

# Using PETSc as a Testbed for Developing and Benchmarking Parallel Power System Applications

Shrirang Abhyankar, Barry Smith, Hong Zhang  
Argonne National Laboratory

Alexander Flueck  
Illinois Institute of Technology

October 24, 2012

Algorithms ,computing resources, and software tools need to be upto date with the market operational demands.

Algorithms ,computing resources, and software tools need to be upto date with the market operational demands.

# Need for parallelizing power system algorithms

- Next-generation power grid
  - PMUs, smart meters, Distributed Generation, Plug-in hybrid vehicles, Smart and Micro-grids, Power electronics, Increased communication
- Resultant Computational challenges
  - Data explosion, Real-time simulation requirements, Larger or denser network, Multi-scale (temporal, geographical)
- Next-generation computing architecture
  - Multicore, Manycore machines.
  - GPGPUs

# Parallel computing in Power Systems

- Research on parallel power system applications
  - Power Flow, Optimal Power Flow, Transient Stability, Contingency Analysis, State Estimation, Electromagnetic transients simulation.
- Survey papers
  - “Parallel processing in power systems computation” (IEEE Task Force)
  - “High Performance Computing in Power Systems” (D. Falcao)
- Many parallel algorithms showed significant time savings yet hesitancy in commercial adoption
  - Specific algorithms tested for on a specific architecture using a specific topology and operating conditions.
  - Parallel algorithms are hard to program: synchronization/communication, partitioning, parallel linear, nonlinear solvers, reductions, debugging,...

# Parallel computing in Power Systems

- Research on parallel power system applications
  - Power Flow, Optimal Power Flow, Transient Stability, Contingency Analysis, State Estimation, Electromagnetic transients simulation.
- Survey papers
  - “Parallel processing in power systems computation” (IEEE Task Force)
  - “High Performance Computing in Power Systems” (D. Falcao)
- Many parallel algorithms showed significant time savings yet hesitancy in commercial adoption
  - Specific algorithms tested for on a specific architecture using a specific topology and operating conditions.
  - Parallel algorithms are hard to program: synchronization/communication, partitioning, parallel linear, nonlinear solvers, reductions, debugging,...

# Parallel computing in Power Systems

- Need to develop and benchmark parallel algorithms on **different** computing architectures on **different** topologies under **different** power system operating conditions.
  - Benchmark existing parallel algorithms.
  - Develop and benchmark new ones.

This a lot of work!!!

High performance libraries, such as PETSc, can aid in the rapid development and benchmarking process:

- Rapid development of parallel applications.
- Portable to variety of computing architectures.
- Wide array of tested for linear, nonlinear, and time-stepping solvers.
- Reduce the experimentation time and effort.

# Parallel computing in Power Systems

- Need to develop and benchmark parallel algorithms on **different** computing architectures on **different** topologies under **different** power system operating conditions.
  - Benchmark existing parallel algorithms.
  - Develop and benchmark new ones.

This a lot of work!!!

High performance libraries, such as PETSc, can aid in the rapid development and benchmarking process:

- Rapid development of parallel applications.
- Portable to variety of computing architectures.
- Wide array of tested for linear, nonlinear, and time-stepping solvers.
- Reduce the experimentation time and effort.

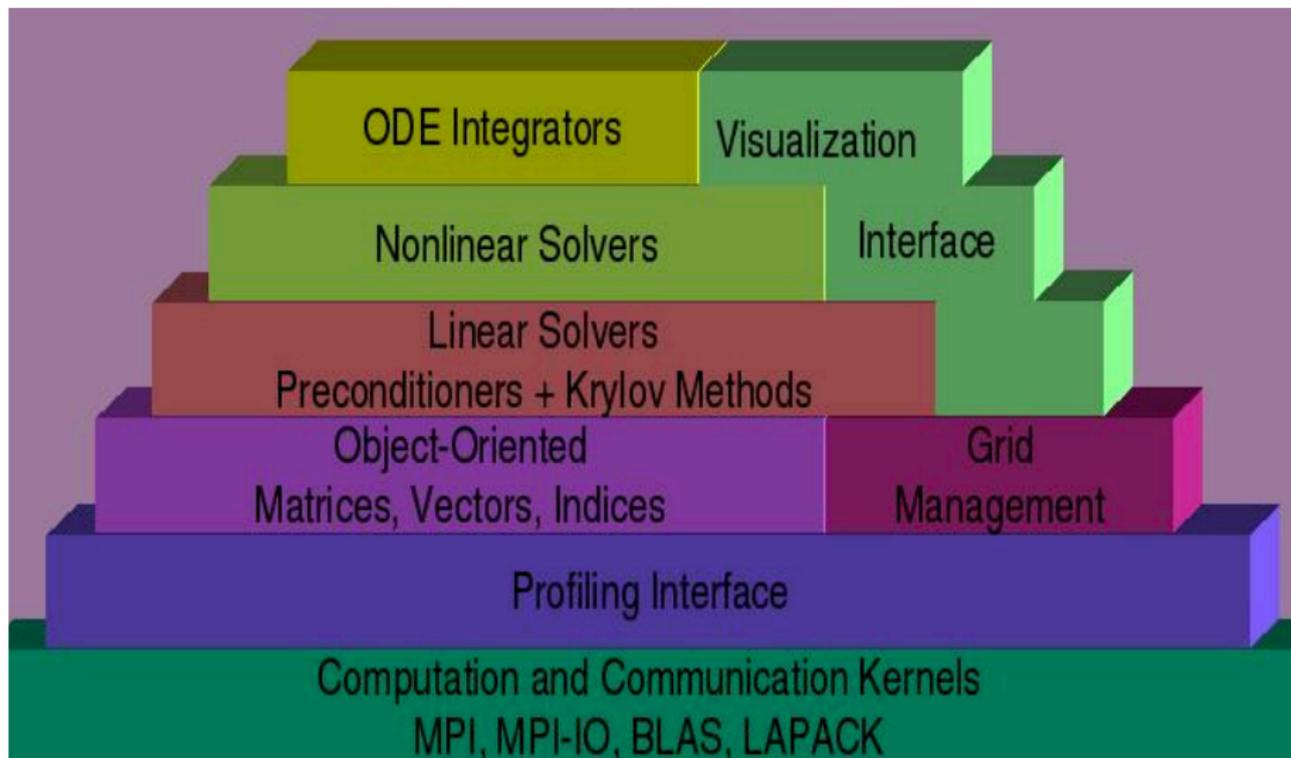
# Portable Extensible Toolkit for Scientific Computation

- What is PETSc?
  - Library for developing large-scale parallel applications.
  - Provides parallel numerical solvers (time-stepping, nonlinear, linear) and basic building blocks (parallel matrices, vectors, communication objects) for rapidly developing parallel applications.
  - Mostly used by researchers in PDE applications.
  - Free for anyone to use including industrial users.
  - Top 100 R & D award in 2009, Cited as DOE's top 10 advancements in computational science accomplishments in 2008.
- What can PETSc handle ?
  - PETSc has run implicit problems with 1 billion unknowns
    - PFLOTRAN for flow in porous media
  - PETSc has run on over 224,000 cores efficiently
    - UNIC on the IBM BG/P at ANL
    - PFLOTRAN on the Cray XT5 Jaguar at ORNL

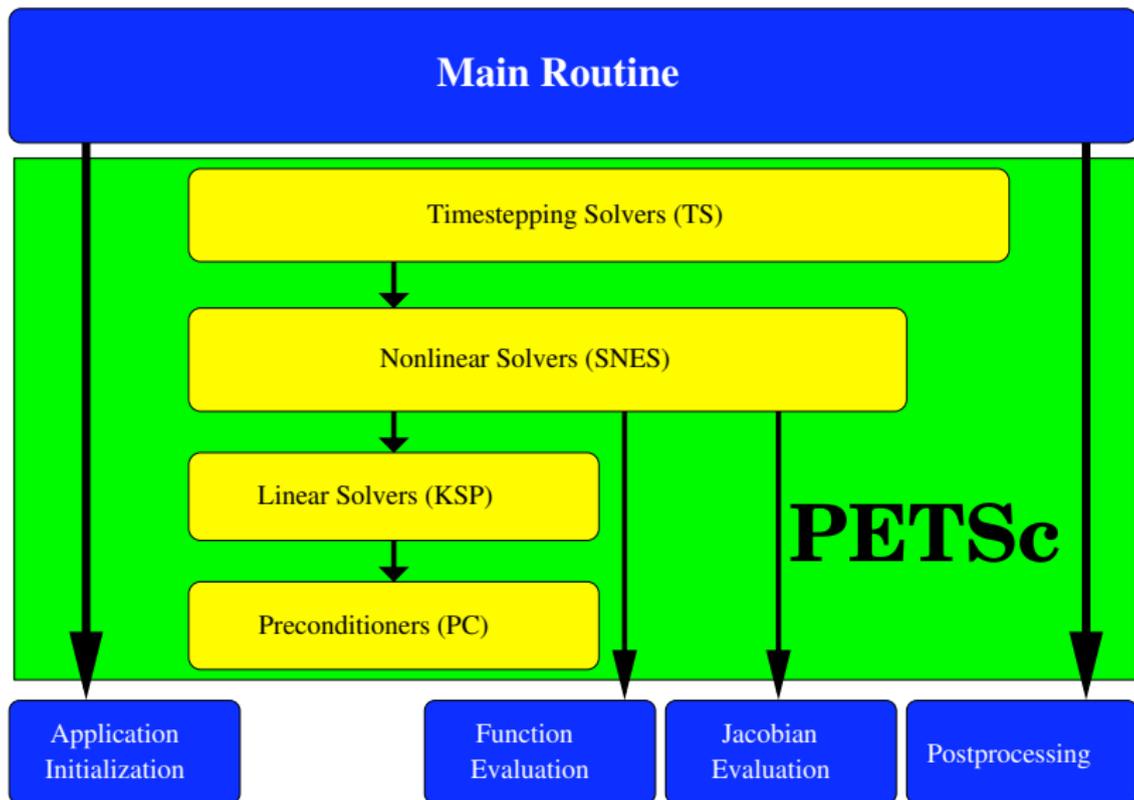
# Portable Extensible Toolkit for Scientific Computation

- What is PETSc?
  - Library for developing large-scale parallel applications.
  - Provides parallel numerical solvers (time-stepping, nonlinear, linear) and basic building blocks (parallel matrices, vectors, communication objects) for rapidly developing parallel applications.
  - Mostly used by researchers in PDE applications.
  - Free for anyone to use including industrial users.
  - Top 100 R & D award in 2009, Cited as DOE's top 10 advancements in computational science accomplishments in 2008.
- What can PETSc handle ?
  - PETSc has run implicit problems with **1 billion** unknowns
    - PFLOTRAN for flow in porous media
  - PETSc has run on over **224, 000** cores efficiently
    - UNIC on the IBM BG/P at ANL
    - PFLOTRAN on the Cray XT5 Jaguar at ORNL

# PETSc Organization

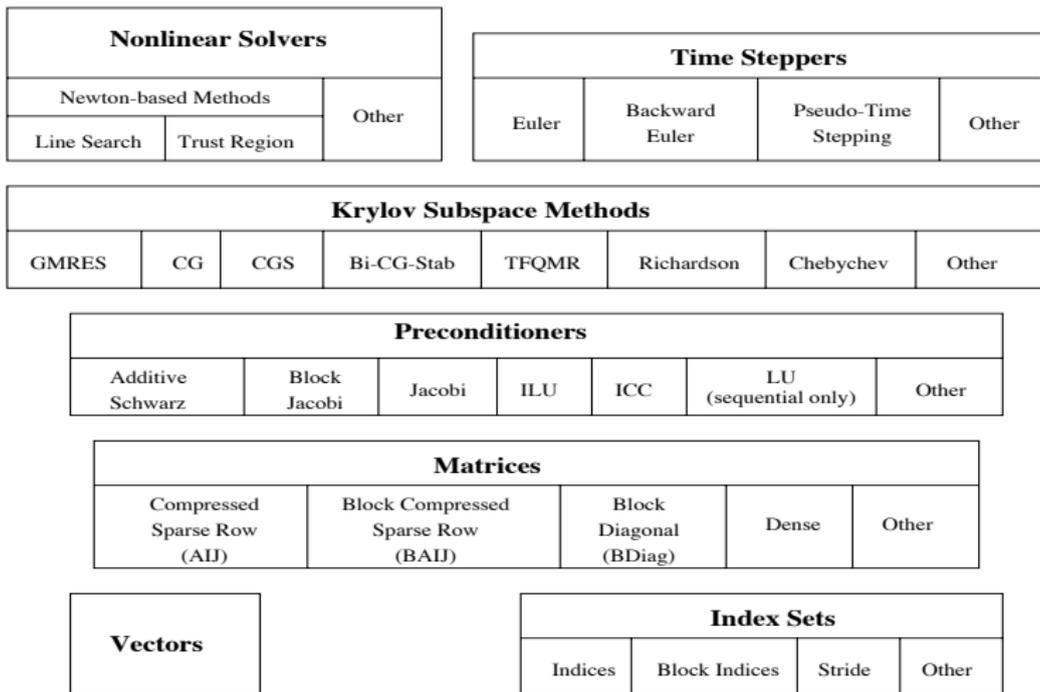


# Flow control of a PETSc application



# PETSc Numerical Components

## Parallel Numerical Components of PETSc



# PETSc features

- Free for anyone, including industrial users
- Portability
  - Unix, Linux, MacOS, Windows
  - Tightly/loosely couple architectures
  - 32/64 bit ints, single/double/quad precision, real/complex
  - C, C++, Fortran, Matlab, Python (petsc4py)
- Extensibility
  - BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
  - MPICH, MPE, Open MPI
  - ParMetis, Chaco, Jostle, Party, Scotch
  - MUMPS, Spooles, SuperLU, SuperLU\_Dist, UMFPack, pARMS
  - PaStiX, BLOPEX, FFTW, SPRNG
  - HYPRE, ML, SPAI
  - Sundials
  - HDF5, Boost
  - Packages can be directly downloaded and installed at configure time  
`--download-<packagename>=1`

# PETSc features

- Free for anyone, including industrial users
- Portability
  - Unix, Linux, MacOS, Windows
  - Tightly/loosely couple architectures
  - 32/64 bit ints, single/double/quad precision, real/complex
  - C, C++, Fortran, Matlab, Python (petsc4py)
- Extensibility
  - BLAS, LAPACK, BLACS, ScaLAPACK, PLAPACK
  - MPICH, MPE, Open MPI
  - ParMetis, Chaco, Jostle, Party, Scotch
  - MUMPS, Spooles, SuperLU, SuperLU\_Dist, UMFPack, pARMS
  - PaStiX, BLOPEX, FFTW, SPRNG
  - HYPRE, ML, SPAI
  - Sundials
  - HDF5, Boost
  - Packages can be directly downloaded and installed at configure time  
--download-<packagename>=1

# PETSc features

- Abstract linear algebra interface (Vectors, Matrices, Index Sets, etc.)
- Consistent user interface.
- Keep MPI opaque to the user.
- Flexible run time options
  - Old  
Modify. Make. Run. Modify. Make. Run
  - New  
`./ex -snes_type <ls,tr,test> -ksp_type  
<gmres,cg,bicg,preonly> -pc_type <lu,ilu,icc,jacobi>  
-mat_type <aij,baij,sbaij>`
- Debugging [gdb, dbx]
  - Automatic generation of trace back
  - Attach debugger at start, on error, to a subset of processes
  - Valgrind
- Check jacobian correctness
  - `-snes_type test -snes_test_display`
- Profiling
  - Logs Time, Memory usage, Calls, Flops, MPI messages, user code (stages, events)

# PETSc features

- Abstract linear algebra interface (Vectors, Matrices, Index Sets, etc.)
- Consistent user interface.
- Keep MPI opaque to the user.
- Flexible run time options
  - Old  
Modify. Make. Run. Modify. Make. Run
  - New  
`./ex -snes_type <ls,tr,test> -ksp_type  
<gmres,cg,bicg,preonly> -pc_type <lu,ilu,icc,jacobi>  
-mat_type <aij,baij,sbaij>`
- Debugging [gdb, dbx]
  - Automatic generation of trace back
  - Attach debugger at start, on error, to a subset of processes
  - Valgrind
- Check jacobian correctness
  - `-snes_type test -snes_test_display`
- Profiling
  - Logs Time, Memory usage, Calls, Flops, MPI messages, user code (stages, events)

# PETSc features

- Abstract linear algebra interface (Vectors, Matrices, Index Sets, etc.)
- Consistent user interface.
- Keep MPI opaque to the user.
- Flexible run time options
  - Old  
Modify. Make. Run. Modify. Make. Run
  - New  

```
./ex -snes_type <ls,tr,test> -ksp_type  
<gmres,cg,bicg,preonly> -pc_type <lu,ilu,icc,jacobi>  
-mat_type <aij,baij,sbaij>
```
- Debugging [gdb, dbx]
  - Automatic generation of trace back
  - Attach debugger at start, on error, to a subset of processes
  - Valgrind
- Check jacobian correctness
  - `-snes_type test -snes_test_display`
- Profiling
  - Logs Time, Memory usage, Calls, Flops, MPI messages, user code (stages, events)

# PETSc features

- Abstract linear algebra interface (Vectors, Matrices, Index Sets, etc.)
- Consistent user interface.
- Keep MPI opaque to the user.
- Flexible run time options
  - Old  
Modify. Make. Run. Modify. Make. Run
  - New  

```
./ex -snes_type <ls,tr,test> -ksp_type  
<gmres,cg,bicg,preonly> -pc_type <lu,ilu,icc,jacobi>  
-mat_type <aij,baij,sbaij>
```
- Debugging [gdb, dbx]
  - Automatic generation of trace back
  - Attach debugger at start, on error, to a subset of processes
  - Valgrind
- Check jacobian correctness
  - `-snes_type test -snes_test_display`
- Profiling
  - Logs Time, Memory usage, Calls, Flops, MPI messages, user code (stages, events)

# Support for GPGPUs and Multicore architectures

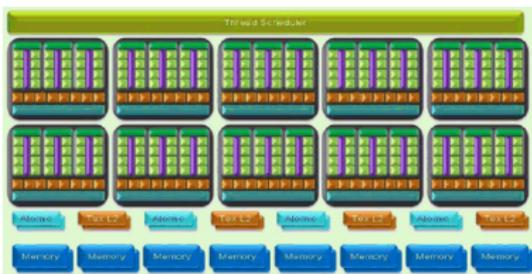


Figure: NVidia GTX 280 GPU architecture

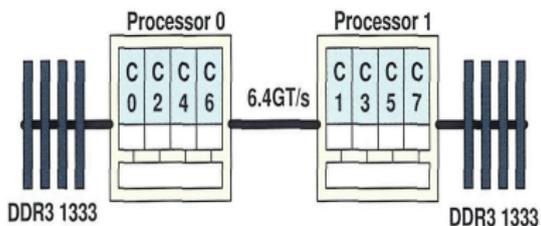


Figure: Intel Nehalem

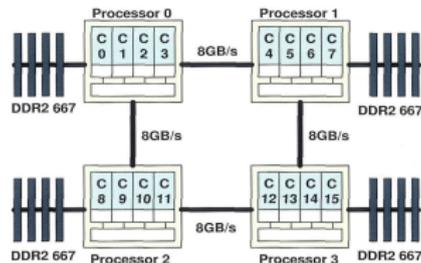


Figure: AMD Barcelona

# Splitting for Multiphysics

Efficient solvers for coupled multiphysics applications

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} f \\ g \end{bmatrix}$$

- Relaxation: `-pc_fieldsplit_type`  
[additive,multiplicative,symmetric\_multiplicative]

$$\begin{bmatrix} A & \\ & D \end{bmatrix}^{-1} \quad \begin{bmatrix} A & \\ C & D \end{bmatrix}^{-1} \quad \begin{bmatrix} A & \\ & \mathbf{1} \end{bmatrix}^{-1} \left( \mathbf{1} - \begin{bmatrix} A & B \\ & \mathbf{1} \end{bmatrix} \begin{bmatrix} A & \\ C & D \end{bmatrix}^{-1} \right)$$

- Gauss-Seidel inspired, works when fields are loosely coupled
- Factorization: `-pc_fieldsplit_type schur`

$$\begin{bmatrix} A & B \\ & S \end{bmatrix}^{-1} \begin{bmatrix} \mathbf{1} & \\ CA^{-1} & \mathbf{1} \end{bmatrix}^{-1}, \quad S = D - CA^{-1}B$$

# List of power system applications that can be developed using PETSc

- Linear (using KSP)
  - DC Power Flow, Sensitivity factors
- Nonlinear (using SNES)
  - AC Power Flow, Contingency analysis, Continuation power flow
  - Distribution power flow, Combined Transmission-distribution power flow
- Time-stepping (using TS)
  - Transient stability, Electromagnetic transients
  - Combined transient stability-electromagnetic transients (hybrid simulation)
- Optimization (using TAO package)
  - SCOPF, LMP calculations
- Eigen-value analysis (using SLEPc package)
  - Small signal stability analysis

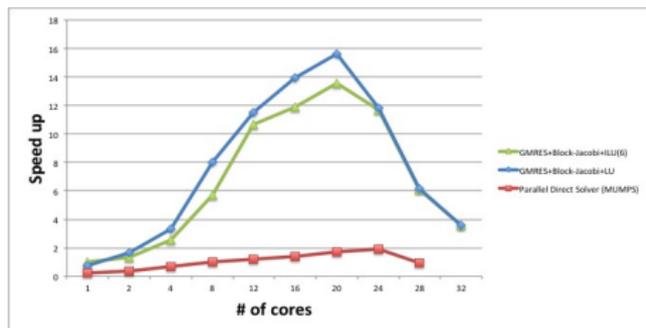
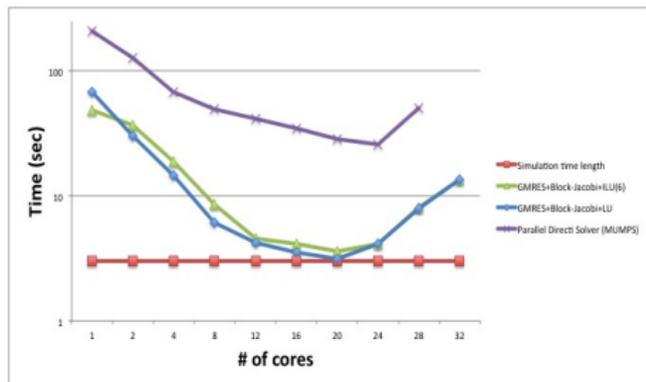
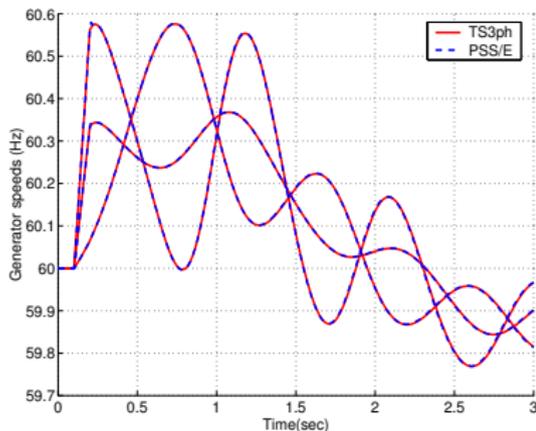
# Real-time electrical power system dynamics

- Nonlinear differential-algebraic power system model

$$\dot{x} = f(x, y)$$

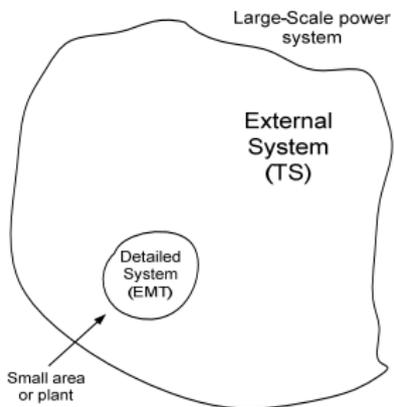
$$0 = g(x, y)$$

- Three-phase network
- Spatial decomposition in parallel



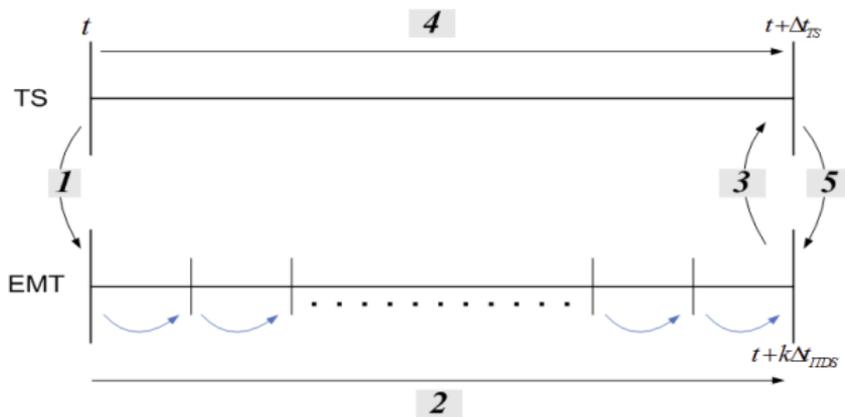
\* Scalability results of 2360 bus, 4670 branches, 1080 generator system

# Combined Electromechanical and Electromagnetic Transients Simulation



- Capture “global” slow dynamics and “local” fast dynamics
- Use TS globally and EMT locally
- Need interface for
  - Time step
  - Network modeling
  - Waveform

## Existing “explicit” hybrid simulation approach



- Make separate TS and EMT programs talk to each other
- Explicit approach
- No iterations between TS and EMT
- Diverges for large changes in voltages/currents
- Limited parallelism

# Proposed “Implicitly-coupled” hybrid simulation approach

- Combine TS and EMT at the equation level rather than at the application level
- Solve TS equations and coupled-in-time EMT equations for each TS time step together
- More robust than the explicit approach
- Allows an integrated parallel implementation

$$x_{TS}(t_{N+1}) - x_{TS}(t_N) - \frac{\Delta t_{TS}}{2}(F(t_{N+1}) + F(t_N)) = 0 \quad (4)$$

$$G(t_{N+1}) = 0 \quad (5)$$

$$x_{EMT}(t_{n+1}) - x_{EMT}(t_n) - \frac{\Delta t_{EMT}}{2}(f_1(t_{n+1}) + f_1(t_n)) = 0 \quad (6)$$

$$i_{bdry}(t_{n+1}) - i_{bdry}(t_n) - \frac{\Delta t_{EMT}}{2}(f_2(t_{n+1}) + f_2(t_n)) = 0 \quad (7)$$

$$x_{EMT}(t_{n+2}) - x_{EMT}(t_{n+1}) - \frac{\Delta t_{EMT}}{2}(f_1(t_{n+2}) + f_1(t_{n+1})) = 0 \quad (8)$$

$$i_{bdry}(t_{n+2}) - i_{bdry}(t_{n+1}) - \frac{\Delta t_{EMT}}{2}(f_2(t_{n+2}) + f_2(t_{n+1})) = 0 \quad (9)$$

$$\vdots$$

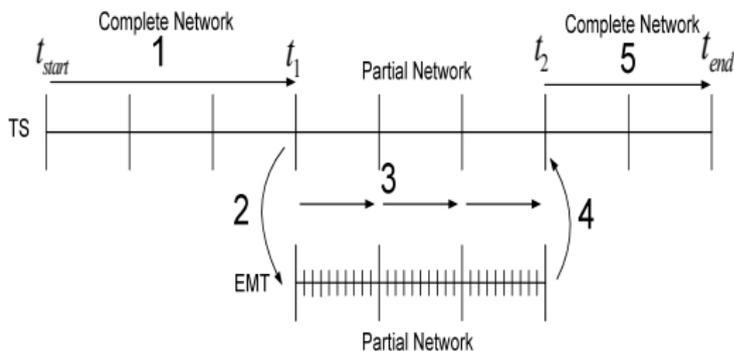
$$\vdots$$

$$x_{EMT}(t_{n+k}) - x_{EMT}(t_{n+k-1}) - \frac{\Delta t_{EMT}}{2}(f_1(t_{n+k}) + f_1(t_{n+k-1})) = 0 \quad (10)$$

$$i_{bdry}(t_{n+k}) - i_{bdry}(t_{n+k-1}) - \frac{\Delta t_{EMT}}{2}(f_2(t_{n+k}) + f_2(t_{n+k-1})) = 0 \quad (11)$$

# Multi-scale dynamics simulation strategy

- Only run the implicitly coupled simulator in the presence of fast dynamics, run TS for all other times

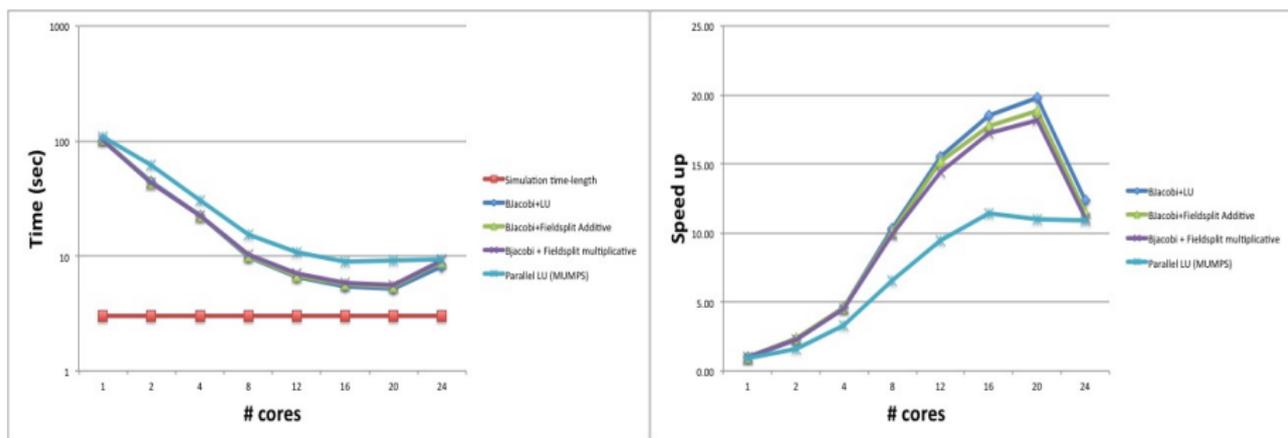


- Time-comparison of different dynamic analyses

System size	Simulated time (sec)	TS3ph	EMT	Only TSEMT	TS3ph-TSEMT
9 bus	3	0.13	4.96	5.46	0.41
118 bus	3	0.36	30.1	4.87	0.53

# Parallel implementation and performance results

- Partition TS network in space and EMT network in time
- Each processor gets equations for
  - TS subnetwork
  - EMT equations for multiple time-steps



\*2360 buses total, 4 buses, 3 transmission lines and 4 loads in EMT network

\*Using GMRES + Block-Jacobi + LU + Very Dishonest preconditioning

# PETSc use in the example applications

- **Easy parallel implementation**
- Partitioning (using ParMetis)
- Tune Linear solvers (using KSP and PC libraries)
- Nonlinear solver (SNES library)
- Portable code
- Reduced experimentation time
  - Selecting different algorithms at run-time!!

# Wrapping up...

*Developing parallel, nontrivial applications that deliver high performance is still difficult and requires months (or even years) of concentrated effort. PETSc is a toolkit that can ease these difficulties and reduce the development time, but it is not a black-box solver, nor a **silver bullet**. – Barry Smith*

## PETSc can help power system applications

- to solve algebraic and DAE problems
- benchmark with different numerical solvers.
- rapidly develop efficient parallel code, can start from examples
- develop new solution methods and data structures
- debug and analyze performance
- advice on software design, solution algorithms, and performance
  - `petsc-{users,dev,maint}@mcs.anl.gov`