SAND2009-7492C

Large Scale Parallel Circuit Simulation

Heidi Thornquist and Eric Keiter
Sandia National Laboratories
Albuquerque, NM

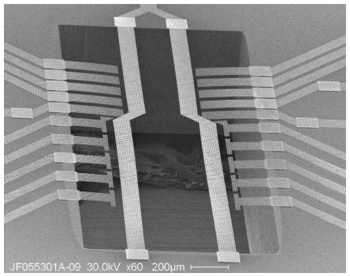
Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company,
for the United States Department of Energy under contract DE-AC04-94AL85000.

Sandia National Laboratories

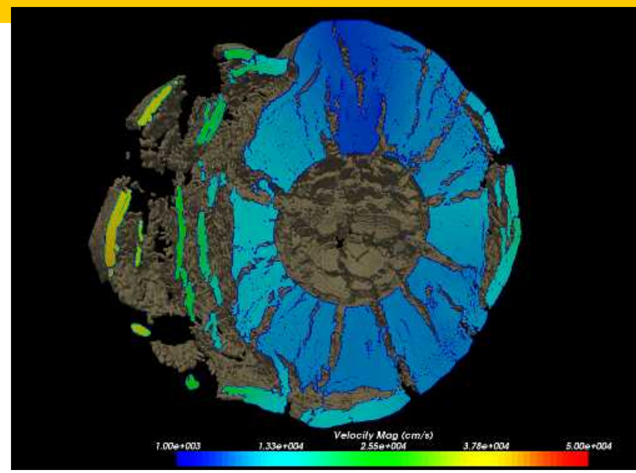
Outline

- Xyce circuit simulator background / Motivation / Evolution
- Trilinos solver library
- Parallel design
  - Parallel parser
  - Independent parallel partition for matrix load and solve
  - Matrix preconditioning
  - Results, 2003
  - Results, 2008
- Solver Design/Trilinos philosphy
- Future developments; multi-core
  - CUDA, TBB, etc

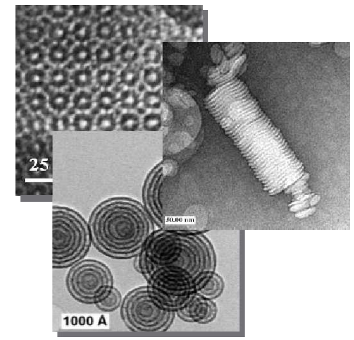
SNL has six core technical capabilities



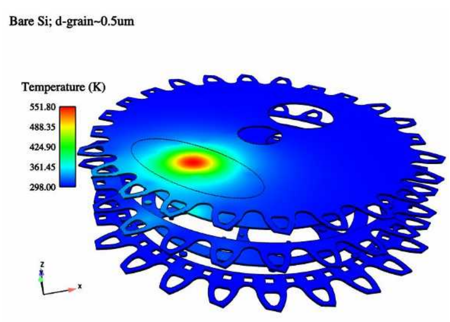
Microelectronics and Photonics



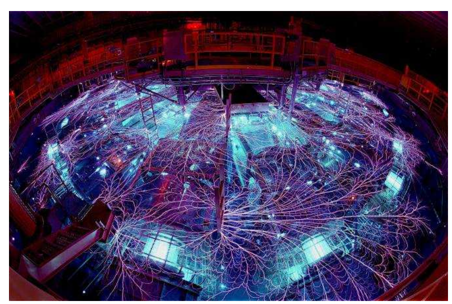
Computational & Informational Sciences



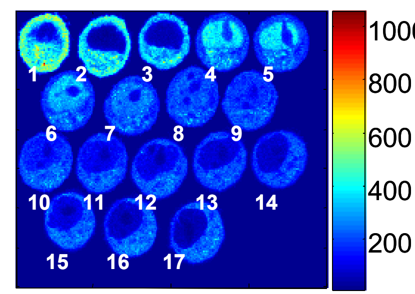
Materials Science & Technology



Engineering Sciences



Pulsed Power



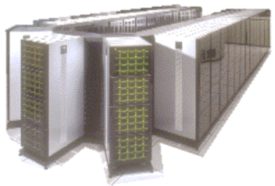
Bioscience

CIS has a rich history in the development and maturation of high performance computing hardware and software technology

CM-2

nCUBE-2

iPSC-860



Paragon

ASCI Red

Cplant

Red Storm

1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007

Gordon Bell Prize

R&D 100
Parallel Software

Karp Challenge

Patent
Parallel Software

R&D 100
Signal Processing

R&D 100
Meshing

Patent
Meshing

Gordon Bell Prize

World Record
281 GFlops

World Record
143 GFlops

R&D 100
Dense Solvers

R&D 100
Storage

Patent
Paving

SC96 Gold Medal
Networking

R&D 100
Aztec

World Record
Teraflops

Gordon Bell Prize

Patent
Decomposition

R&D 100
Salvo

Patent
Data Mining

Mannheim
SuParCup

R&D 100
Trilinos

Fernbach
Award

R&D 100
Allocator

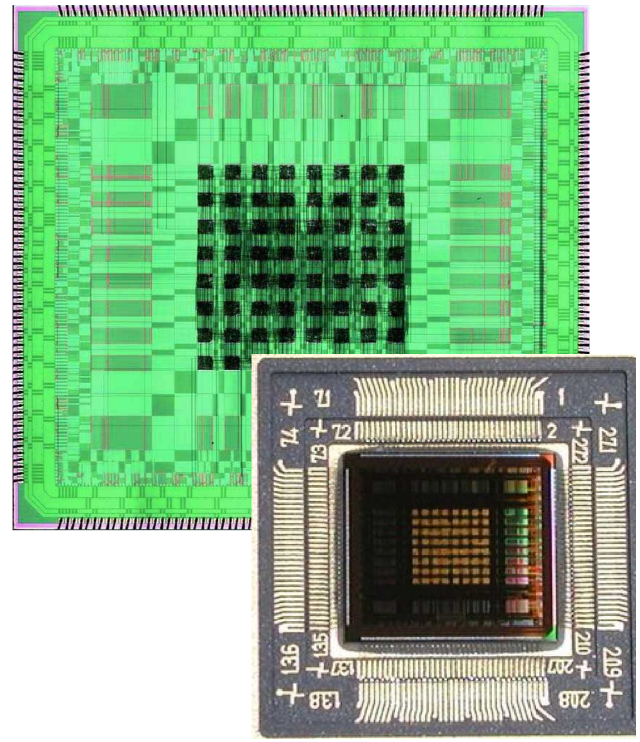
R&D 100
Xyce

R&D 100
3D-Touch

Xyce™ Parallel Circuit Simulator

R&D 100 2008 Award Winner

- Xyce: Massively Parallel circuit simulator:
  - SPICE-Compatible
  - Industry standard models
  - Distributed Memory Parallel (MPI-based for now)
  - Unique solver algorithms
- Unique, Sandia-specific models
  - Prompt Photocurrent
  - Prompt Neutron
  - Thermal
- Xyce Release 5.1
  - 11th major release
  - ~100 internal customers
- http://xyce.sandia.gov



Xyce Motivation

- Lack of NW testing:
  - Comprehensive Test Ban Treaty (CTBT), 1993
  - Advanced Simulation & Computing (ASC), 1995
  - Qualification Alternatives to SPR (QASPR), 2005

- Unique Requirements ➡ Differentiating capabilities
  - Full system simulation
  - Unique models: Radiation Effects
  - High fidelity: "true SPICE" level or higher
  - Large capacity: Massively-parallel

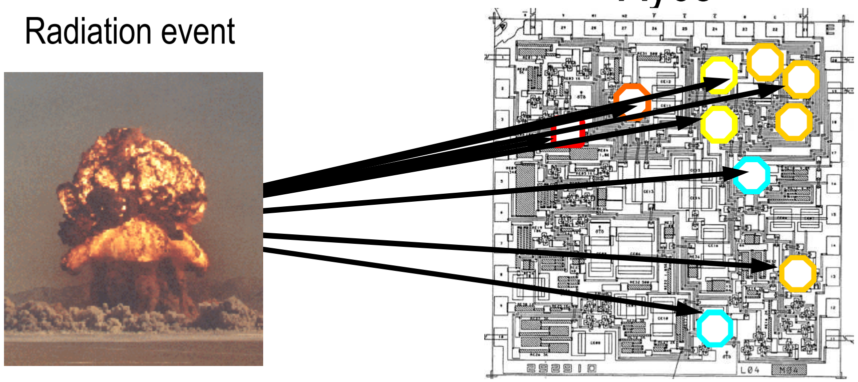
- IP: Sandia owns it, source-level access
- Commercial Tools are expensive ➡ $5K-$1M

Xyce Motivation
Radiation Effects Prediction

Goal: Credible Predictive Simulation

Radiation event

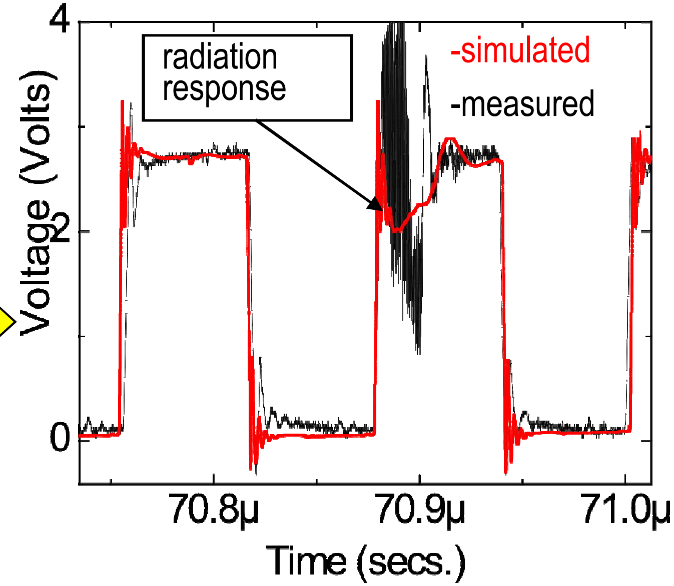
Xyce



Radiation Effects
- Transient Photocurrent (ϒ dot)
- Neutron

Circuit response
- Devices
- Integrated circuit
- Circuit Board

output clock *



* Courtesy of C. Lam and B. Owens

Xyce Parallel Design

- Design Issues/Concerns:
  - To be successful, crucial to design parallel “from the ground up”
  - Type of machines? Capacity or Capability?
  - More flexibility if parallel partitioning is done on the matrix level.
  - Sandia’s expertise: large scale parallel iterative solvers.
  - Circuit simulators usually use direct solvers, but these scale poorly.
  - Sometimes hierarchal methods (FastSPICE) not accurate enough
  - Distributed memory scales much better than shared memory.
  - Emergence of multicore technology.
    - Parallel computing no longer just supercomputers
    - Is MPI enough or do we incorporate options for TBB, CUDA, etc.?

Parallel Bottleneck: Netlist Parsing

- Design for flexibility:
  - On some systems, some nodes may not have I/O
  - On clusters, nodes may have completely independent file systems.
  - Most large netlists are very hierarchical, which makes parallel I/O tricky (but not impossible).
- Lessons learned:
  - Original Xyce parser would do everything on processor 0, and then distribute to other processors.
  - For larger problems, this approach ran out of memory on processor 0.
- Parser redesign:
  - minimal processing on proc 0.
  - MPI_send minimally processed character buffers from proc 0.

Parallel Bottleneck: Netlist Parsing

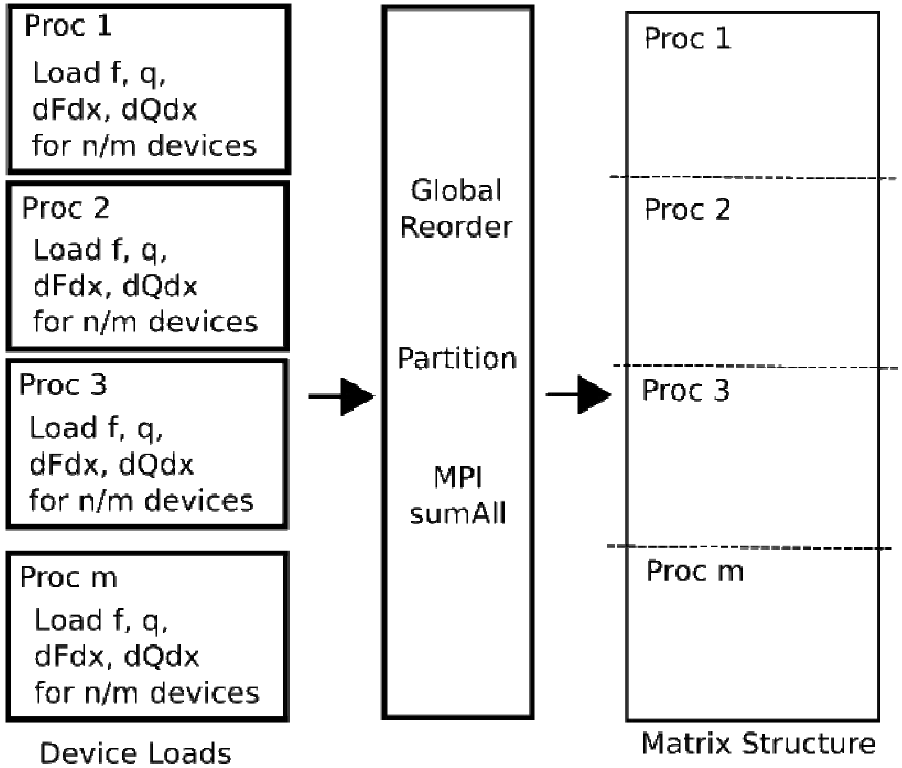
- Xyce parallel parser
  - Netlist is never held in memory. Dynamically query netlist.
  - Netlist is streamed in on processor 0, multiple passes.
  - Pass 1:
    - Local diagnostics
    - Global "Dot" statements (.TRAN, .OPTION, .PRINT, etc) broadcast to all procs.
    - Symbolic flatting, including total device count. Get D/N.
    - File pointers determined: .SUBCKT locations. Most subckt info left in netlist file, not stored in memory.
  - Pass 2:
    - On proc 0, stream in file in blocks of lines at a time.
    - Based on previous symbolic flattening, resolve names (node, device, model)
    - MPI_send resolved devices as raw character buffers to next processor.
    - Once the current "send" processor has D/N devices, move on to next.
    - Each proc allocates devices as they are received, and owned only by that proc.
- This makes parsing scalable, but still mostly serial.
- The D/N distribution establishes the initial naïve parallel partition for device evaluation.

Two Parallel Paritions

Ideal load balance for matrix:
- highly dependent on communication
- circuit topology

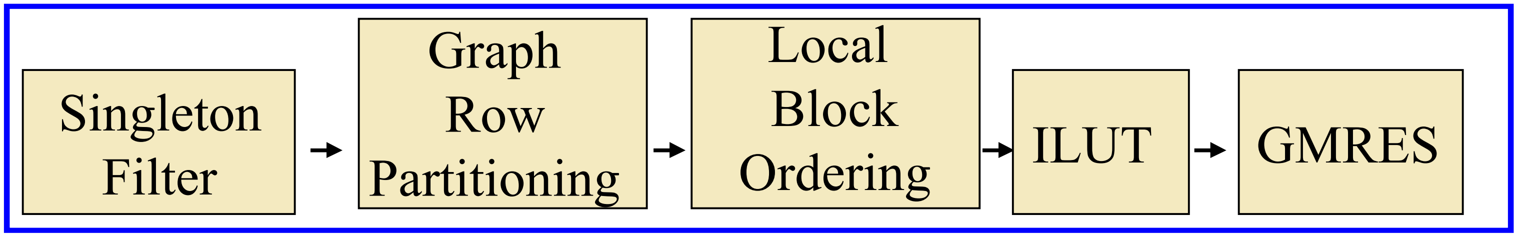
Ideal load balance for device evaluation:
- not much communication
- independent of topology
- need to balance work only
- naïve partition often sufficient

Solution: two parallel partitions
- one for load
- one for solve

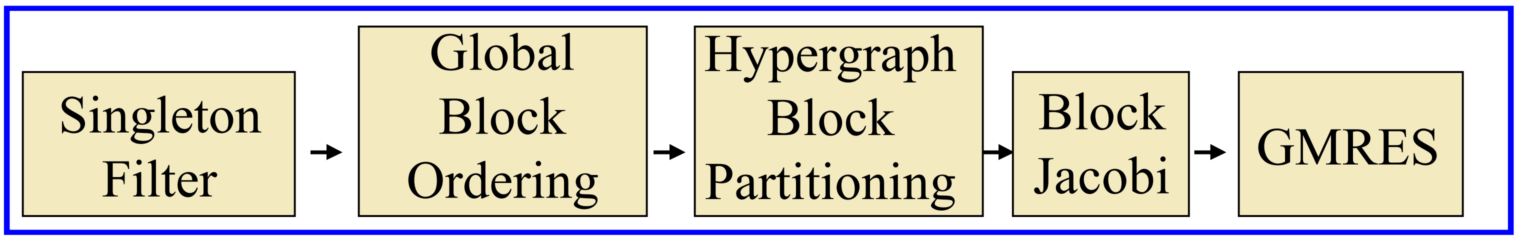


Matrix Strategies

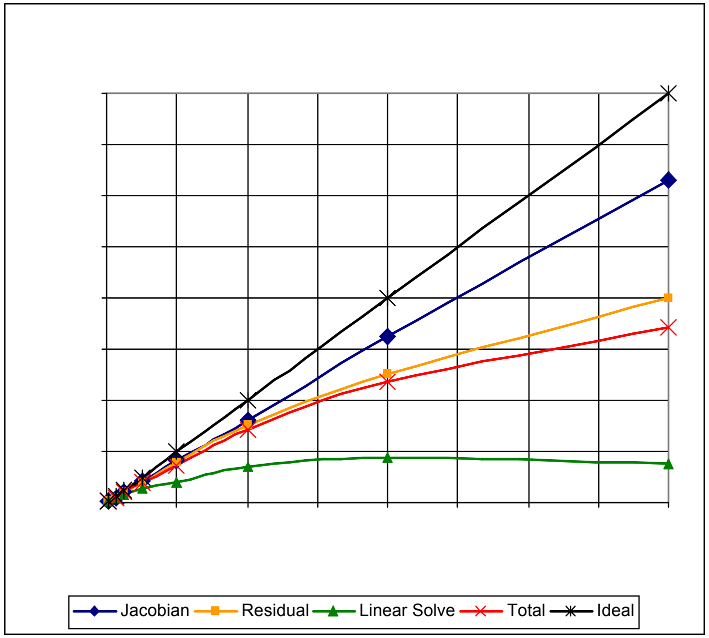
- Strategy 1: (old strategy, circa 2003)



- Strategy 2: (new strategy, circa 2008)

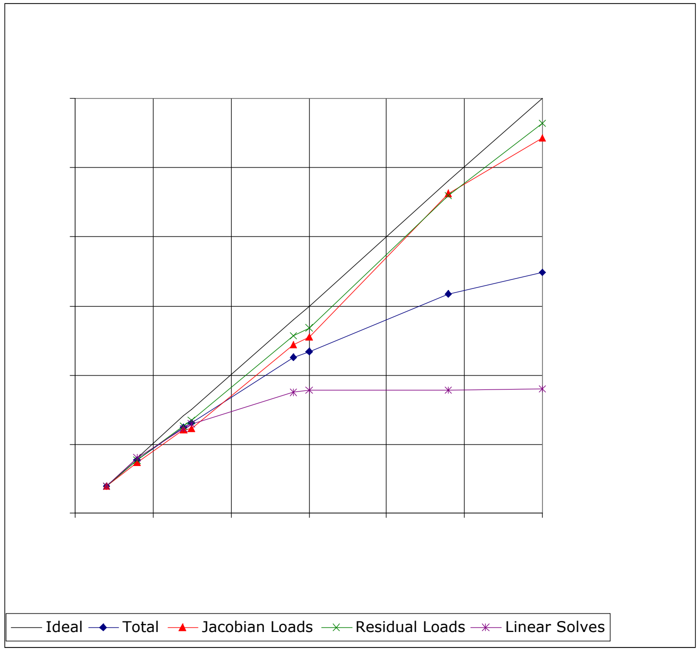


Parallel Scaling Results, circa 2003



Transmission line scaling

variable problem size



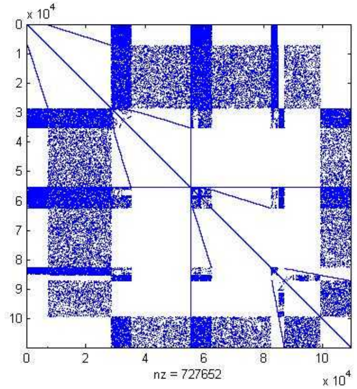
ASIC scaling

fixed problem size

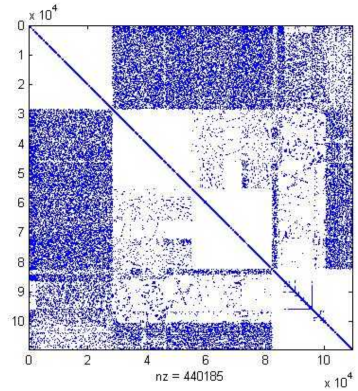
- Transmission line (max size = 14 million devices).
- ASIC scaling on the right. (much harder problem)
- For both problems, roll off occurs in the linear solve phase.

Strategy Comparison: 100K Transistor IC Problem

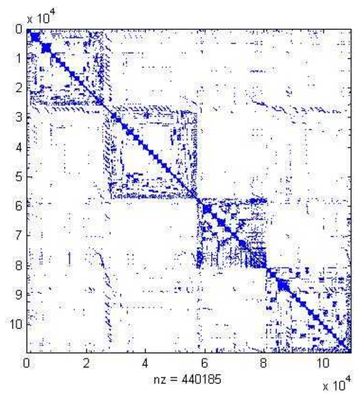
Original



ParMETIS+AMD

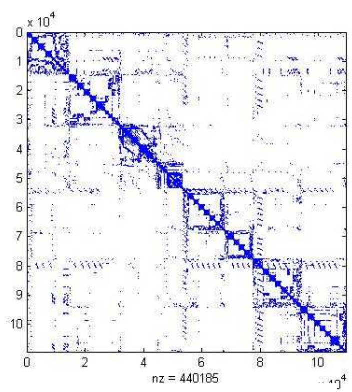


BTF+Hypergraph



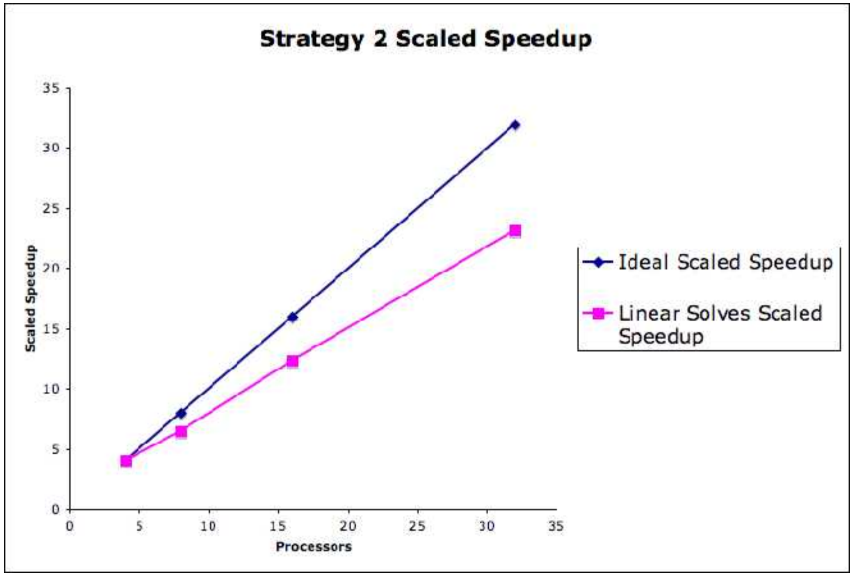
4 processors

BTF+Hypergraph



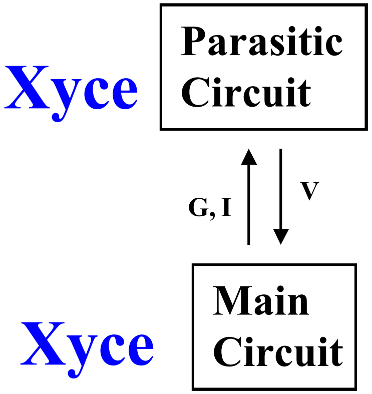
8 processors

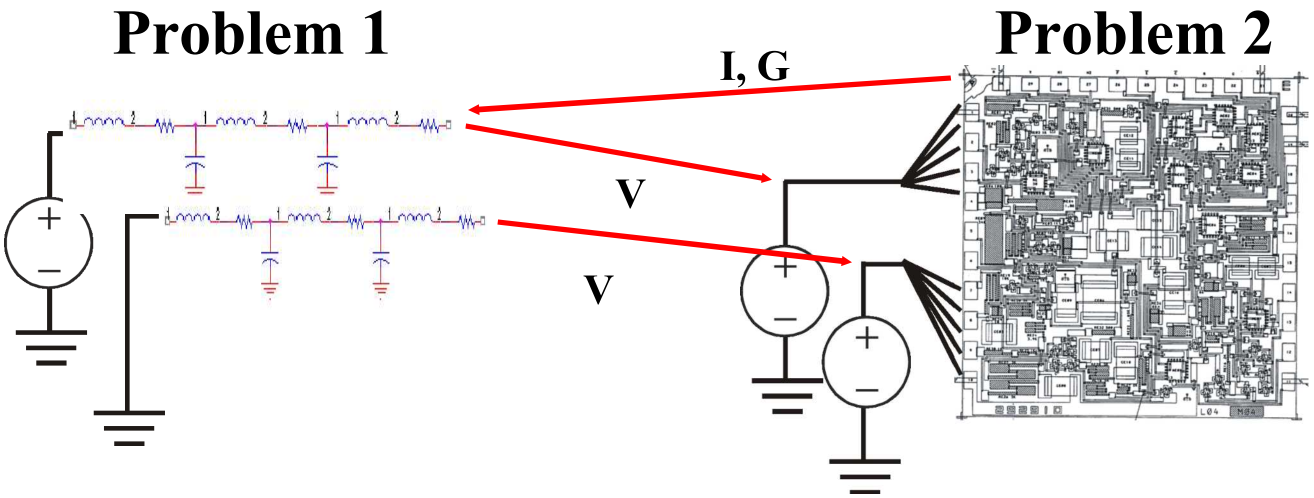
Strategy	Method	Residual	GMRES Iters	Solver Time (seconds)
1	Local AMD ILUT ParMETIS	3.425e-01	500	302.573
2	BTF KLU Hypergraph	3.473e-10	3	0.139



Xyce Multilevel Newton Algorithm

- Problem: non-ideal power supplies break parallel solvers.
- Solution: multi-level Newton
  - Different methods for each level
  - Preserves “singleton removal” algorithm
  - Essential for parallel simulation.





Numerical Solvers and Linear Algebra for Parallel Circuit Simulation

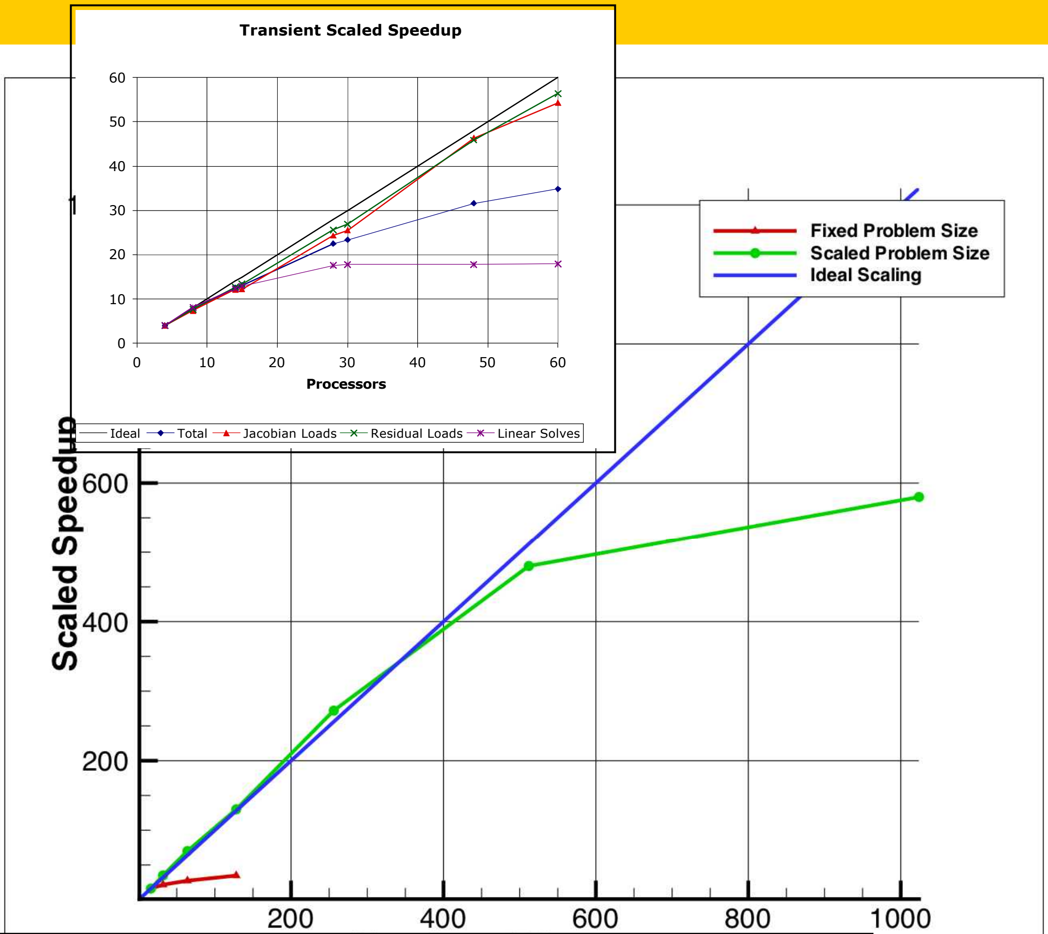
Parallel Circuit Simulation Challenges

Simulating large circuits requires:

- Advanced partitioning
- Parallel solvers
- Preconditioners
- Nonlinear methods (homotopy, continuation)
- Efficient time integration

Parallel performance depends on

- Circuit topology
- Device nonlinearity
- Circuit environment
- Transient duration



Success requires domain experts in a variety of numerical methods

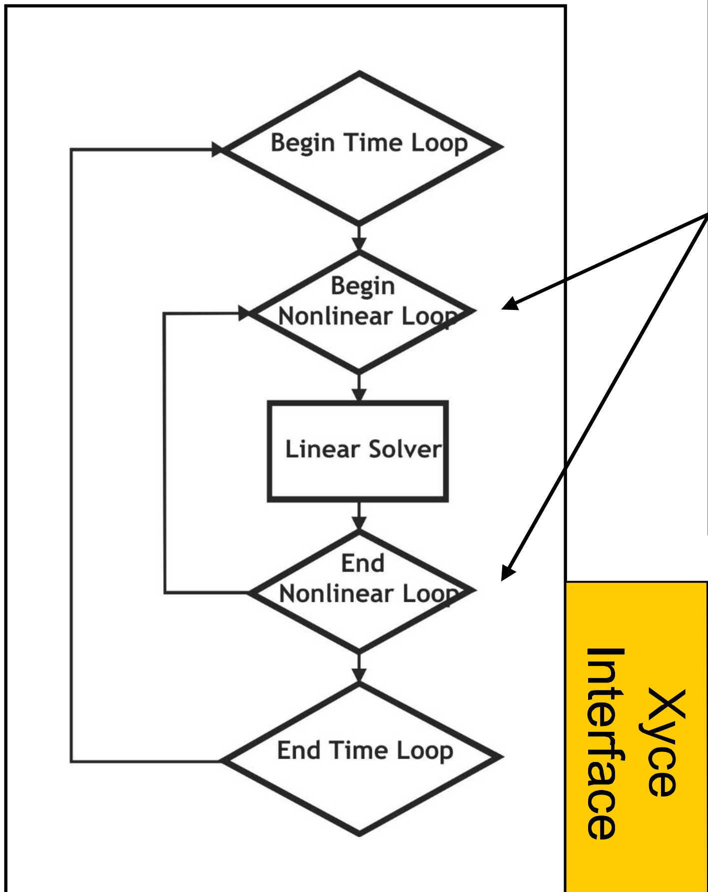
Trilinos

- Trilinos is an evolving framework to support large-scale simulation codes:
  - Fundamental atomic unit is a *package*
  - Includes core set of vector, graph and matrix classes (Epetra/Tpetra packages)
  - Provides a common abstract solver API (Thyra package)
  - Provides a ready-made package infrastructure:
    - Source code management (cvs, bonsai)
    - Build tools (cmake)
    - Automated regression testing (queue directories within repository)
    - Communication tools (mailman mail lists)
  - Specifies requirements and suggested practices to address ASC SQA/SQE requirements
- Trilinos allows the separation of efforts:
  - Efforts best done at the Trilinos level (useful to most or all packages)
  - Efforts best done at a package level (peculiar or important to a package)
  - **Allows package developers to focus only on things that are unique to their package**

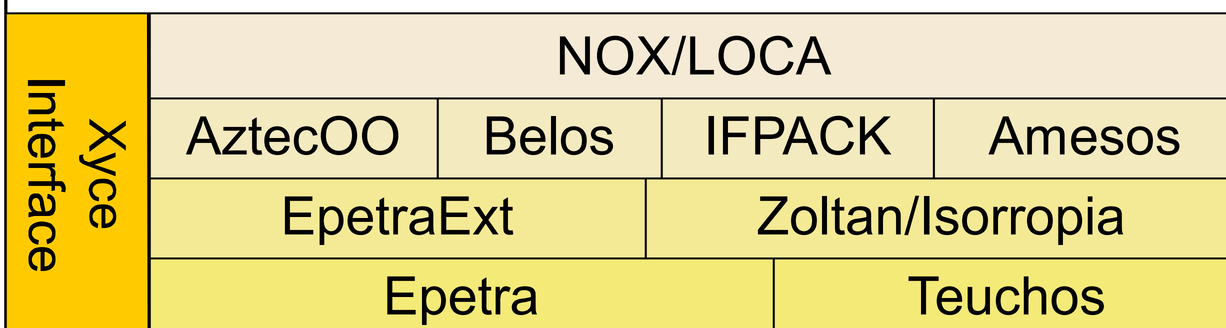
Trilinos Package Summary

	Objective	Package(s)
Discretizations	Spatial Discretizations (FEM,FV,FD)	Intrepid
	Time Integration	Rythmos
Methods	Automatic Differentiation	Sacado
	Mortar Methods	Moertel
Core	Linear algebra objects	Epetra, Jpetra, Tpetra
	Abstract interfaces	Thyra, Stratimikos, RTOp
	Load Balancing	Zoltan, Isorropia
	"Skins"	PyTrilinos, WebTrilinos, Star-P, ForTrilinos
	C++ utilities, (some) I/O	Teuchos, EpetraExt, Kokkos, Triutils
Solvers	Iterative (Krylov) linear solvers	AztecOO, Belos, Komplex
	Direct sparse linear solvers	Amesos
	Direct dense linear solvers	Epetra, Teuchos, Pliris
	Iterative eigenvalue solvers	Anasazi
	ILU-type preconditioners	AztecOO, IFPACK, TIFPACK
	Multilevel preconditioners	ML, CLAPS
	Block preconditioners	Meros
	Nonlinear system solvers	NOX, LOCA
	Optimization (SAND)	MOOCHO, Aristos

Parallel Circuit Simulation Structure
(Transient Simulation)

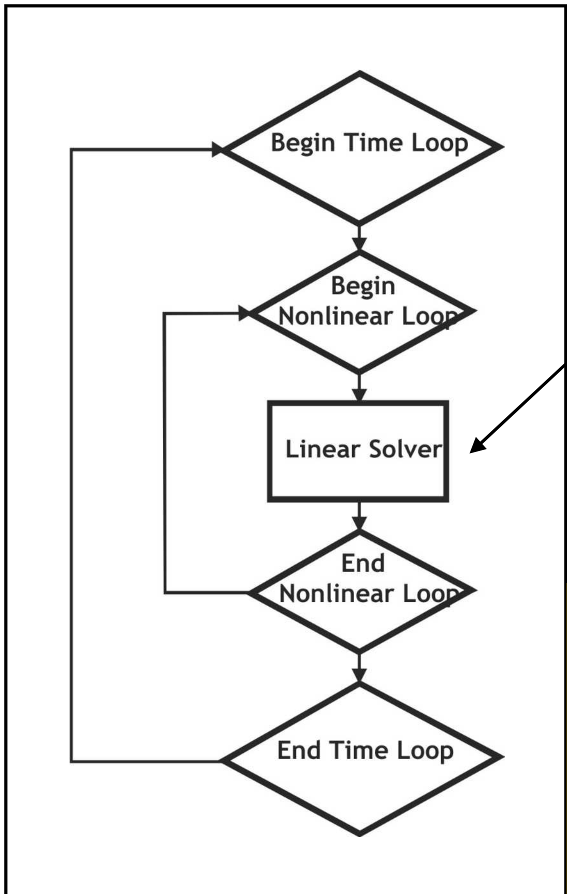


- NOX
  - Suite of nonlinear solution methods
  - Globalizations: Line Search and Trust Region
  - Parallel, OO C++, independent of linear algebra
- LOCA
  - Library of continuation algorithms
  - Zero-order, first-order, arc length, etc.
- Parallel, OO C++, independent of linear algebra

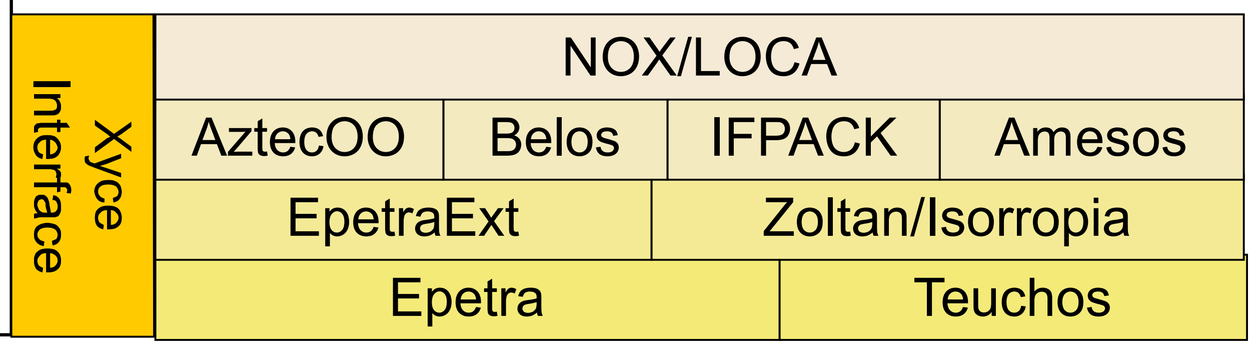


Parallel Circuit Simulation Structure

(Transient Simulation)

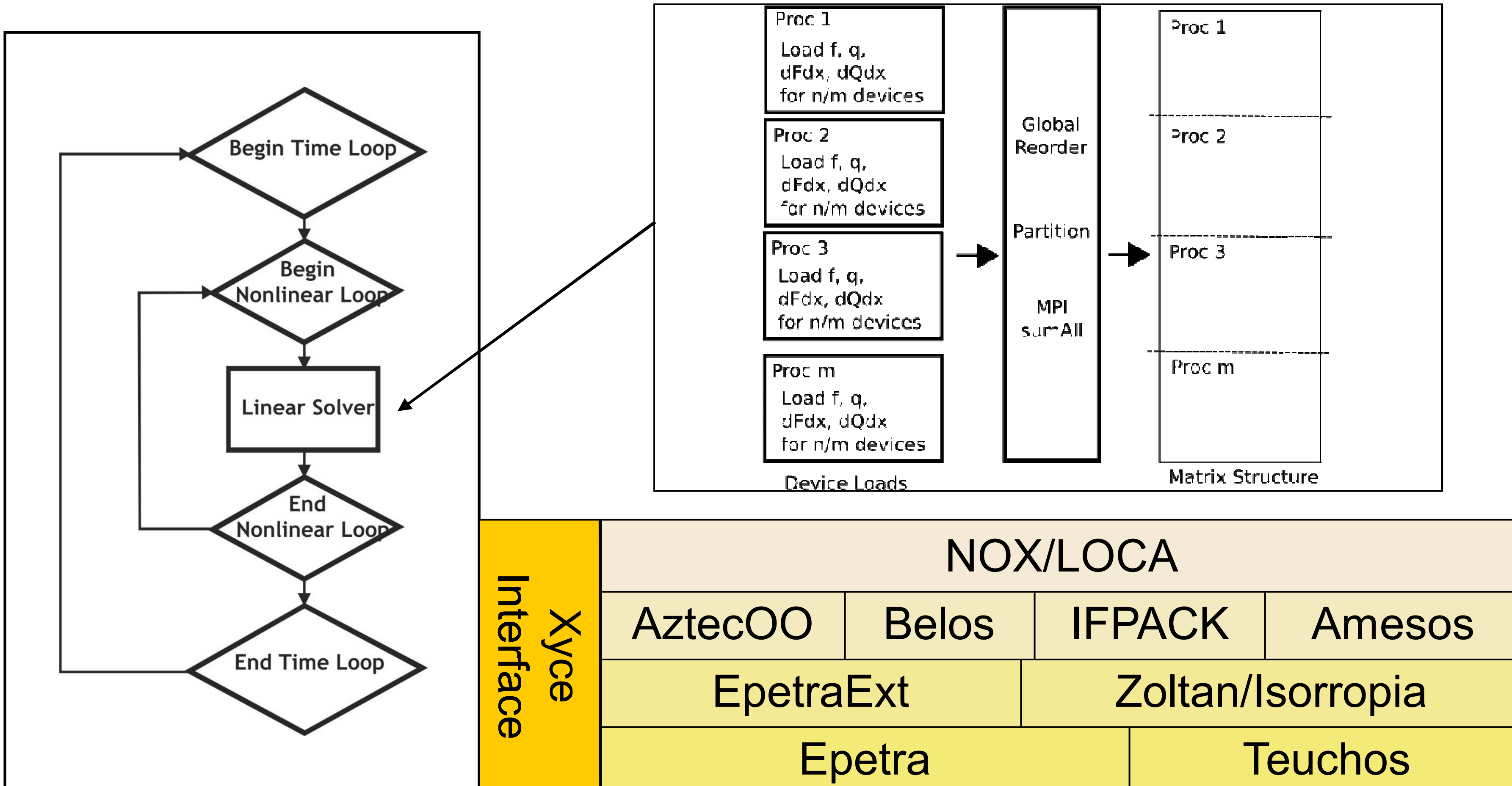


- Circuit characteristics problematic for linear solver
  - Direct solvers more friendly
  - Iterative solvers have potential for better scalability
- Iterative solvers have often been declared unusable for transient circuit simulation
  - Black box methods **do not** work!
  - Need to address these challenges in creation of preconditioner



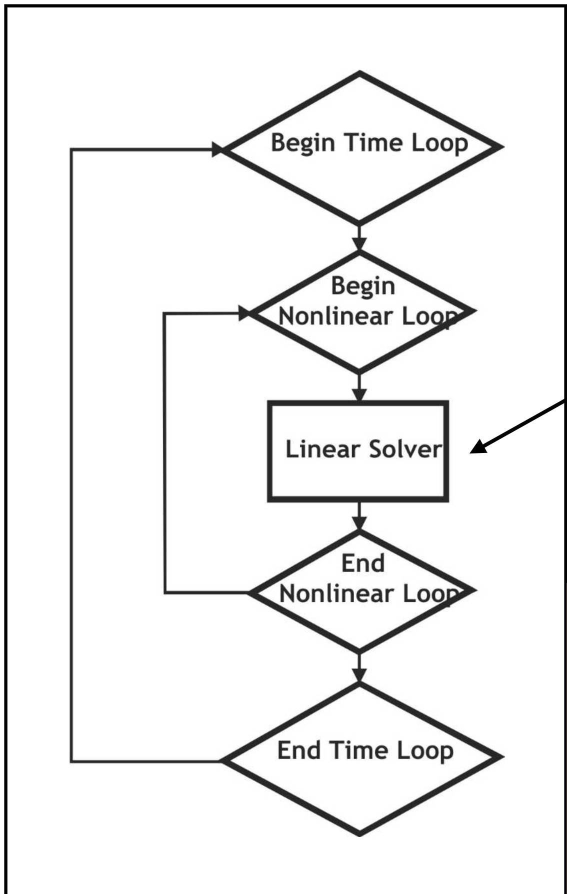
Parallel Circuit Simulation Structure

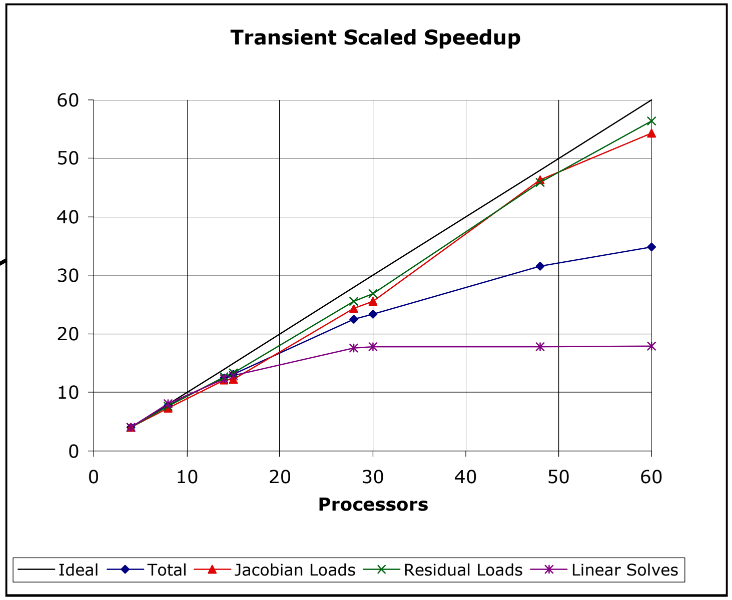
(Transient Simulation)



Parallel Circuit Simulation Structure

(Transient Simulation)

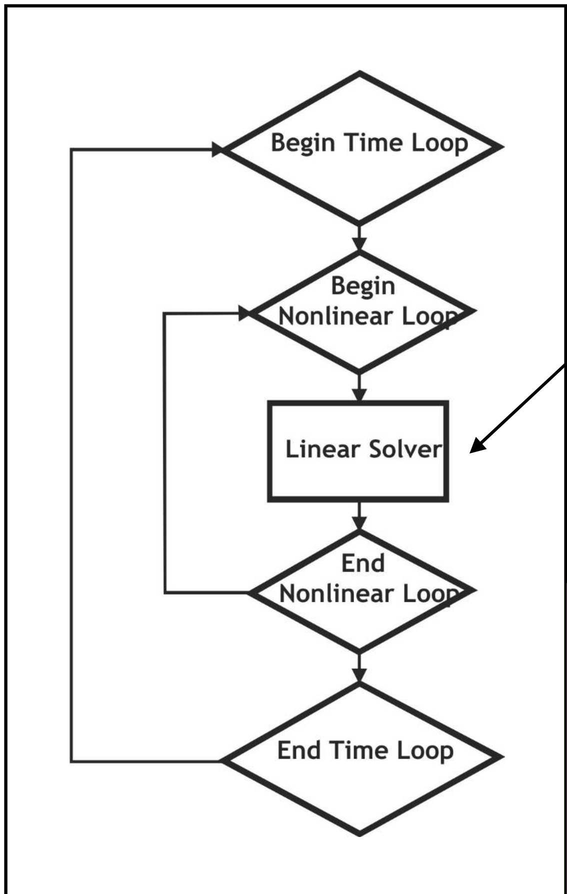




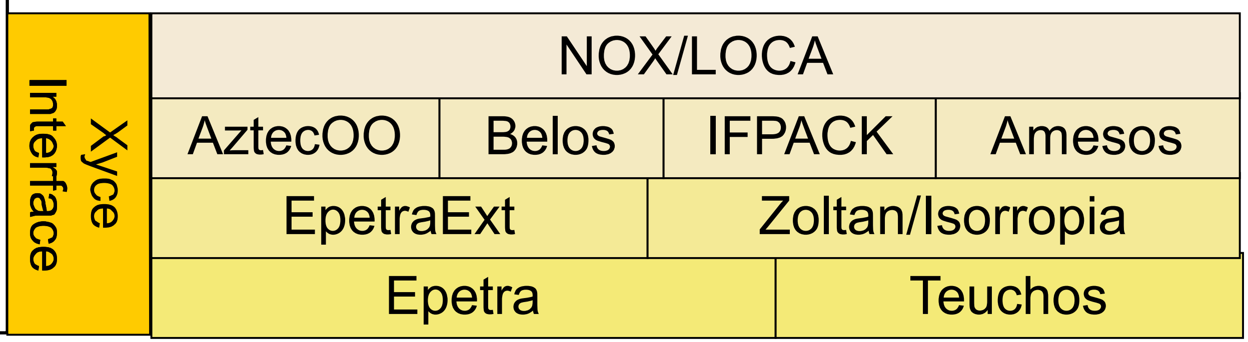
Xyce Interface

NOX/LOCA			
AztecOO	Belos	IFPACK	Amesos
EpetraExt		Zoltan/Isorropia	
Epetra		Teuchos	

Parallel Circuit Simulation Structure
(Transient Simulation)

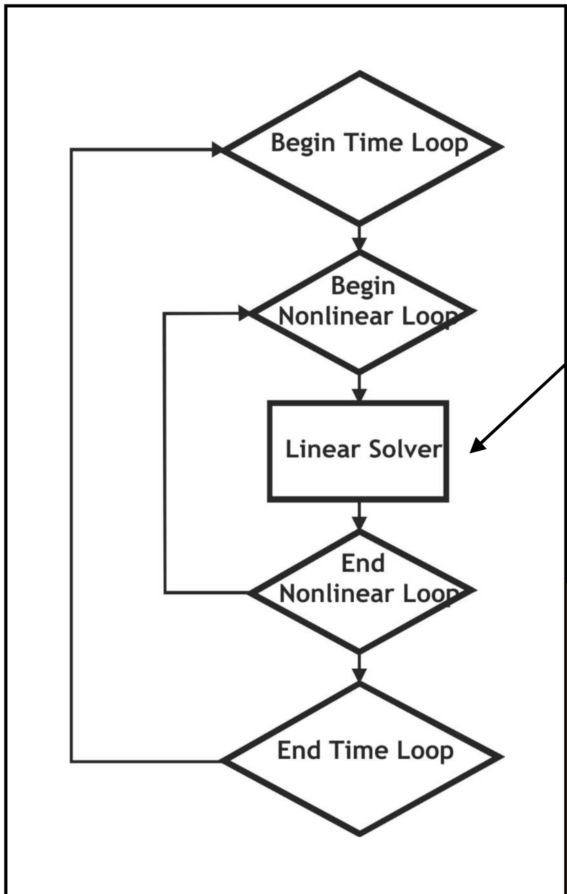


- Amesos
  - Interface to direct solvers for distributed sparse linear systems (KLU, SuperLU, PARDISO, …)
- AztecOO
  - Krylov subspace solvers: CG, GMRES, …
- IFPACK
  - Overlapping Schwarz preconditioners with incomplete factorizations, block relaxations, block direct solves
- Dependent upon Epetra linear algebra

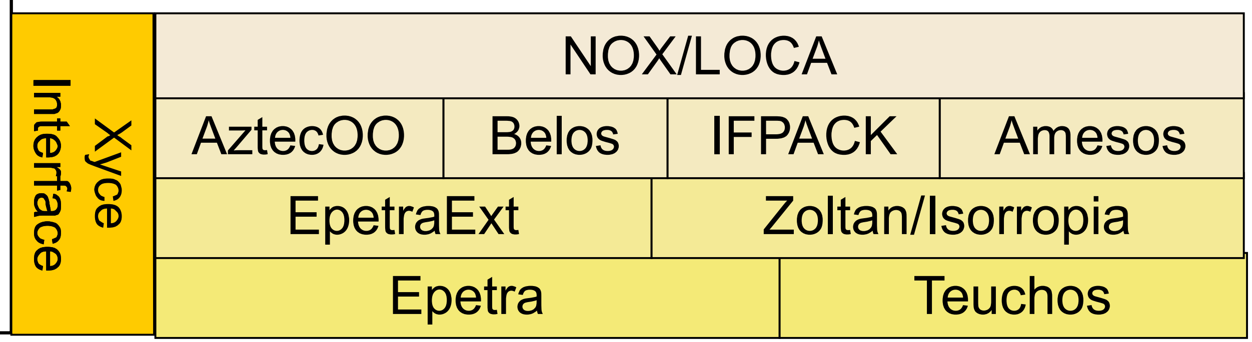


Parallel Circuit Simulation Structure

(Transient Simulation)



- Epetra
  - Petra provides a “common language” for distributed linear algebra objects (operator, matrix, vector)
  - Restricted to real, double precision arithmetic
  - Uses stable core subset of C++ (circa 2000)
- EpetraExt
  - Extensions to Epetra; linear transformations
- Isorropia
  - Interface from linear algebra objects to partitioners

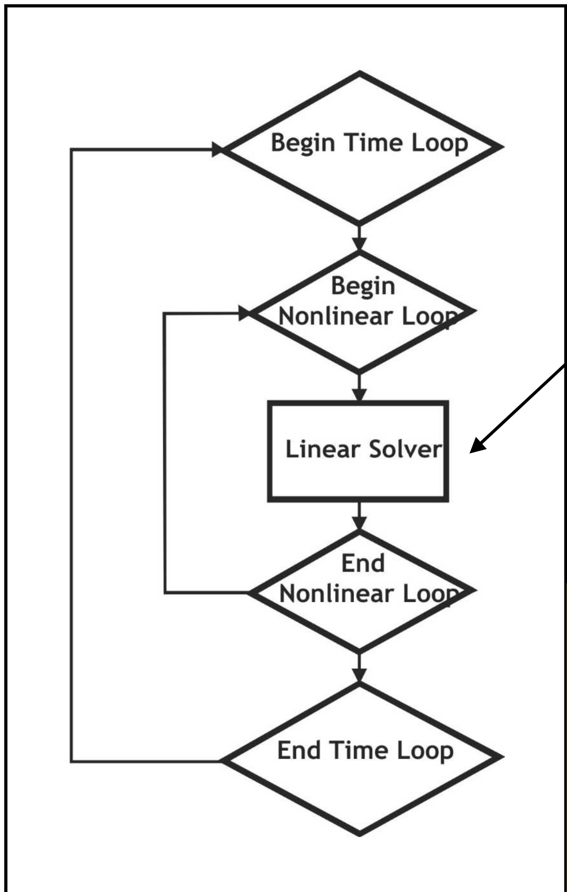


The Impact of Next Generation Computing

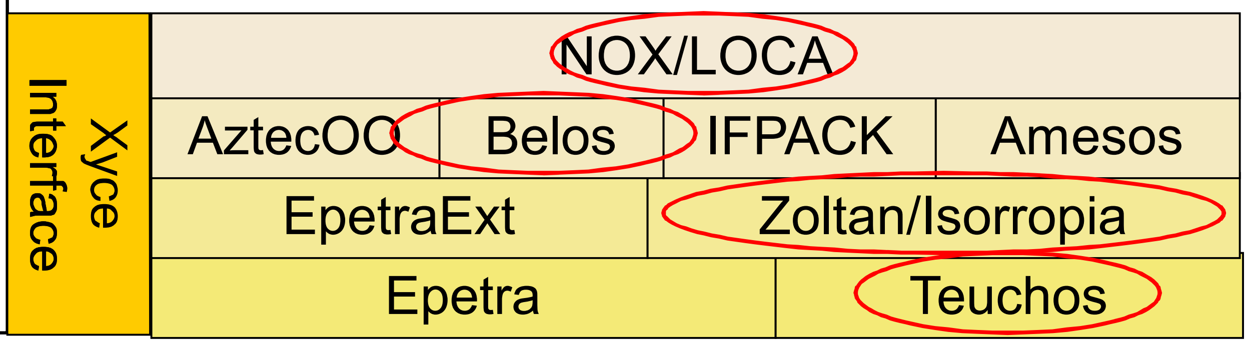
- Parallel computing no longer just supercomputers
- Requires a combination of programming paradigms / languages / data types
- How can we effectively support this?
  - C++ templating of the scalar type (Teuchos)
  - Template Petra object model (Tpetra)
  - Use computational kernels to address architecture differences (Kokkos)
- This provides generic programming capability, independent of data types.
- Templating implements compile time polymorphism
- Pro: No runtime penalty
- Con: Potentially large compile-time penalty
  - Compiling is a good use of multiple cores.
  - Techniques exist for alleviating this for common and user data types (specifically, explicit instantiation).

Parallel Circuit Simulation Structure

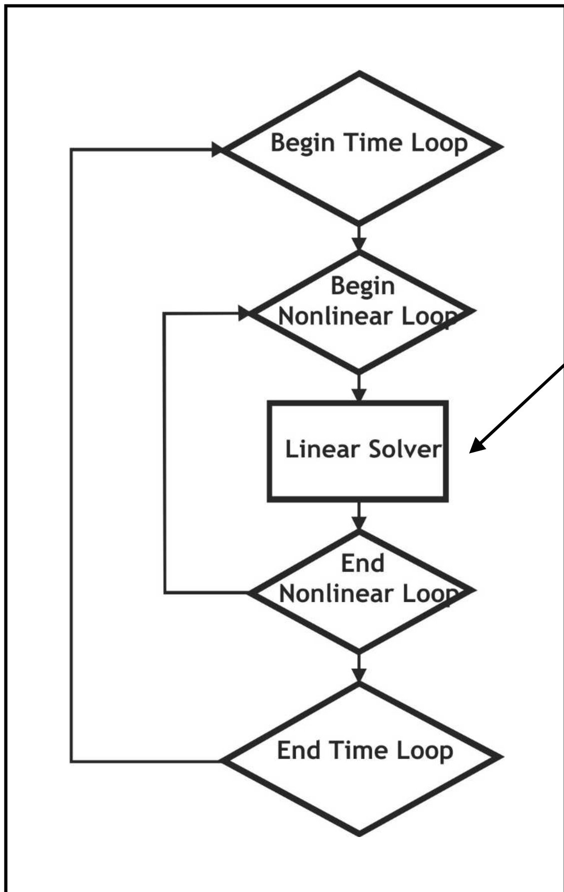
(Future Transient Simulation)



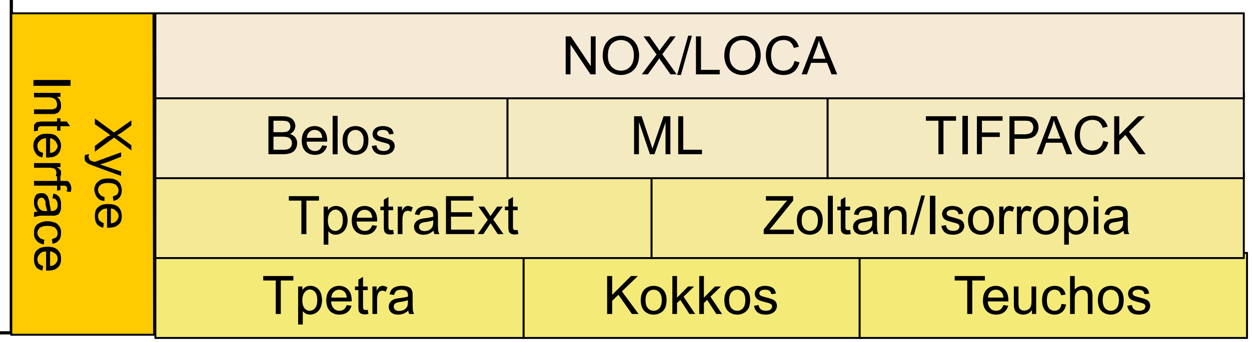
- Teuchos
  - Parameter list, templated BLAS/LAPACK wrappers, serial dense matrix/vector class, smart pointers, …
  - *Ordinal/Scalar Traits support: Defines of 'zero', 'one', etc.*
  - *Generic communicator class*
- Belos
  - Iterative linear solver package, written in templated C++
  - Krylov subspace solvers: CG, GMRES, …
  - Advanced methods: GCRO-DR, RCG, PCPG, …



Parallel Circuit Simulation Structure
(Future Transient Simulation)

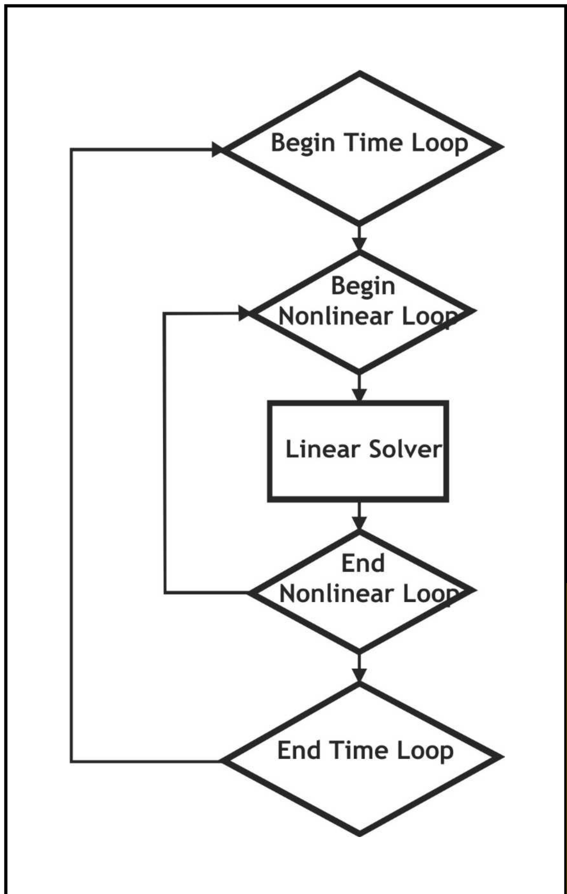


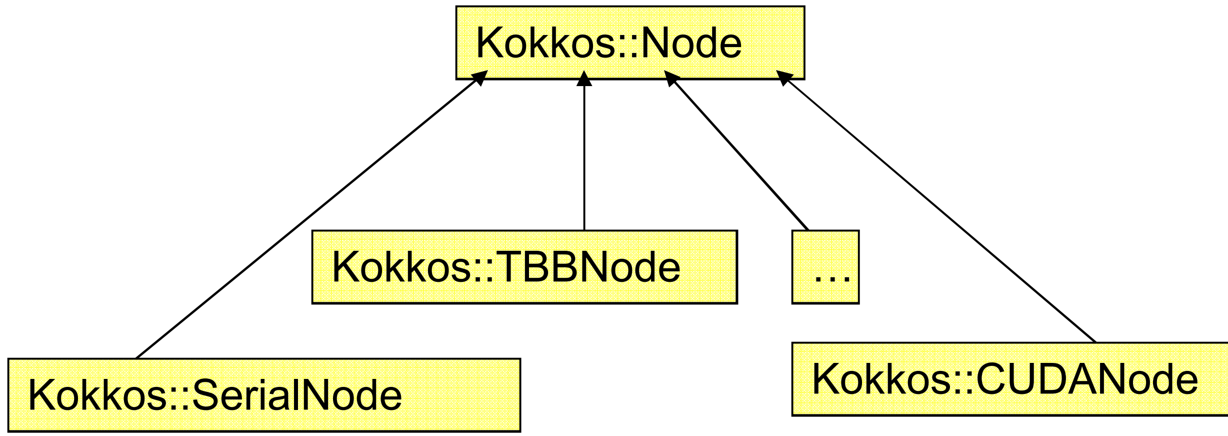
- Tpetra
  - Templated C++ implementation of Petra object model
  - Comm, DistObject, Map, Operator, Vector
- Kokkos
  - Trilinos compute node package
  - Generic Node object defines:
    - Memory structures for parallel buffers
    - Parallel computation routines (e.g., parallel_for, parallel_reduce)
  - Kokkos also employs this API to provide local linear algebra objects for use in Tpetra distributed objects.

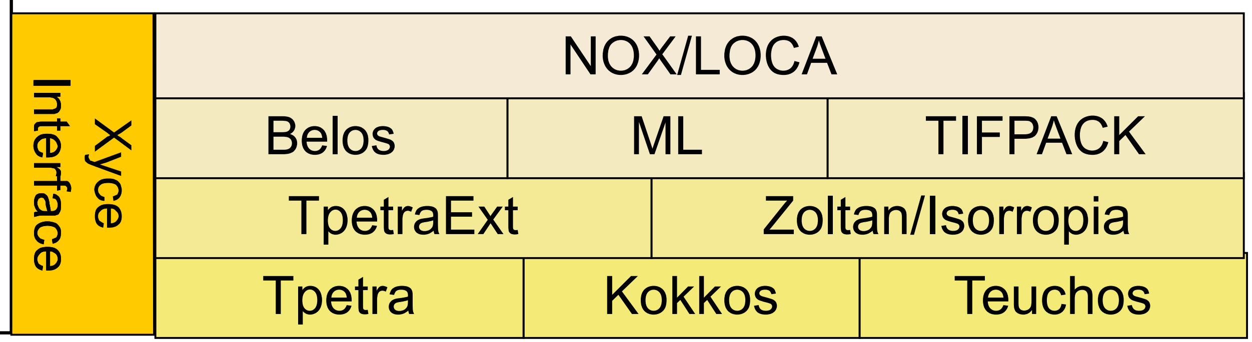


Parallel Circuit Simulation Structure

(Future Transient Simulation)







Acknowledgements

- **Sandia** researchers:
  - David Day
  - Erik Boman
  - Robert Hoekstra
  - Ray Tuminaro
- Xyce Development team

Questions?