GridPACK™ Framework for Developing Power Grid Applications on High Performance Computing Architectures

Bruce Palmer
Pacific Northwest National Laboratory
Collaborator: Argonne National Laboratory
June 17, 2014

Advancement is needed in grid modeling

- The power grid, despite it size and complexity, is still being modeled primarily using workstations
- Serial codes are limited by memory and processor speed, and this limits the size and complexity of existing models
- Modeling large systems using small computers involves substantial aggregation and approximations
- Parallel computing can potentially increase memory and computing power by orders of magnitude, thereby increasing the size and complexity of power grid models that can be simulated using computing
- Parallel computing is more complex than writing serial code and the investment costs are relatively high
- Parallel software is a rapidly changing field and keeping up with new developments can be both expensive and time consuming

Outline

- Objectives
- Impact
- GridPACK™ Framework
- Performance Results

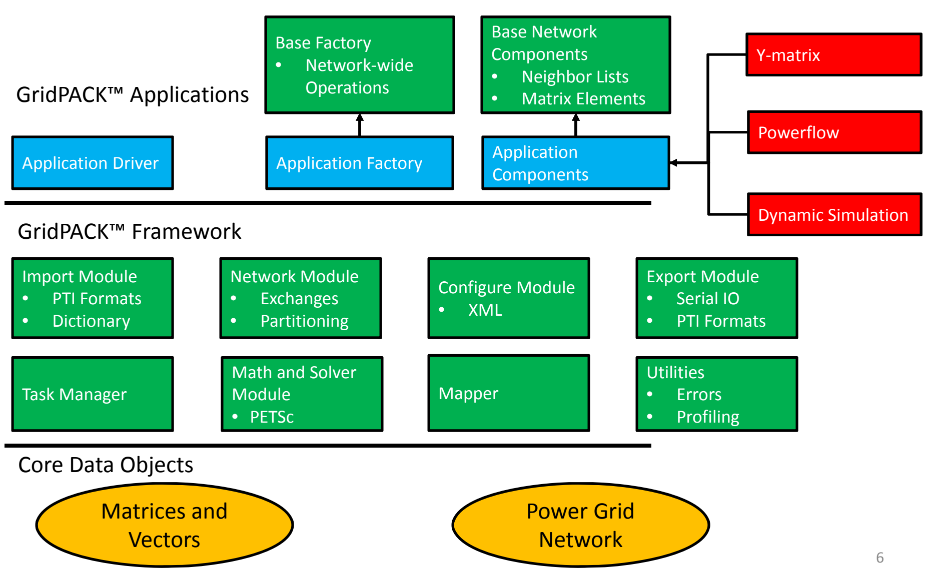
Objectives

- Simplify development of HPC codes for simulating power grid
- Create high level abstractions for common programming motifs in power grid applications
- Incapsulate high performance math libraries and make these available for power grid simulations
- Promote reuse of power grid software components in multiple applications to reduce development and maintenance costs
- Incorporate as much communication and indexing calculations as possible into high level abstractions to reduce application development complexity
- Compartmentalize functionality to reduce maintenance and development costs

Impact

- Access to larger computers with more memory and processing power
- Models containing larger networks and higher levels of detail can be simulated
- Reduced time to solution
- Greater capacity for modeling contingencies and quantifying uncertainty

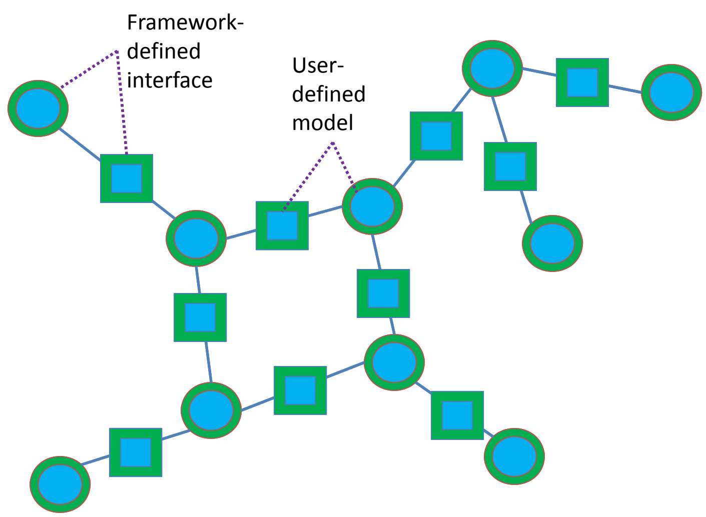
GridPACK™ framework



GridPACK™ network module

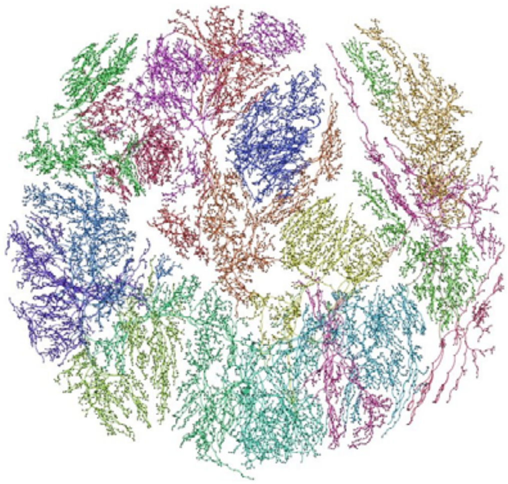
- Network: Manages the topology and partitioning of the network
  - Provides a framework for inserting application specific bus and branch models
  - Keeps track of neighbor relationships
  - Partitions network between processors (the partitioner is built on top of the Parmetis library)
  - Manages ghost buses and branches representing parts of the network on other processors and implements data exchanges between processors

Network component templates to describe arbitrary power grid models



Partition of the WECC network

WECC (Western Electricity Coordinating Council) network partitioned between 16 processors

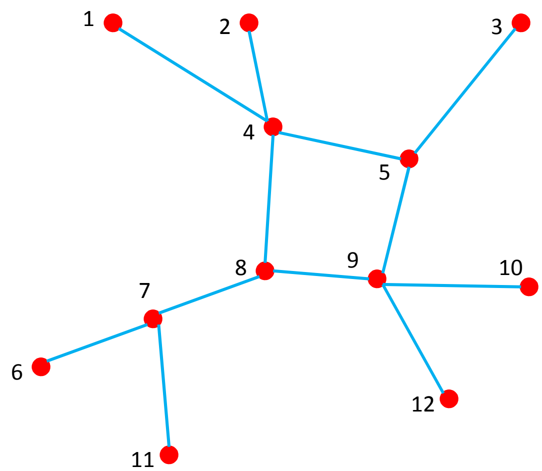


Mapper module

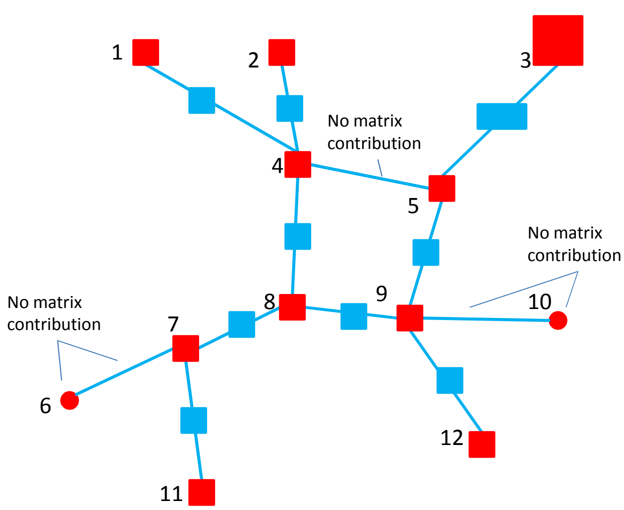
- Generic tool for generating vectors and matrices from network components (buses and branches)
  - Network components are responsible for evaluating their local contributions to vector or matrix
  - Calculations are usually simple and involve only elements that are immediately connected to the contributing component
  - Mapper is responsible for identifying the global location of the contribution in the matrix

$Y_{ii} = -\sum_j Y_{ij}$ Sum over branches to get bus entry

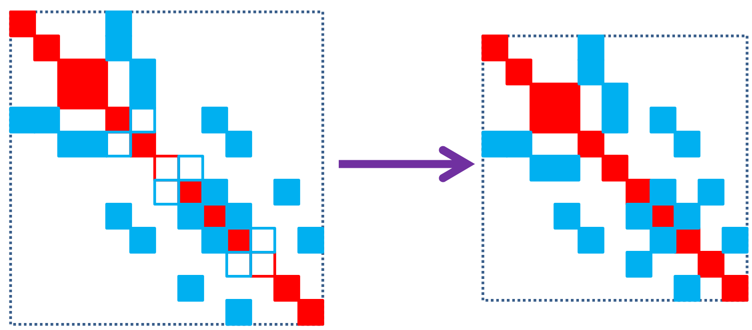
Mapper module



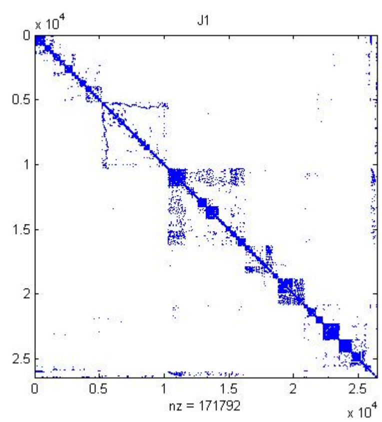
Matrix contributions from network components



Distribute component contributions and eliminate gaps

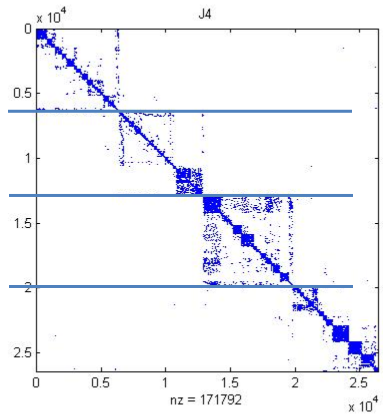


Powerflow Jacobian from mapper (1 processor)



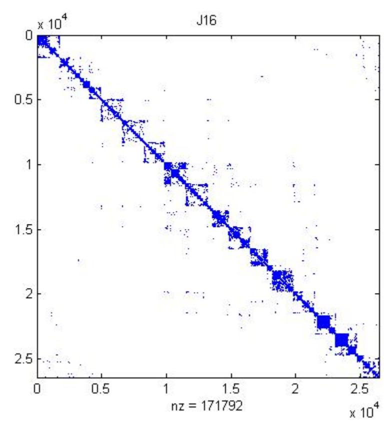
16351 bus
WECC
system

Powerflow Jacobian from mapper (4 processors)



16351 bus
WECC
system

Powerflow Jacobian from mapper (16 processors)



16351 bus
WECC
system

Other modules

- Math: Provide high level abstractions for distributed matrices and vectors
  - Built on top of PETSc math libraries
  - Creation of distributed matrices and vectors
  - Access to the complete suite of linear and non-linear solvers in PETSc
  - Supports basic algebraic operations (matrix-vector multiply, transpose, etc.)
- Factory
  - Implements operations that run over entire network
  - Initializations, function evaluations on all buses and branches, etc.
- Parser: ingest network from external configuration file
  - Currently supports PTI v23 format, other formats are being developed
- Output
  - Organizes output from buses and branches in a consistent manner

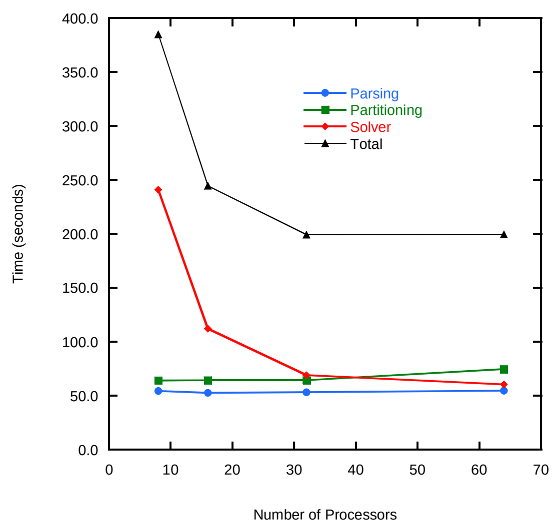
Applications

- Powerflow
  - Demonstrates basic functionality of GridPACK™, including networks, mappers and solvers
  - Hand-coded Newton-Raphson loop and non-linear solver implementation
- Dynamic simulation
  - Dense matrices
  - Algebraic manipulations
  - Local modifications of matrices
- Static Contingency Analysis
  - Managing multiple independent tasks
- Dynamic Contingency Analysis
  - Multiple levels of parallelism

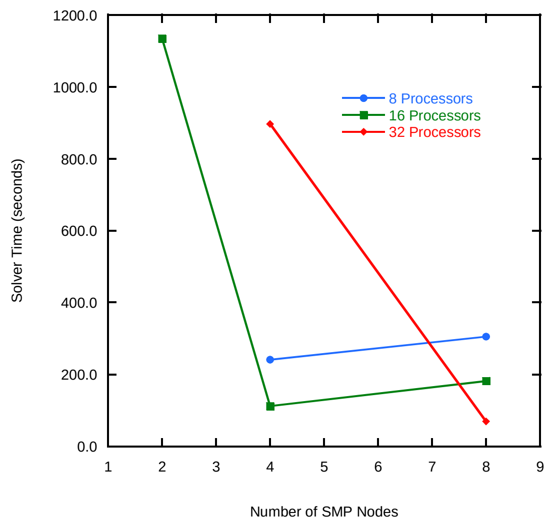
Performance results

- Applications
  - Powerflow
  - Dynamic Simulation
  - Dynamic Contingency Analysis
- Strong scaling performance
  - Fixed size problem, increasing number of processors

Powerflow scaling for artificial 777646 bus network

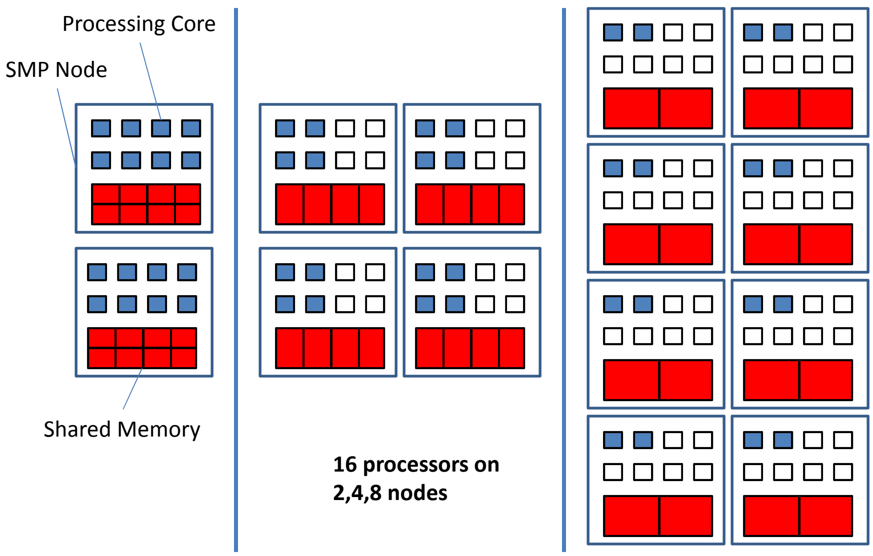


Shared memory effects



Solver performance for powerflow calculation on artificial 777646 bus network

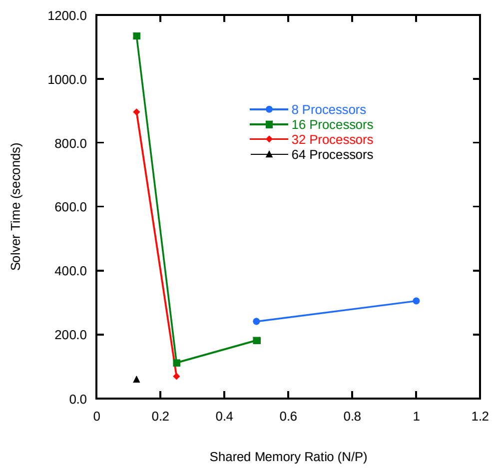
Processor configurations and shared memory



16 processors on 2,4,8 nodes

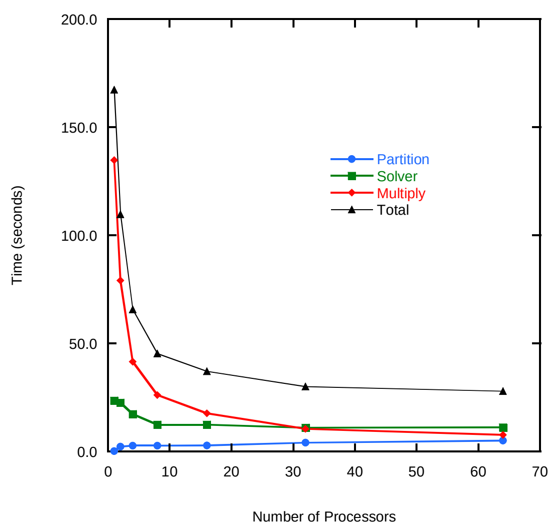
Shared memory effects

Less memory per processor



More internode communication over network

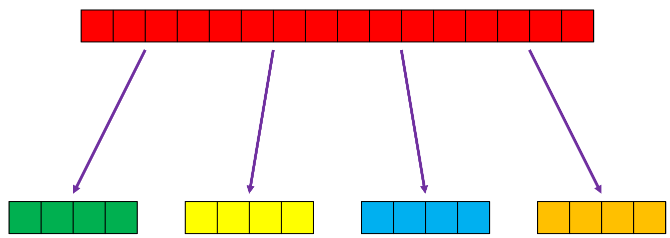
Dynamic simulation



Simulation of 16351 bus WECC network

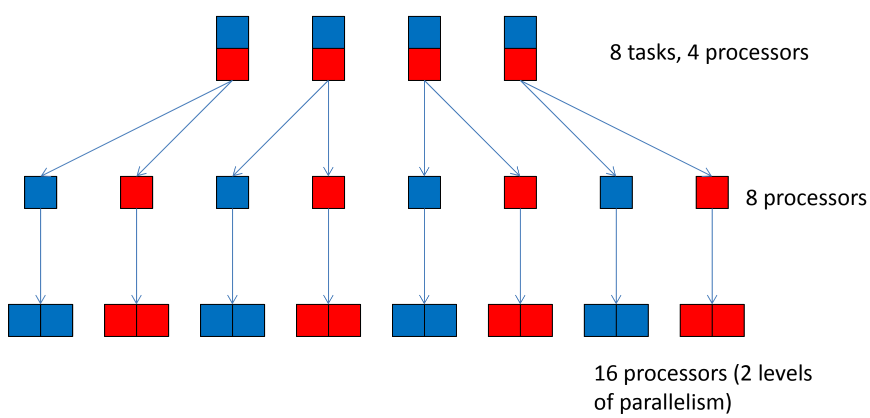
Subtasks on processor groups

World Group

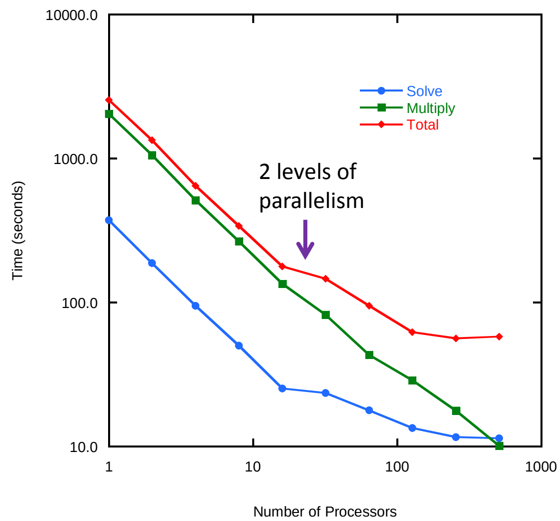


Parallel tasks running on subgroups

Multiple levels of parallelism



Dynamic contingency analysis



Simulation of 16 contingencies on 16351 bus WECC network

Development team

- Bruce Palmer (PI): Parallel code development
- William Perkins: Parallel code development
- Yousu Chen: Power grid application development
- Shuangshuang Jin: Power grid application development
- David Callahan: Data integration
- Kevin Glass: Data integration and optimization
- Ruisheng Diao: Power grid engineering and model validation
- Stephen Elbert: Optimization and economic modeling
- Mallikarjuna Vallem: Synthetic data and model validation
- Nathan Tenney: Automatic builds and testing
- Zhenyu (Henry) Huang: Program management

Conclusion

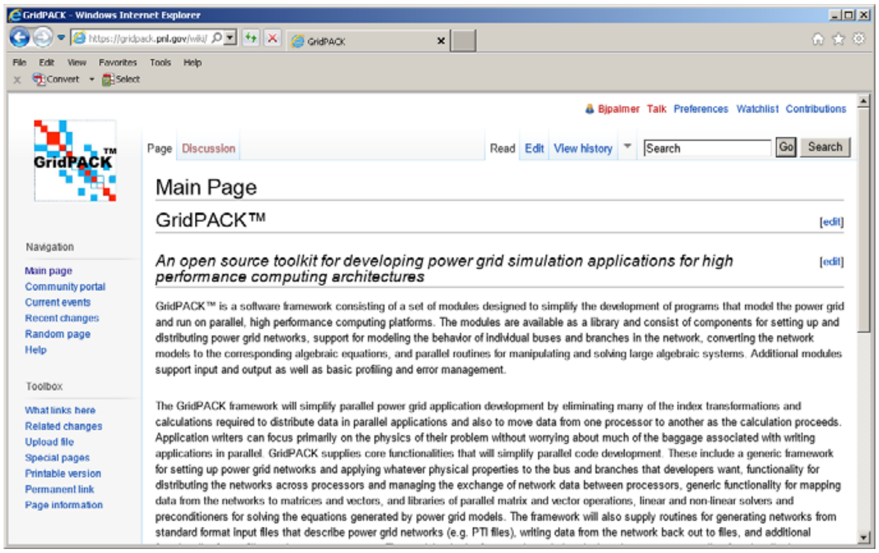
- GridPACK™ has been successfully deployed and is available to the public at https://gridpack.org, along with documentation
- Multiple demonstration programs of power grid applications have been developed with GridPACK™ and have shown scaling behavior. These include
  - Powerflow
  - Dynamic Simulation
  - Static Contingency Analysis
  - Dynamic Contingency Analysis
- Development of state estimation calculation and Fortran interface is underway
- Mini-tutorial at 3rd at Workshop on Next-Generation Analytics for the Future Power Grid, PNNL, Richland, WA, July 17-18, 2014

Contacts

Bruce Palmer (PI): bruce.palmer@pnnl.gov

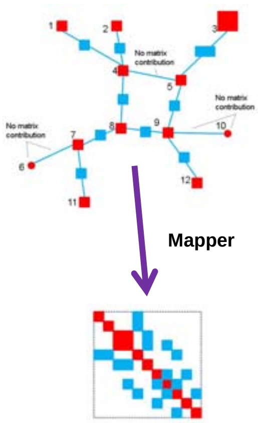
GridPACK™ webpage: https://gridpack.org

Webpage



GridPACK™ Toolkit

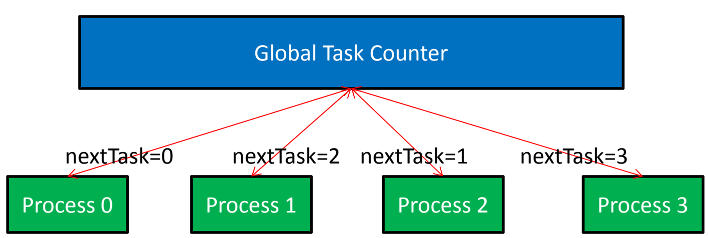
- **Research Objective:**
  - GridPACK™ is a collection of modules designed to simplify the development of power grid programs that can run on today's high performance computing architectures
- **Research Highlights**
  - Base modules for the GridPACK™ toolkit have been developed and include
    - Network module for creating and distributing power grid network models over multiple processors
    - Network component interfaces that define how bus and branch models interact with the rest of the GridPACK™ framework
    - Math modules that support the creation of distributed matrices and vectors and implement linear and non-linear solvers
    - Mapper modules that support the creation of matrices and vectors from network models.
  - Completed development of a parallel power flow, dynamic simulation and contingency analysis applications using the GridPACK™ functionality and demonstrated parallel speedup for all applications
  - GridPACK™ software is available on website https://gridpack.org, along with substantial documentation



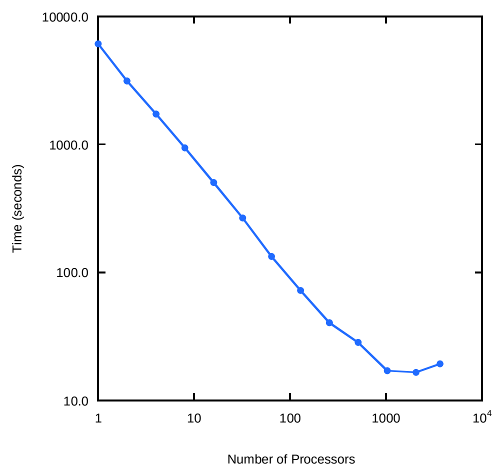
Powerflow code

```
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();
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++;
}
```


Task Manager

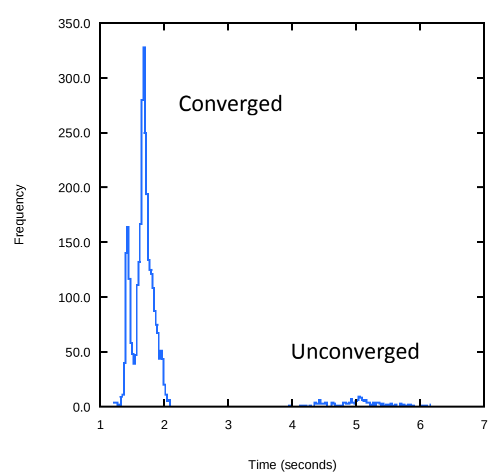


Static contingency analysis



Static Contingency Analysis of WECC system using 1 processor per contingency for 3638 contingencies

Distribution of times in solver loop



Static Contingency Analysis of WECC system using 1 processor per contingency for 3638 contingencies