SAND2017-3648PE

Xyce Parallel Circuit Simulator

Eric Keiter, and the Xyce development team, et al
Sandia National Laboratories

DARPA IDEA Workshop
Arlington, VA
April 13, 2017

Acknowledgements

Xyce team

xyce.sandia.gov

- Jason Verley (PI)
- Eric Keiter (Research Lead)
- Tom Russo
- Heidi Thornquist
- Rich Schiek
- Ting Mei
- Aadithya Karthik
- Sivasankaran Rajamanickam

Trilinos team

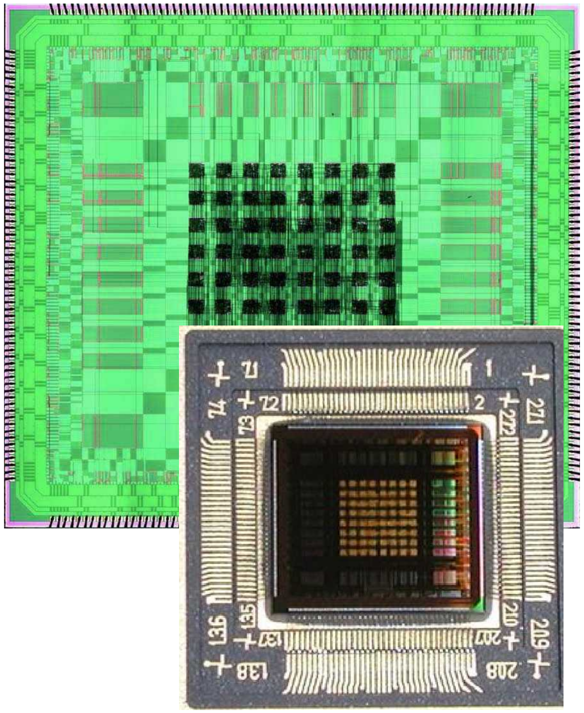
trilinos.sandia.gov

Dakota team

dakota.sandia.gov

Xyce Parallel Circuit Simulator

- Xyce: Massively Parallel circuit simulator:
  - Distributed Memory Parallel (MPI-based)
  - Unique solver algorithms
  - SPICE "Compatible"
  - Industry standard models (BSIM, PSP, EKV, VBIC, etc)
  - ADMS model compiler
- Analysis types
  - DC, TRAN, AC
  - Harmonic Balance (HB)
  - Multi-time PDE (MPDE)
  - Model order reduction (MOR)
  - Direct and Adjoint sensitivity analysis
- Sandia-specific models
  - Prompt Photocurrent
  - Prompt Neutron
  - Thermal
- Other, non-traditional models
  - Neuron/synapse
  - Reaction network
  - TCAD (PDE-based)
- Xyce Release 6.7 pending
  - Open Source!
  - GPL v3 license



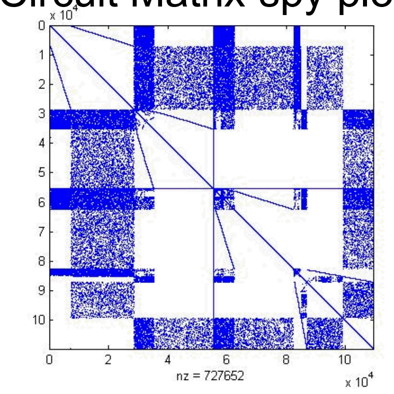
http://xyce.sandia.gov

Open Source Releases (starting in 2013):
Versions 6.0, 6.1, 6.2, 6.3, 6.4, 6.5, 6.6
>1000 unique external downloads since 6.0.
Next release (v6.7) ~May 2017

What Xyce Is, and Is Not

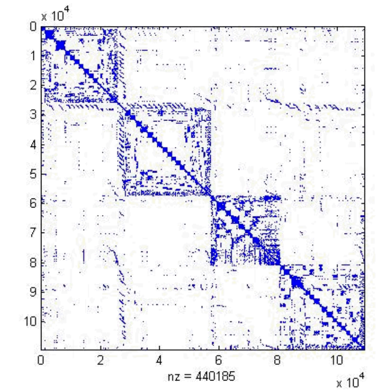
- **Xyce is: "True Spice"**
  - Large, monolithic, single Jacobian matrix.
  - Accurate.
  - Known parallel linear solvers don't scale perfectly.

Circuit Matrix spy plot



- **Xyce is not (currently): "Fast Spice"**
  - Loosely coupled separate blocks
    - Implicit solver methods within blocks
    - Explicit methods used to couple blocks
  - Table models
  - Model order reduction
  - Exploits circuit hierarchy
  - Effective primarily for digital circuits
  - less accurate than "true spice"

BTF+Hypergraph

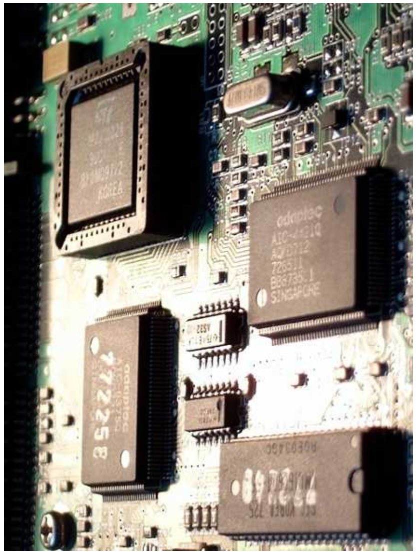


4 processors

Sandia National Laboratories

"True Spice" Xyce Circuit Simulation Challenges

- Need to efficiently and seamlessly simulate circuits that have a wide range of devices ($10\text{-}10^6$) and complexity
  - Advanced preconditioning techniques for iterative solvers
  - Must support direct solvers for smaller circuits
- Internal Customer simulation efforts inspired many of the past and current advancements for circuit simulation

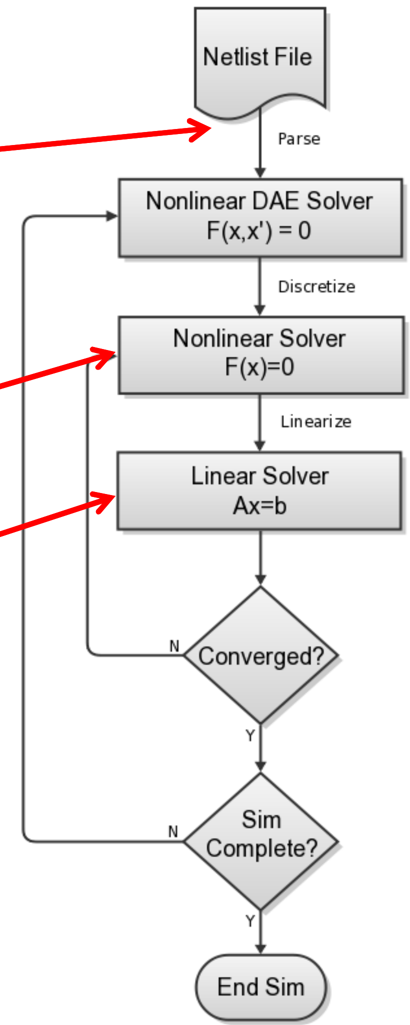


Circuit 1 (2000) → Circuit 2 (2003) → Circuit 3 (2006) → Circuit 4 (2008) → Circuit 5 (2011)

~25K unknowns | ~600K unknowns* | ~680K unknowns | ~430K unknowns | ~2M unknowns

Xyce Simulation Flow

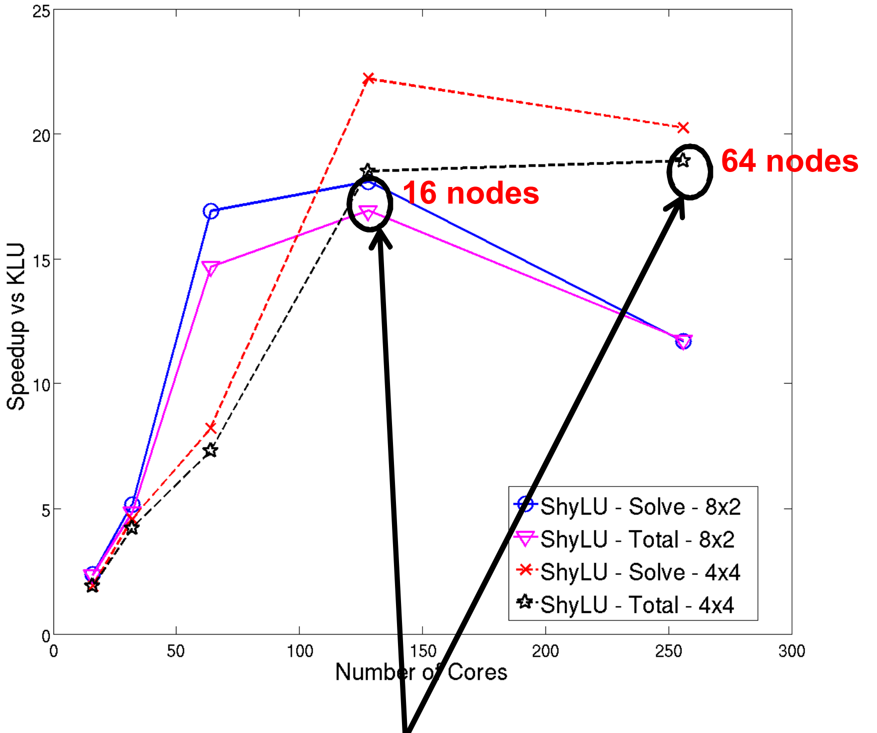
- Parsing
  - Convert netlist file syntax to equivalent devices and network/circuit connectivity
  - Distribute devices over multiple processors
  - Determine global ordering and communication
- Device Evaluation
  - Loop through all devices for state evaluation and matrix loading
- Linear Solve
  - Sparse linear algebra and solvers used to solve linearized system
- Advanced Analysis Methods
  - Sampling: Monte Carlo, LHS (DAKOTA)



New Linear Solver Achieves 19x Speedup for ASIC Simulation

- ShyLU is a sparse linear solver framework, based on Schur complements (*S. Rajamanickam, E. Boman, M. Heroux*)
  - Incorporates both direct and iterative methods
  - Coarse-scale (multi-processor) and fine-scale (multi-threaded) parallelism
  - Can be a subdomain solver / preconditioner or stand-alone linear solver

- In-house ASIC: 1.6M total devices, ~ 2M unknowns:
  - Xyce w/ KLU solver takes ~ **2 weeks**, w/ ShyLU solver takes ~ **1 day**
  - ShyLU: Optimal # partitions = 64; number of rows in *S* = 1854 (4 MPI procs)



Strong scaling of Xyce's simulation time and ShyLU linear solve time for different configurations of MPI Tasks X Threads per node on TLCC

Xyce Model Support with ADMS

ADMS = *Automatic Device Model Synthesizer*

Verilog-A: industry standard format for new models:e.g. VBIC, Mextram, EKV, HiCUM, etc

ADMS translates Verilog-A to Xyce-compliant C/C++ code;

API automatically handles data structures, matrices, tedious details.

itivities via Sacado automatic differentiation

nd to include Stocahstic Expansions via Stokhos.

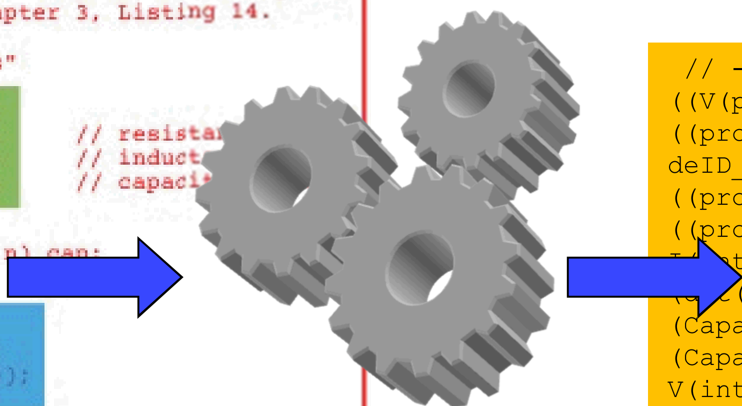
```
// Series RLC
// Version 1a, 1 June 04
// Ken Kundert
//
// Downloaded from The Designer's Guide Community
// (www.designers-guide.org).
// Taken from "The Designer's Guide to Verilog-AMS"
// by Kundert & Zinke. Chapter 3, Listing 14.

`include "disciplines.vams"

module series_rlc2 (p, n);
    parameter real r=1000;     // resista
    parameter real l=1e-9;     // induct
    parameter real c=1e-6;     // capaci
    inout p, n;
    electrical p, n, i;
    branch (p, i) rl, (i, n) cap;

    analog begin
    V(rl) <+ r*I(rl);
    V(rl) <+ ddt(l*I(rl));
    I(cap) <+ ddt(c*V(cap));
    end
endmodule
```

Verilog-A



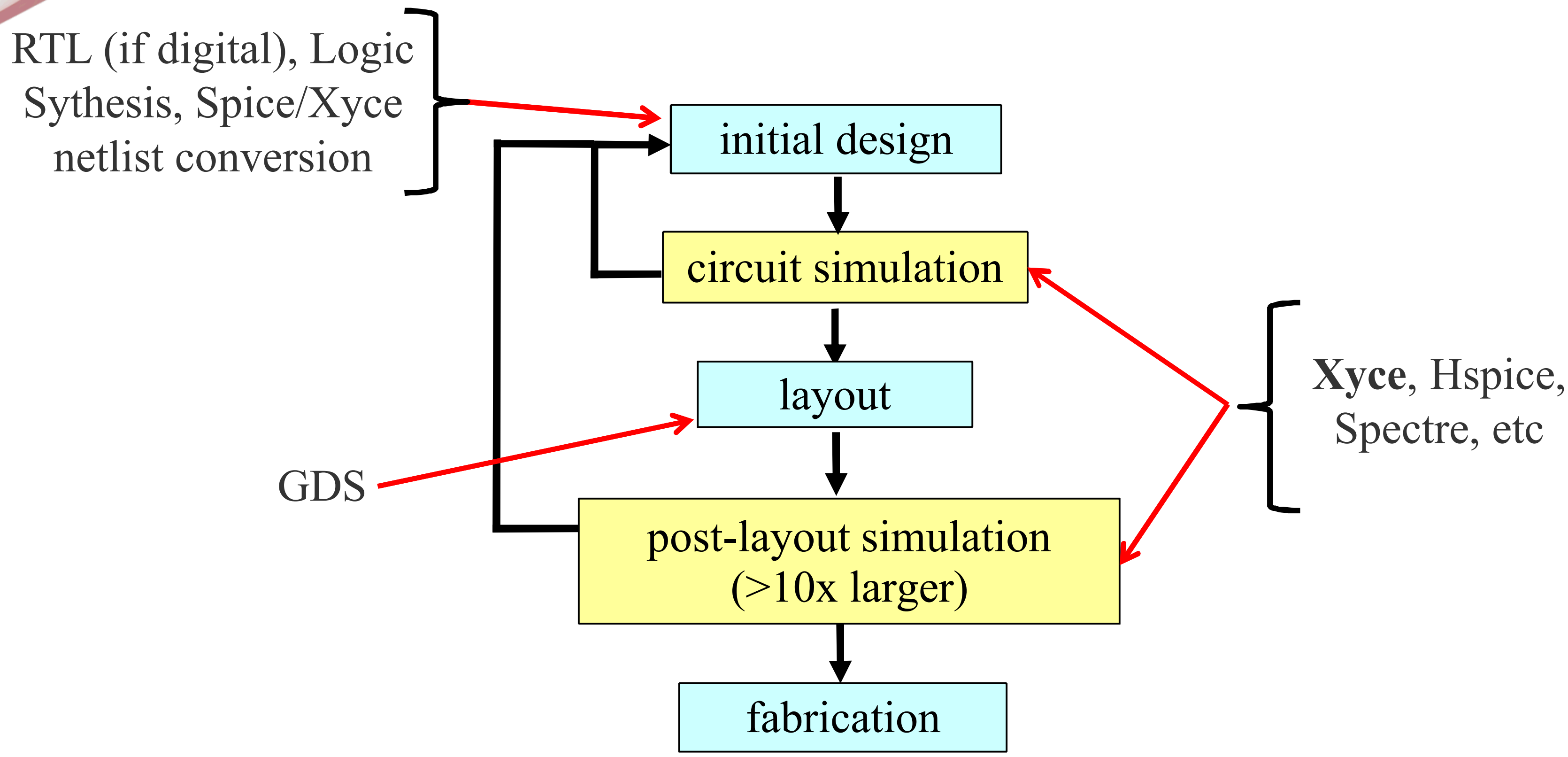
Run admsXyce

```
 // -- code converted from analog/code block// I(
((V(p,internal1)/R))staticContributions[admsNodeI
((probeVars[admsProbeID_V_p_internal1])/instanceP
deID_internal1] -=
((probeVars[admsProbeID_V_p_internal1])/instanceP
((probeVars[admsProbeID_V_internal1_internal2])*i
I  ternal1,internal2) <+
    (CapacitorCharge))dynamicContributions[admsNo
(CapacitorCharge);dynamicContributions[admsNodeID
(CapacitorCharge);InductorCurrent = (probeVars[ad
V(internal2,n) <+
((L*ddt(InductorCurrent)))dynamicContributions[ad
(instancePar_L*(InductorCurrent));
```

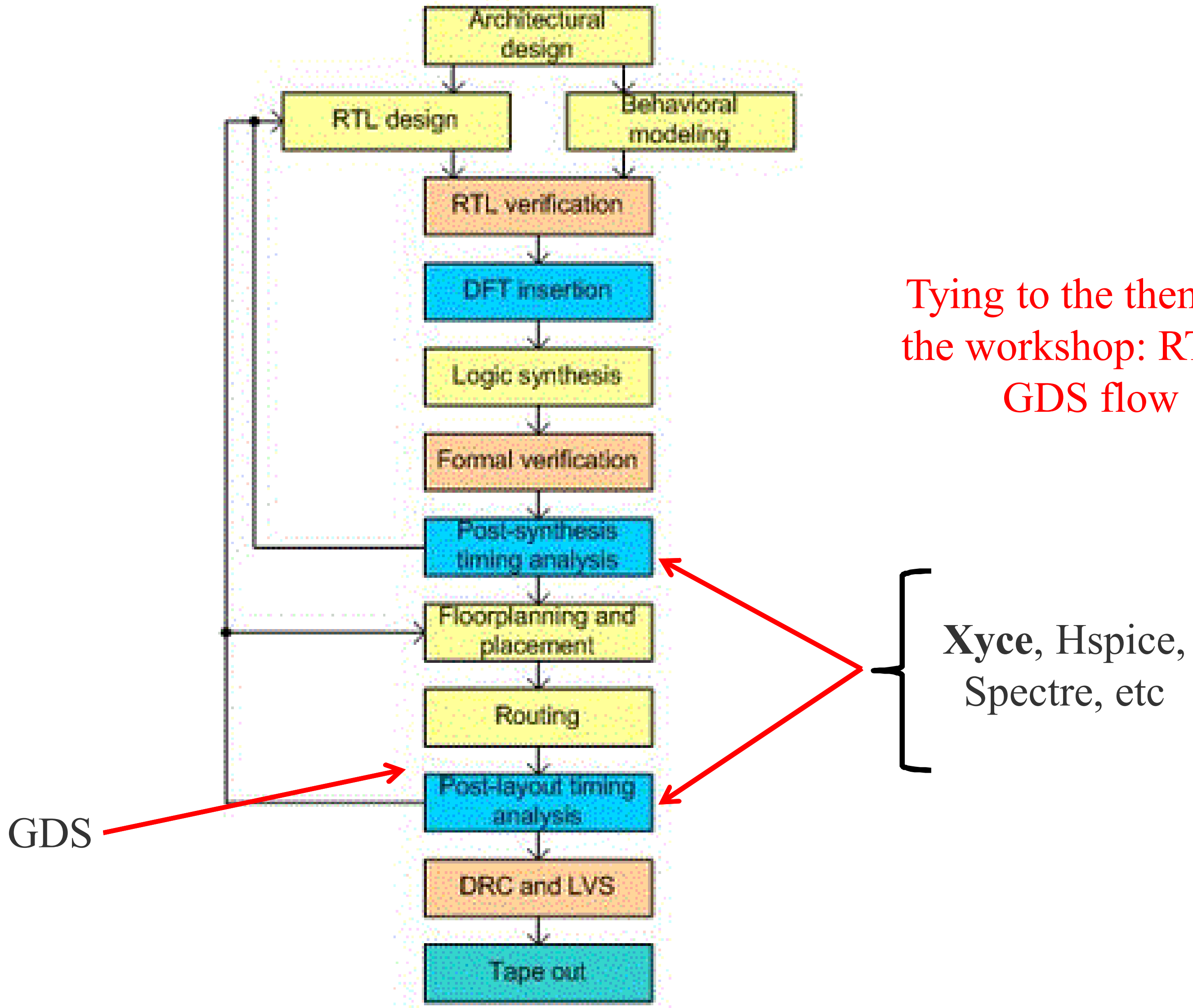
C++ code snippet

(actual Xyce file is 1500 lines)

Tool Flow


RTL (if digital), Logic Sythesis, Spice/Xyce netlist conversion
initial design
circuit simulation
layout
GDS
post-layout simulation (>10x larger)
fabrication
Xyce, Hspice, Spectre, etc

Tool Flow (More Detail, ASIC)

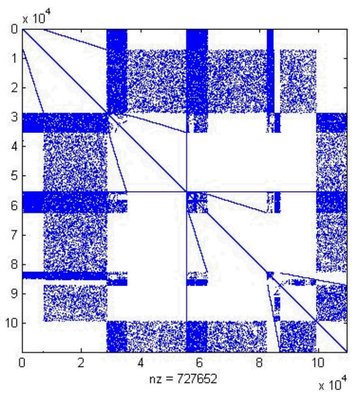


Tying to the theme of the workshop: RTL to GDS flow

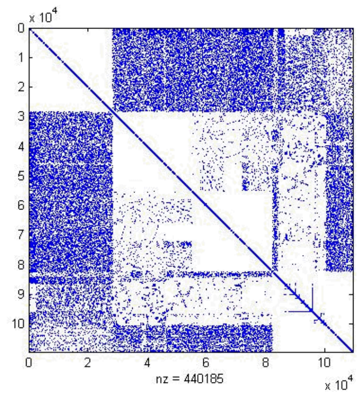
Extra Slides

Linear Solver Strategy Comparison: 100K Transistor IC Problem

Original

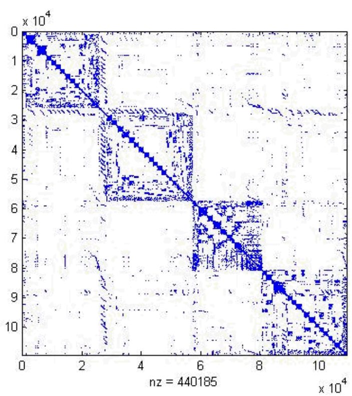


ParMETIS+AMD



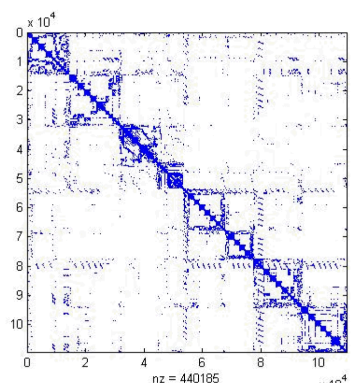
Strategy	Method	Residual	GMRES Iters	Solver Time (seconds)
1	Local AMD ILUT ParMETIS	3.425e-01	500	302.573
2	BTF KLU Hypergraph	3.473e-10	3	0.139

BTF+Hypergraph

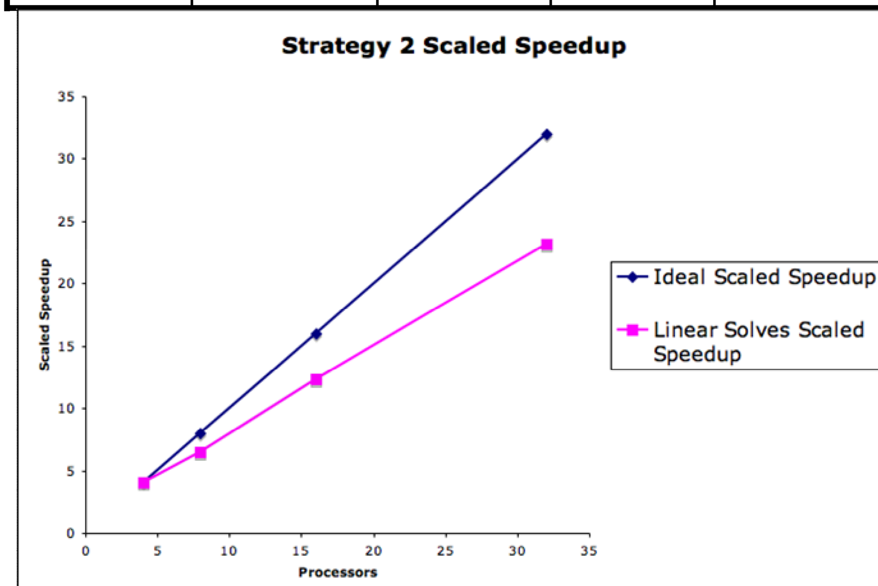


4 processors

BTF+Hypergraph

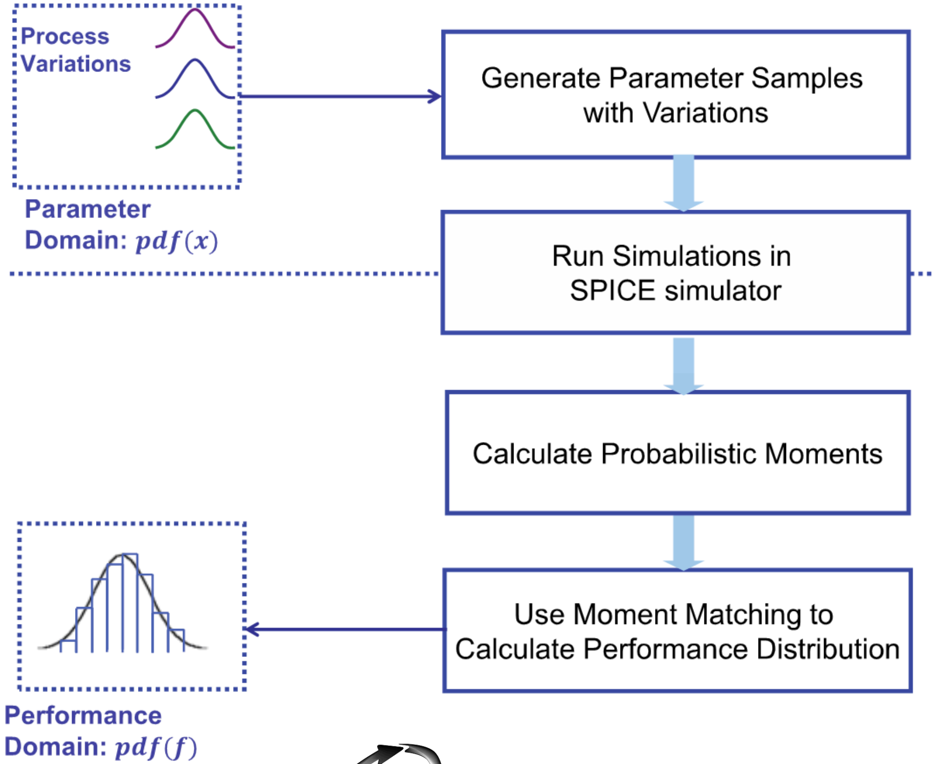


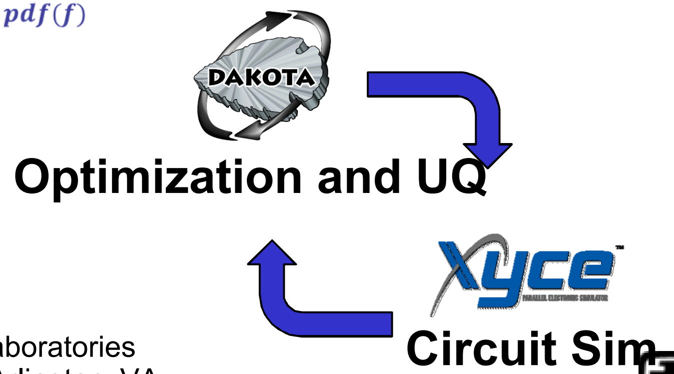
8 processors



Optimization and Uncertainty Quantification (UQ)

- Xyce has a number of capabilities to support uncertainty quantification (UQ) and/or optimization.
- Mostly these come via the DAKOTA framework, or from ROL.
- General idea: given uncertainty on parameter inputs, how to estimate that on circuit outputs.
- Most simulators support “brute force” approaches based on sampling.
- Many much more sophisticated methods exist, including stochastic collocation methods, etc.





Embedded sensitivity calculations

- Many advanced UQ techniques can use parameter derivatives.
  - Failure analysis methods
  - Regression-based Polynomial Chaos (PCE)
  - Gradient-based optimization
- Embedded sensitivities in an application code is better than relying on finite differences:
  - much faster (single forward solve)
  - much more accurate.

- Sensitivities can be useful by themselves. Consider an SEE (Single Event Effects) example:
  - Want to apply SEE mitigations (redundant circuitry) to a large IC.
  - Want to apply them selectively, only where absolutely necessary.
  - Which transistors of a circuit are most sensitive to SEE?
  - An adjoint method can determine this for you, in a single calculation → determines sensitivity of an output with respect to currents applied at each transistor.

Parameter sensitivities

- Xyce (v6.5) now supports:
  - Direct and Adjoint steady state (.DC)
  - Direct and Adjoint transient (.TRAN)
- Parameter sensitivities (steady state):

$$\left(\frac{\partial F}{\partial x}\right)\frac{\partial x}{\partial p} = \frac{\partial F}{\partial p}$$

$$\frac{dO}{dp} = \frac{\partial O}{\partial x}\left(\frac{\partial F}{\partial x}\right)^{-1}\frac{\partial F}{\partial p}$$

- $\frac{\partial O}{\partial x}$: Handled by the Xyce expression library
- $\left(\frac{\partial F}{\partial x}\right)^{-1}$: Xyce Jacobian Matrix (inverse)
- $\frac{\partial F}{\partial p}$: Provided by device models (analytic or finite difference)

Direct

$$\frac{dO}{dp} = \frac{\partial O}{\partial x}\left[\left(\frac{\partial F}{\partial x}\right)^{-1}\frac{\partial F}{\partial p}\right]$$

Adjoint

$$\frac{dO}{dp} = \left[\frac{\partial O}{\partial x}\left(\frac{\partial F}{\partial x}\right)^{-1}\right]\frac{\partial F}{\partial p}$$

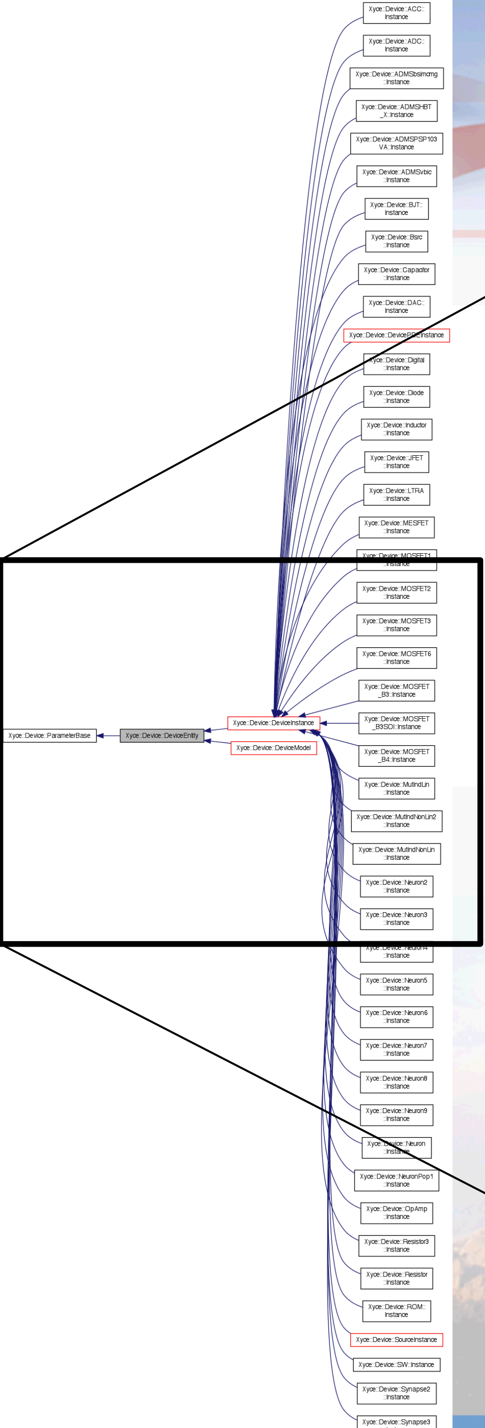
$$\left[\frac{\partial O}{\partial x}\left(\frac{\partial F}{\partial x}\right)^{-1}\right] = \left(\frac{\partial F}{\partial x}\right)^{-T}\frac{\partial O}{\partial x}$$

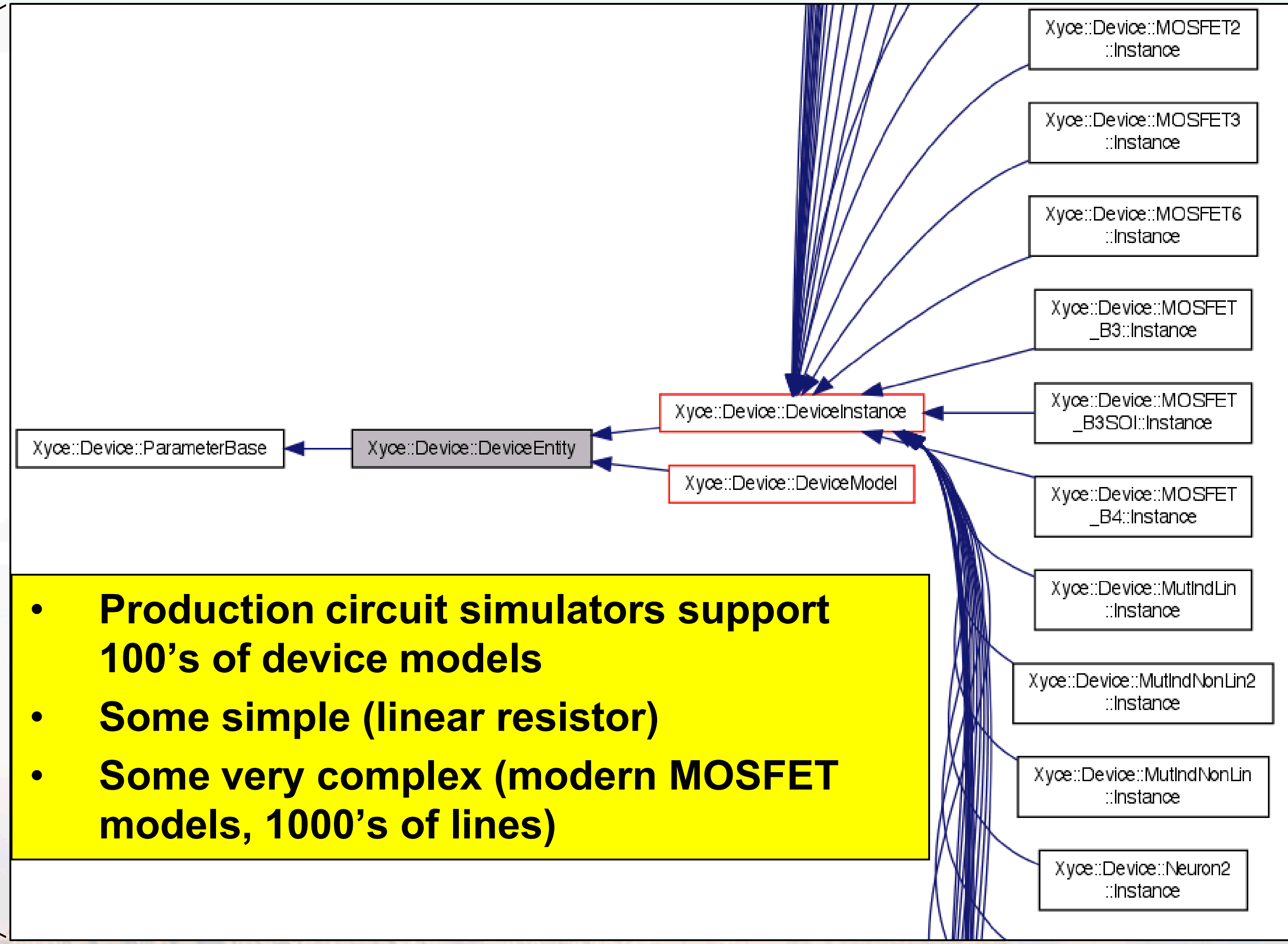
$$\left(\frac{\partial F}{\partial x}\right)^{T}\theta^{*} = \frac{\partial O}{\partial x}$$

$$\theta^{*} = \frac{\partial O}{\partial F}$$

O=objective function (scalar)
p=parameter (scalar)
F=residual vector
X=solution vector

Xyce devices





- **Production circuit simulators support 100’s of device models**
- **Some simple (linear resistor)**
- **Some very complex (modern MOSFET models, 1000’s of lines)**

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t), \vec{u}(t)) = \vec{0}$$

ADMS-Xyce

Development Status and Plans

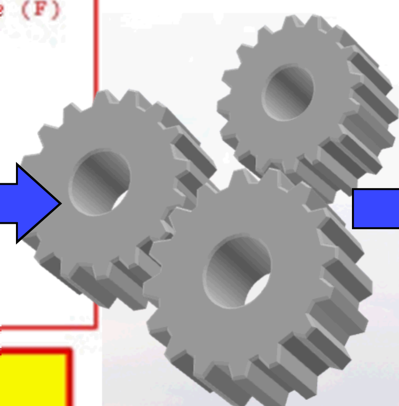
```
// Series RLC
// Version 1a, 1 June 04
// Ken Kundert
//
// Downloaded from The Designer's Guide Community
// (www.designers-guide.org).
// Taken from "The Designer's Guide to Verilog-AMS"
// by Kundert & Zinke. Chapter 3, Listing 14.

`include "disciplines.vams"

module series_rlc2 (p, n);
    parameter real r=1000;      // resistance (Ohms)
    parameter real l=1e-9;      // inductance (H)
    parameter real c=1e-6;      // capacitance (F)
    inout p, n;
    electrical p, n, i;
    branch (p, i) rl, (i, n) cap;

    analog begin
    V(rl) <+ r*I(rl);
    V(rl) <+ ddt(l*I(rl));
    I(cap) <+ ddt(c*V(cap));
    end
endmodule
```

Verilog-A



Planned

NextGen (ATDM)
Kokkos

Embedded UQ
Stokhos

$$f(\theta) = \sum_{k=0}^{\infty} F_{pk}\Phi_k(\epsilon)$$

Implemented

Dynamic Linking
`*.so` files

Param Sensitivities
Sacado

$$\frac{d\vec{f}(\vec{x})}{dp}, \frac{d\vec{q}(\vec{x})}{dp}$$

DAE support
***Sacado* (Jacobian)**

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t), \vec{u}(t)) = \vec{0}$$

Verilog-A (high level modeling language) Input

Run admsXyce

Ready-to-compile C++ code