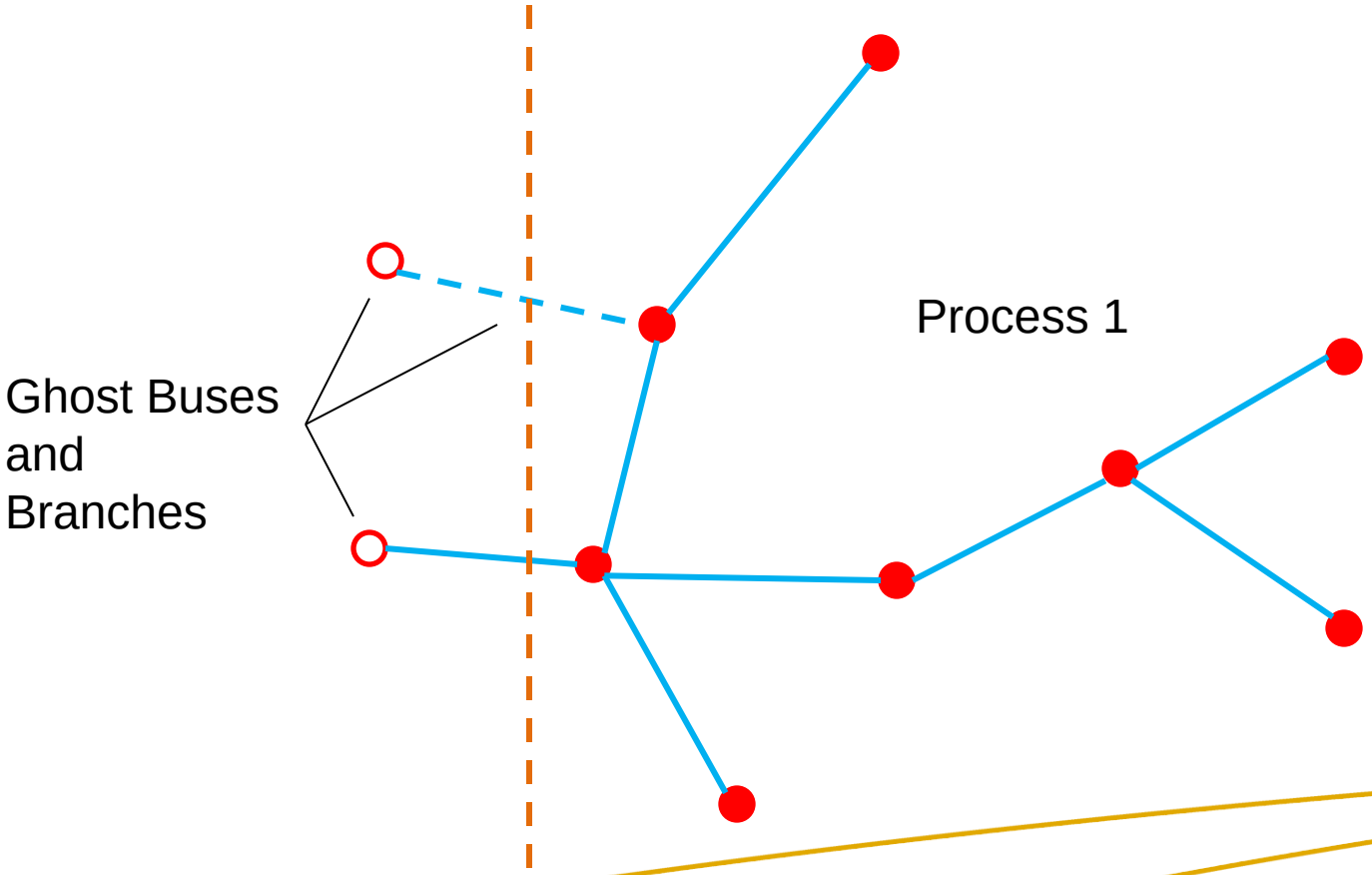
Partitioning the Network



Process 0 Partition



Process 1 Partition



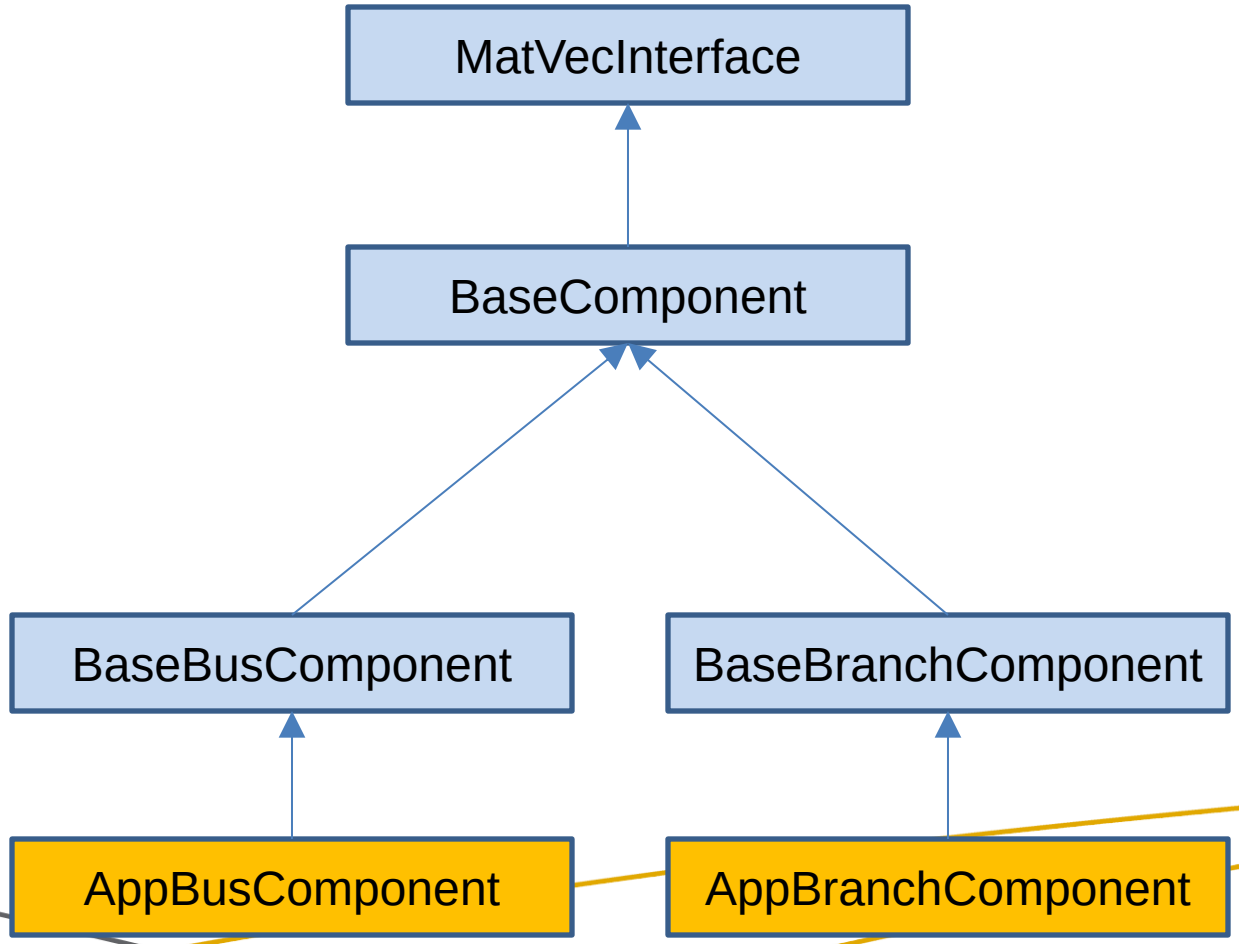
Math Module

- 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



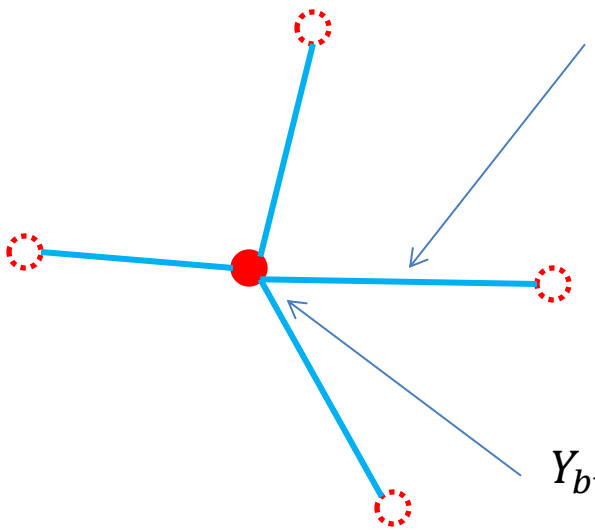
The MatVecInterface

- 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



$$Y_{branch_{mn}} = \sum_{k} \frac{-1}{r_{mnk} + jx_{mnk}}$$

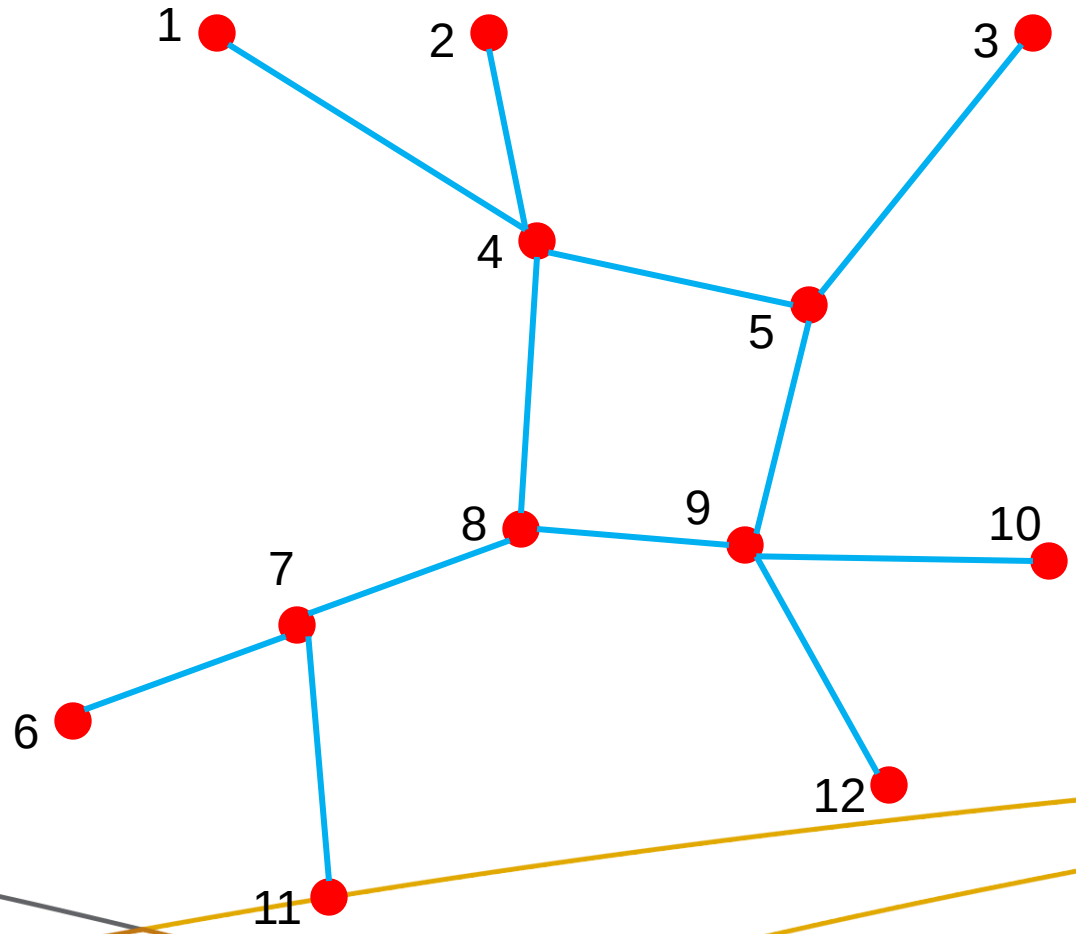
$$Y_{bus_{mm}} = S_{bus_{mm}} - \sum_{n \neq m} Y_{branch_{mn}}$$

Ignore values of indices *m* and *n*. These are evaluated by mapper

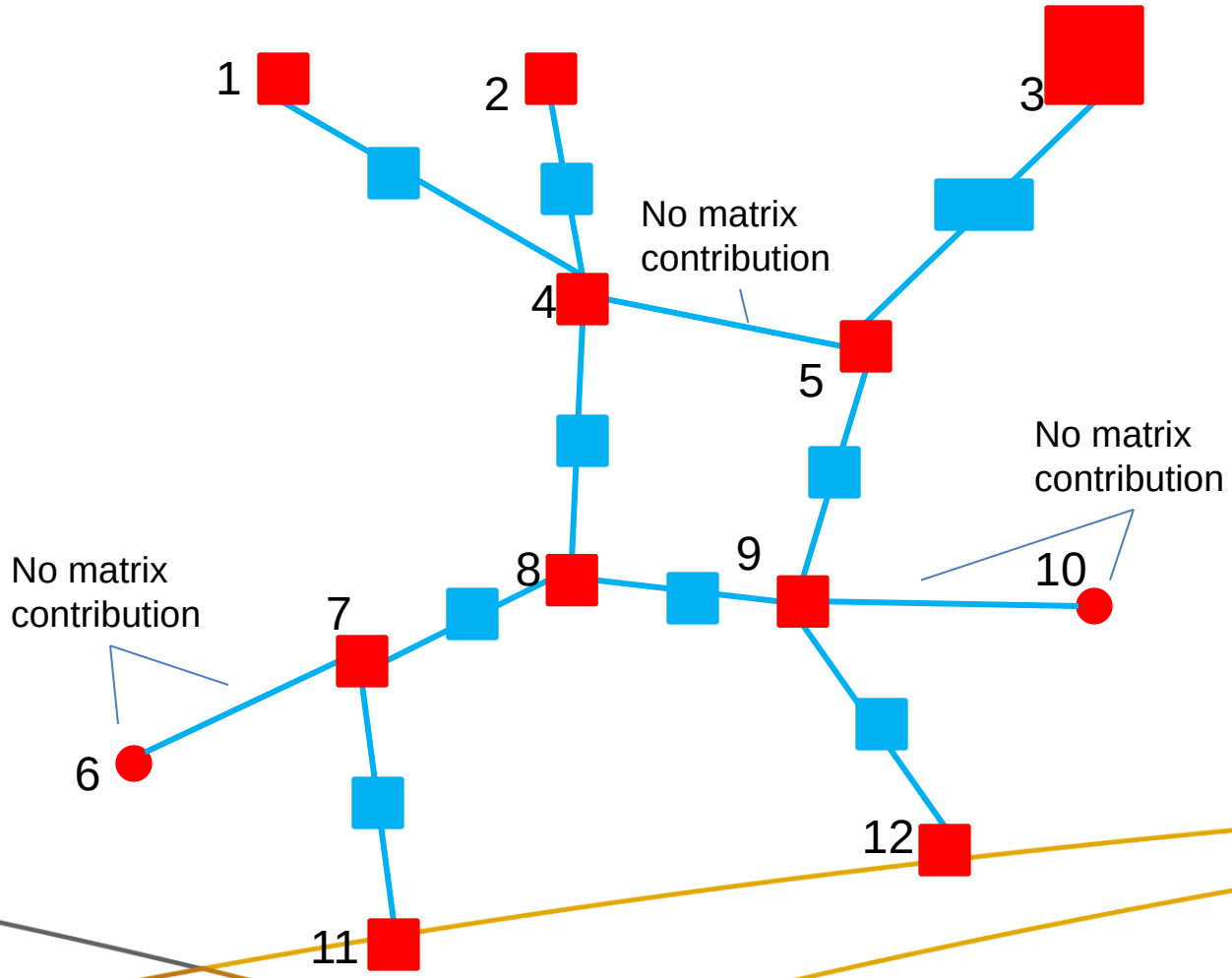
Mapper

- 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

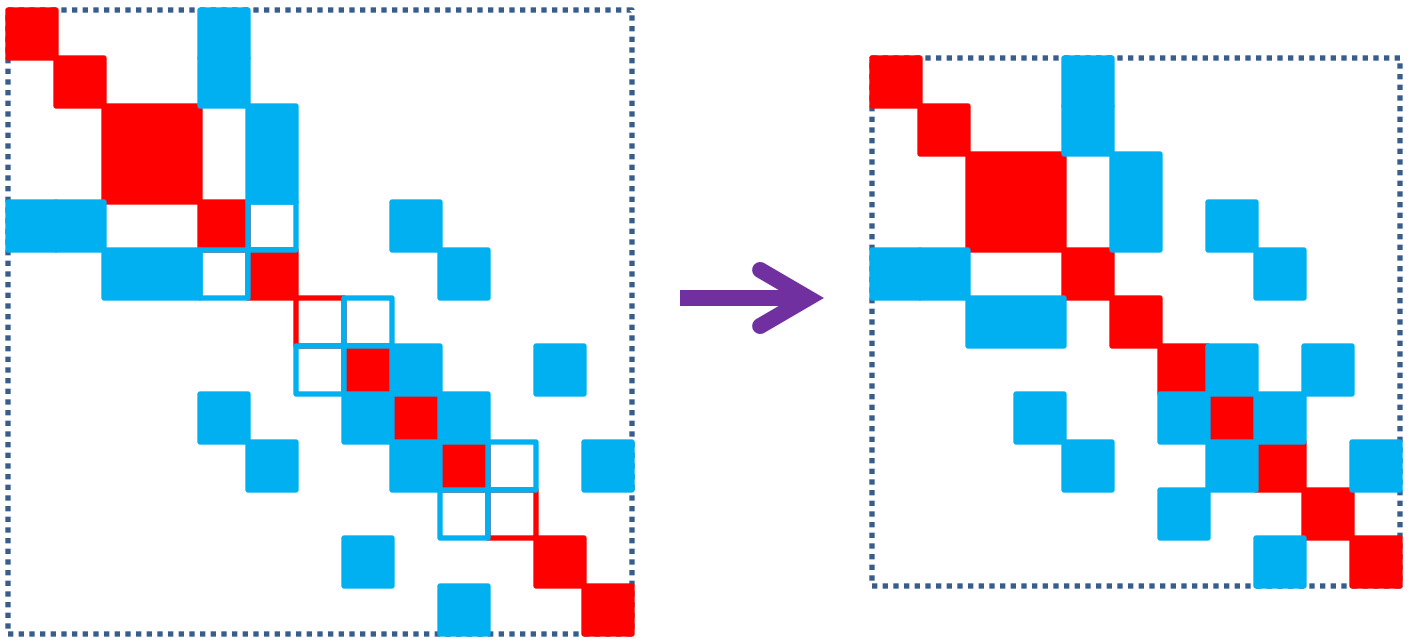
Network Fragment



Matrix Contributions from Network Components



Matrix Generation



Initial Placement

Final Matrix

Powerflow Application using GridPACK™

```
typdef BaseNetwork<PFBus,PFBranch> 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();
```

Create network object

Read in and partition network from external file

Initialize network components and set up ghost exchanges

Create Y-matrix

Create RHS vector for power flow equations

Powerflow Application using GridPACK™

```
factory.setMode(Jacobian);
FullMatrixMap<PFNetwork> jMap(network);
shared_ptr<Matrix> J = jMap.mapToMatrix();
shared_ptr<Vector> X(PQ->clone());

double tolerance = 1.0e-6;
int max_iteration = 100;
ComplexType tol = 2.0*tolerance;
LinearSolver isolver(*J);

int iter = 0;

// Solve matrix equation J*X = PQ
isolver.solve(*PQ, *X);
tol = X->norm2();

while (real(tol) > tolerance &&
iter < max_iteration) {
  factory.setMode(RHS);
  vMap.mapToBus(X);
  network->updateBuses();
  factory.setMode(RHS);
  vMap.mapToVector(PQ);
  factory.setMode(Jacobian);
  jMap.mapToMatrix(J);
  LinearSolver solver(*J);
  solver.solve(*PQ, *X);
  tol = X->norm2();
  iter++;
}
```

Create Jacobian and solution vector

Create linear solver and evaluate initial solution

Project solution back onto network

Exchange data between network components

Newton-Raphson loop

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