

Enabling Next-Generation Parallel Circuit Simulation with Trilinos

Chris Baker¹, Erik Boman², Mike Heroux², Eric Keiter², Siva Rajamanickam²,
Rich Schiek², and Heidi Thornquist²

¹ Oak Ridge National Laboratory, Oak Ridge, TN 37831, USA

² Sandia National Laboratories, Albuquerque, NM 87185, USA **

Abstract. The Xyce Parallel Circuit Simulator, which has demonstrated scalable circuit simulation on hundreds of processors, heavily leverages the high-performance scientific libraries provided by Trilinos. With the move towards multi-core CPUs and GPU technology, retaining this scalability on future parallel architectures will be a challenge. This paper will discuss how Trilinos is an enabling technology that will optimize the trade-off between effort and impact for application codes, like Xyce, in their transition to becoming next-generation simulation tools.

Keywords: circuit simulation, parallel computing, hybrid computing, preconditioned iterative methods, load balancing

1 Motivation

Traditional analog circuit simulation, originally made popular by the Berkeley SPICE program [1], does not scale well beyond tens of thousands of devices, due to the use of sparse direct matrix solvers [2]. Given the importance of this simulation tool for circuit design verification, many attempts have been made to allow for faster, larger-scale circuit simulation. Fast-SPICE tools use event-driven simulation techniques and lookup tables for precomputed device evaluations, while hierarchical simulators use circuit-level partitioning algorithms [4, 5] and more efficient data structures to enable the simulation of much larger problems. Unfortunately, the approximations inherent to these simulation approaches can break down under some circumstances, rendering such tools unreliable.

The availability of inexpensive clusters, multi-core CPUs, and GPUs, has resulted in significant interest for efficient parallel circuit simulation. Several approaches have been investigated for enabling parallel SPICE-accurate simulation. These generally involve a higher-level partitioning of the devices [6] or lower-level partitioning of the linear system of equations [7] to facilitate the creation of a more efficient parallel matrix solver. Recently, GPUs have been used

** Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

to accelerate transistor model evaluation [8], which can dominate time-domain circuit simulation. While these are examples of targeted improvements to parallel performance, efficient circuit simulation on next-generation architectures will require exploiting both coarse and fine-grained parallelism throughout the entire simulation flow.

Xyce [9], is a simulator designed “from-the-ground-up” to be distributed memory-parallel, and is intended for a spectrum of parallel platforms, from high-end supercomputers to multi-core desktops. It relies primarily upon a message-passing implementation (MPI) [10], but employs software abstractions that allow the simulator to adapt to other parallel paradigms. Xyce already leverages many of the high-performance scientific libraries provided by Trilinos [11] for its MPI-based implementation. This paper will present the enabling technologies provided by Trilinos that will allow Xyce to retain scalable performance on next-generation architectures.

2 Background

Circuit simulation adheres to a general flow, as shown in Fig. 1. The circuit, described by a netlist file, is transformed via modified nodal analysis (MNA) into a set of nonlinear differential algebraic equations (DAEs)

$$\frac{dq(x(t))}{dt} + f(x(t)) = b(t), \quad (1)$$

where $x(t) \in \mathbb{R}^N$ is the vector of circuit unknowns, q and f are functions representing the dynamic and static circuit elements (respectively), and $b(t) \in \mathbb{R}^M$ is the input vector. For any analysis type, the initial starting point is this set of DAEs. The numerical approach employed to compute solutions to equation (1) is predicated by the analysis type.

Transient and DC analysis are two commonly used simulation modes, in which the set of equations (1), more generally expressed as $F(x, x') = 0$, is solved by numerical integration methods corresponding to the nested solver loop in Fig. 1. Both analysis types require the solution to a sequence of nonlinear equations, $F(x) = 0$. Typically, Newton’s method is used to solve these nonlinear equations, resulting in a sequence of linear systems

$$Ax = b$$

that involve the conductance, $G(t) = \frac{df}{dx}(x(t))$, and capacitance, $C(t) = \frac{dq}{dx}(x(t))$, matrices. For DC analysis, the q terms are not present in equation (1), so the linear system only involves the conductance matrix.

The computational expense in circuit simulation is dominated by repeatedly solving linear systems of equations, which are at the center of the nested solver loop (Fig. 1). Solving these linear systems requires their assembly, which depends upon device evaluations for the whole circuit. This means the computational expense includes both the device evaluations and the numerical method used to solve the linear systems. The linear systems solved during transient and DC

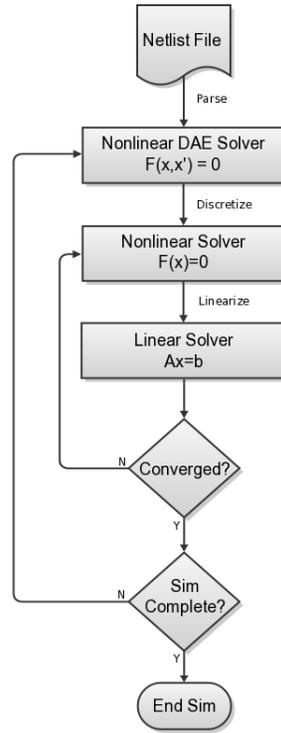


Fig. 1. General circuit simulation flow

analysis are typically sparse, have heterogeneous non-symmetric structure, and are often ill-conditioned. Direct sparse solvers [2,3] are the industry standard approach because of their reliability and ease of use. However, direct solvers scale poorly with problem size and become impractical when the linear system has hundreds of thousands of unknowns or more.

3 Xyce-Trilinos Interface

Xyce is written in ANSI C++ and exploits modern software paradigms to enable the development of a production simulator as well as a testbed for parallel algorithm research. Xyce uses abstract interfaces and runtime polymorphism throughout the code, which facilitates code reuse and algorithmic flexibility. Many of the higher-level abstractions, relating to the analysis type or time integration methods, have implementations that are contained in Xyce. However, the lower-level numerical abstractions, related to the nested solver loop in Fig. 1, have interfaces to the high-performance scientific libraries provided by Trilinos. The current Trilinos software stack that is employed by Xyce is illustrated in Fig. 2.

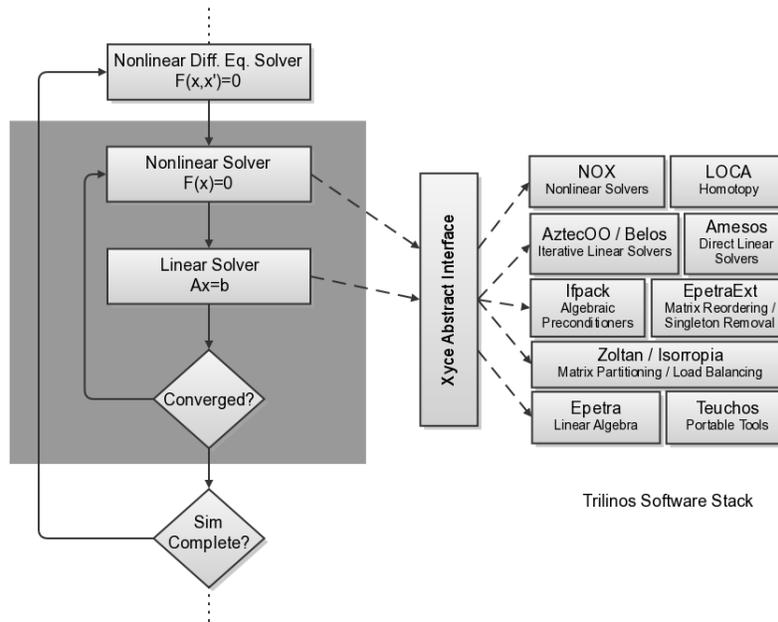


Fig. 2. Nested solver loop interface to Trilinos

Xyce employs these software abstractions to enable adaptation to future parallel paradigms and arithmetic precision strategies with minimal effort. Xyce currently uses MPI with double precision arithmetic through essential Petra (Epetra). However, future computational platforms may require the use of other parallel paradigms, such as hybrid techniques that combine MPI with threads, to achieve optimal performance. Furthermore, to address ill-conditioned matrices, it may prove useful to use higher precision arithmetic in the linear solvers. These next-generation computational strategies are the motivation for the newer Trilinos linear algebra packages: templated Petra (Tpetra) and Kokkos. Tpetra provides a templated interface to parallel linear algebra and Kokkos contains the underlying computational kernels enabling platform-dependent optimizations. Several pre-existing Trilinos packages can use Tpetra, like NOX, LOCA, Belos, and Teuchos, and many other packages are under development to provide direct solvers and preconditioners using Tpetra.

4 Next-Generation Circuit Simulation

Parallelism can be integrated into every step of the nested solver loop shown in Fig. 1. Furthermore, parallelism can be achieved through both coarse-scale (multi-processor) and fine-scale (multi-threaded) approaches. A composition of these two approaches will provide circuit simulation with the best performance impact on the widest variety of next-generation architectures. As discussed before, the majority of the computational time is spent in device evaluations and linear solvers, so this paper will present some of Trilinos' capabilities that will improve the parallelism pertaining to those specific tasks. In particular, the focus is on four packages that will enable more efficient circuit simulation in the mi-

gration to next-generation architectures: Epetra, ShyLU, and Zoltan. Numerical results will be presented for Epetra and ShyLU.

4.1 Computational Setup

Results presented in this section are generated by using Xyce (post release 5.2.1) and Trilinos release 10.8. The simulations are performed on a single node of a small cluster, where each node has a dual-socket/quad-core Intel Xeon® E5520 2.67 GHz processor and 36 GB of memory. Xyce and Trilinos are compiled using Intel 11.1 compilers, and the MPI library is supplied by OpenMPI version 1.3.3. Table 1 partially describes the circuits used in the numerical experiments. Two of the circuits, ckt2 and ckt3, are from the freely available and well known test suite CircuitSim90 [12], and respectively correspond to the chip2 and ram2k test cases. The other three circuits are proprietary integrated circuits.

Table 1. Circuits: matrix size(N), capacitors(C), MOSFETs(M), resistors(R), voltage sources(V), diodes (D).

Circuit	N	C	M	R	V	D
ckt1	63761	208236	11732	51947	56	0
ckt2	46850	21548	18816	0	21	0
ckt3	32632	156	13880	0	23	0
ckt4	25187	0	71097	0	264	0
ckt5	15622	7507	10173	11057	29	0

4.2 Epetra

Xyce currently uses Epetra underneath its abstract interface to provide serial and distributed parallel linear algebra objects (Fig. 2). As of Trilinos release 10.4, several of the linear algebra objects that Xyce interfaces to provide multi-threading capabilities via OpenMP to speed up basic computations. The classes that have been decorated with “parallel for” pragmas are `Epetra_Vector`, `Epetra_MultiVector`, `Epetra_CrsMatrix`, and `Epetra_CrsGraph`. Given that Xyce uses Epetra by default, it requires no code modifications to evaluate these hybrid (MPI with OpenMP) linear algebra computations. However, for Xyce, iterative linear solvers exercise these computations the most, which makes them necessary for illustrating the potential performance improvements.

For these numerical experiments, a preconditioned iterative solution strategy is employed that consists of several steps including the removal of dense rows or columns (singleton filtering), block triangular form (BTF) reordering, and hypergraph partitioning [13] to generate a block Jacobi preconditioner. This preconditioner, combined with AztecOO’s Generalized Minimal Residual (GMRES) method, has been shown to speed up the simulation time for ckt2, ckt3, ckt4 and ckt5 [14]. For ckt1, this technique results in a large irreducible block, making this preconditioner inefficient, so ckt1 will not be considered for this test.

Table 2. Comparison of Xyce simulation times using a non-threaded build of Epetra (MPI only; 2 MPI processes) versus a hybrid build of Epetra (MPI w/ OpenMP; 2 MPI processes, 2 threads per process).

Circuit	Linear Solver (sec.)			Total Simulation (sec.)		
	MPI only	MPI w/OpenMP	x Speedup	MPI only	MPI w/OpenMP	x Speedup
ckt2	92.8	66.1	1.40	165.9	143.3	1.15
ckt3	246.7	101.2	2.43	351.0	198.4	1.76
ckt4	36.2	23.4	1.54	186.5	157.3	1.18
ckt5	92.9	46.3	2.00	239.5	181.3	1.32

The results from performing a full transient simulation of these four circuits using Xyce compiled against a non-threaded (MPI only) and a hybrid (MPI w/ OpenMP) build of Trilinos is presented in Table 2. The simulations are performed with 2 MPI processes and, additionally, 2 threads per process for the hybrid build of Trilinos. The timings are averaged from three simulations of each scenario. Table 2 shows the speedup that was achieved using the hybrid build of Trilinos for both the linear solver, as well as for the whole simulation.

As previously mentioned, the potential performance improvements are concentrated in the linear solver, so the dramatic speedups achieved in that portion of the simulation are tempered by the rest of the simulation cost. Performance of the preconditioned iterative methods used in circuit simulation can degrade with an increasing number of MPI processes, as the preconditioner becomes less effective. Using hybrid techniques enable a performance improvement by leveraging shared memory techniques for fine-grained processes, like linear algebra, while maintaining the robustness of the preconditioner by reducing the number of distributed memory processes. The results presented in Table 2 indicate a total simulation speedup between 1.15 and 1.76 by making this minor change.

4.3 ShyLU

ShyLU provides a “hybrid-hybrid” sparse linear solver framework, based on Schur complements, that incorporates both direct and iterative methods, as well as coarse-scale (multi-processor) and fine-scale (multi-threaded) parallelism. It can serve as both a standalone black-box solver for medium-sized problems, and a subdomain solver or preconditioner within a larger distributed-memory framework. ShyLU is targeted towards next-generation architectures with many CPU-like cores within a single compute node, is based on Trilinos, and is also intended to become a Trilinos package.

The Schur complement approach solves the linear system $Ax = b$, by partitioning it into

$$A = \begin{bmatrix} D & C \\ R & G \end{bmatrix}, x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}, \quad (2)$$

where D and G are square, D is non-singular, and x and b are conformally partitioned to A . The Schur complement, after elimination of the top row, is $S = G - R * D^{-1}C$. Solving $Ax = b$ then consists of the three steps:

Table 3. Comparison of average number of outer GMRES iterations using ShyLU as a preconditioner with the BTF-based and block Jacobi preconditioner. A dash indicates simulation failure.

Circuit	ShyLU	BTF	Block Jacobi
ckt1	2	-	151
ckt2	1	4	1
ckt3	1	7	60
ckt4	1	6	6
ckt5	2	9	131

1. Solve $Dz = b_1$.
2. Solve $Sx_2 = b_2 - Rz$.
3. Solve $Dx_1 = b_1 - Cx_2$.

Iterative methods, based on Schur complement techniques, have proven effective and robust enough for circuit simulation [15, 16]. In general, these approaches partition A so that the first and third steps can be performed quickly using direct methods, and the second step is performed inexactly by either approximating S or using an iterative method to solve $Sx_2 = b_2 - Rz$.

For these numerical experiments, ShyLU is used to generate a preconditioner for AztecOO’s GMRES method. The Schur complement is dense for these circuits, so an approximation $\tilde{S} \approx S$ is constructed using value-based dropping with a threshold relative to the diagonal entries (10^{-2}). The approximation \tilde{S} is used as a preconditioner to iteratively solve the Schur complement system, with a relative residual tolerance of 10^{-10} in at most 30 iterations. The initial partitioning of A is performed using the ParMETIS [18] graph partitioner through Zoltan. The simulations are performed with 2 MPI processes without any threads, since KLU [17] is used as the direct solver for the diagonal blocks of D .

The results from performing a full transient simulation of these five circuits using Xyce with different preconditioners is presented in Table 3. The average number of GMRES iterations needed to achieve convergence is reported for each of the three preconditioners: ShyLU (as described above), BTF-based preconditioning (described in Section 4.2), and block Jacobi preconditioning (KLU used to factor the diagonal blocks). While the BTF preconditioner is effective on four of the five circuits, ShyLU performs more consistently and robustly on all five circuits. Alternatively, the block Jacobi preconditioner is sporadically effective in solving these five circuits, performing well on ckt2 and ckt4, but poorly on ckt1, ckt3, and ckt5.

4.4 Zoltan

In general, circuits of interest tend to be heterogeneous in structure, so the optimal parallel load balance for device evaluation (including matrix and residual vector assembly) will likely be different than for solving the linear system. For this reason, Xyce employs a different load balance for both these phases of the simulation, as illustrated in Fig. 3. Zoltan [19] is the parallel partitioning and load

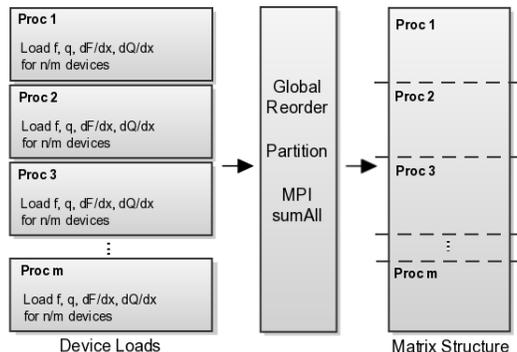


Fig. 3. Parallel load balance for device evaluation and matrix structure

balancing library that Xyce currently uses to compute an efficient distributed-memory matrix partition. However, Zoltan can also be employed to develop a device evaluation phase that more intelligently balances the fine-grained computations performed by individual devices.

Xyce currently uses a naive partitioning of devices across MPI processes (Fig. 3), which has proven to be relatively well load balanced and reasonably scalable on current architectures. During the device evaluation phase, on each processor, devices are evaluated sequentially according to type. For example, given a transmission line (RLC circuit), all the resistors will be evaluated and loaded into the matrix and residual vector, then the capacitors, and finally the inductors. These fine-scale sequential computations can be accelerated through multi-threading techniques. However, to avoid race conditions, the static circuit connectivity (graph) can be leveraged to determine in what order the devices should be evaluated. Using the partitioning algorithms in Zoltan, a device ordering can be computed for each node that optimizes the thread-parallelism, and minimizes thread conflicts, during the matrix and residual vector load.

5 Conclusion

Scientific libraries that enable scalable performance of application codes on a wide variety of next-generation architectures are essential. Trilinos is one such project that is attempting to mitigate the challenge of this transition for application codes, like Xyce. This paper presents some enabling technologies delivered through the Epetra, ShyLU, and Zoltan packages that will facilitate this transition for circuit simulation.

References

1. L.W. Nagel: SPICE 2, a Computer Program to Simulate Semiconductor Circuits, Memorandum ERL-M250, University of California, Berkeley (1975)
2. T. A. Davis: Direct Methods for Sparse Linear Systems, SIAM (2006)

3. K.S. Kundert: Sparse Matrix Techniques, Circuit Analysis, Simulation and Design (1987)
4. A.R. Newton and A.L. Sangiovanni-Vincentelli: Relaxation based electrical simulation, *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.* 4, 308–330 (1984)
5. J.K. White and A. Sangiovanni-Vincentelli: Relaxation techniques for the simulation of VLSI circuits, Kluwer Academic Publishers (1987)
6. N. Fröhlich, B.M. Riess, U. Wever, and Q. Zheng: A New Approach for Parallel Simulation of VLSI-Circuits on a Transistor Level, *IEEE Transactions on Circuits and Systems Part I*, 45, 6, 601–613 (1998)
7. H. Peng and C.K. Cheng: Parallel transistor level circuit simulation using domain decomposition methods, In *Proceedings of ASP-DAC 2009*, 397–402 (2009)
8. K. Gulati, J.F. Croix, S.P. Khatri, and R. Shastry: Fast circuit simulation on graphics processing units, In *Proceedings of ASP-DAC 2009*, 403–408 (2009)
9. E.R. Keiter, H.K. Thornquist, R.J. Hoekstra, T.V. Russo, R.L. Schiek, and E.L. Rankin: Parallel Transistor-Level Circuit Simulation, *Advanced Simulation and Verification of Electronic and Biological Systems* (2011)
10. W. Gropp, E. Lusk, N. Doss, and A. Skjellum: A high-performance, portable implementation of the MPI message passing interface standard, *Parallel Computing*, 22, 6, 789–828 (1996)
11. M.A. Heroux et al.: An Overview of the Trilinos Project, *ACM TOMS*, 31, 397–423 (2005)
12. J.A. Barby and R. Guindi: CircuitSim93: A circuit simulator benchmarking methodology case study, *Proc. of Sixth Annual IEEE International ASIC Conference and Exhibit* (1993)
13. K.D. Devine, E.G. Boman, R.T. Heaphy, R.H. Bisseling, and U.V. Catalyurek, Parallel Hypergraph Partitioning for Scientific Computing, *Proc. of 20th International Parallel and Distributed Processing Symposium* (2006)
14. H.K. Thornquist et al.: A Parallel Preconditioning Strategy for Efficient Transistor-Level Circuit Simulation, *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 410-417 (2009)
15. A. Basermann, U. Jaekel, and M. Nordhausen: Parallel iterative solvers for sparse linear systems in circuit simulation, *Fut. Gen. Comput. Sys.*, 21(8), 1275-1284 (2005)
16. C. Bomhof and H. vanderVorst: A parallel linear system solver for circuit simulation problems, *Num. Lin. Alg. Appl.*, 7, 649-665 (2000)
17. Ken Stanley and Tim Davis: KLU: a Clark Kent' sparse LU factorization algorithm for circuit matrices, *SIAM Conference on Parallel Processing for Scientific Computing* (2004)
18. G. Karypis and V. Kumar: ParMETIS: Parallel Graph Partitioning and Sparse Matrix Ordering Library, CS Dept., Univ. Minn., <http://glaros.dtc.umn.edu/gkhome/views/metis> (1997)
19. Erik Boman, Karen Devine, Robert Heaphy, Bruce Hendrickson, William F. Mitchell, Matthew St. John, and Courtenay Vaughan, Zoltan: Data-Management Services for Parallel Applications: User's Guide, Sandia National Laboratories, <http://www.cs.sandia.gov/Zoltan/Zoltan.html>, (2004)