A Hybrid Approach for Parallel Transistor-Level Full-Chip Circuit Simulation

Heidi K. Thornquist, Sivasankaran Rajamanickam

Sandia National Laboratories[⋆⋆],
Albuquerque, New Mexico.

Abstract. The computer-aided design (CAD) applications that are fundamental to the electronic design automation industry need to harness the available hardware resources to be able to perform full-chip simulation for modern technology nodes (45nm and below). We will present a hybrid (MPI+threads) approach for parallel transistor-level transient circuit simulation that achieves scalable performance for some challenging large-scale integrated circuits. This approach focuses on the computationally expensive part of the simulator: the linear system solve. Hybrid versions of two iterative linear solver strategies are presented, one takes advantage of block triangular form structure while the other uses a Schur complement technique. Results indicate up to a 27x improvement in total simulation time on 256 cores.

1 Introduction

Circuit simulation is a technique for checking and verifying the design of electrical and electronic circuits and systems prior to manufacturing and deployment. Circuit simulators use a detailed, transistor-level description of the circuit to achieve relatively accurate performance characteristics. For integrated circuit (IC) design, where probing the behavior of internal signals is extremely difficult, time-domain circuit simulation is an essential, yet expensive, part of the CAD process. Efficient, scalable simulation tools are even more important for simulation of modern technology nodes, where parasitic effects can increase the device count in an integrated circuit by an order of magnitude or more. Traditional transistor-level simulation, made popular by the Berkeley SPICE program [8], becomes impractical beyond tens of thousands of devices, due to the reliance on sparse direct linear solvers [3]. Many attempts have been made to allow for faster, large-scale circuit simulation with Fast-SPICE tools or hierarchical simulators. Unfortunately, the approximations inherent to these simulation approaches can break down under some circumstances, rendering such tools unreliable. With the transition to manycore processors, parallel transistor-level simulation has received more interest from the electronic design automation community.

[⋆⋆] Sandia is a multiprogram laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

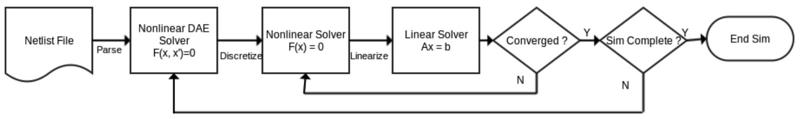

Fig. 1. General circuit simulation flow

Our contributions in this paper are: a *scalable and robust* transistor-level circuit simulation approach for large and challenging problems that uses a hybrid version of the BTF-based preconditioner [10] or a multithreaded Schur complement computation within a hybrid (direct+iterative) solver [9], integration of these linear solvers to a distributed memory parallel circuit simulator, Xyce [5], and a thorough comparison with other serial, multithreaded and distributed solvers for the full simulation. Our hybrid solvers can achieve a parallel speedup of up to *27x* when compared with fastest third party solver. A proof of concept MPI-only Schur complement solver was presented earlier in References [1, 11].

2 Xyce Framework

Xyce is a transistor-level simulator and adheres to a general flow, as shown in Fig. 1. The circuit is described by a netlist file, which lists the individual components and how they are connected together. This list of devices and interconnectivity is transformed via modified nodal analysis (MNA) into a set of nonlinear differential algebraic equations (DAEs)

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

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.

Time-domain simulation, or transient analysis, solves the nonlinear DAEs (1) implicitly through numerical integration methods, resulting in the nested solver loop in Fig. 1. Any numerical integration method requires the solution to a sequence of nonlinear equations, $F(x) = 0$. Typically, Newton's method is used to solve these nonlinear equations, which generates a sequence of linear systems

$$Ax = b \tag{2}$$

with conductance, $G(t) = \frac{df}{dx}(x(t))$, and capacitance, $C(t) = \frac{dq}{dx}(x(t))$, matrices.

The computational expense in large-scale circuit simulation is dominated by repeatedly solving linear systems of equations, which are at the center of the nested solver loop (Fig. 1). These matrices are typically sparse, non-symmetric and are often ill-conditioned. Direct sparse linear solvers [3] are the industry standard approach because of their robustness. When the linear system has hundreds of thousands of unknowns or more, direct solvers become less practical. Despite the problems inherent to circuit matrices, preconditioned iterative solvers have the potential to be a scalable solution method for large-scale linear systems. Standard preconditioners, such as multigrid and domain decomposition do not generally work well for circuit problems. It has been shown that effective preconditioners can be generated by using the block triangular form (BTF) matrix structure often found during DC analysis [10]. Unfortunately, with these preconditioning techniques, the number of iterations to solve the linear system will increase with the number of MPI processes. A hybrid approach controls the number of iterations resulting in better performance and Schur complement approaches have been shown to work well as preconditioners [1, 2].

3 Linear Solver Strategies for Circuit Simulation

This section describes a preconditioner and a Schur complement solver that is effective for circuit problems, their limitations in a distributed memory only implementation and hybrid techniques that improve scalability. For full-chip circuit simulation, where the number of devices can reach into the millions, these scalable hybrid linear solver strategies are imperative.

BTF-based Preconditioned Iterative Method: In general, a good preconditioner for the linear system (2) is inexpensive to apply and approximates the coefficient matrix A well. Unfortunately, these two properties often conflict. So, like with many applications, domain-specific structure must be leveraged to develop an effective preconditioner for circuit simulation. The motivation behind the BTF-based preconditioning technique is the observation that the conductance matrix $G(t)$ is often reducible when $t = 0$, and sometimes may be permuted to a block triangular form with small diagonal blocks [3, 10].

This linear solution strategy has several steps and, in the end, generates a block Jacobi preconditioner for the Generalized Minimal Residual (GMRES) method. The first step, *singleton removal*, removes the dense rows and columns that typically result from ideal power supplies and ground nodes, which are common to digital circuits [2]. These dense rows (or columns) correspond to singleton columns (or singleton rows) with one and only one non-zero entry. These matrix features have the potential to increase communication costs dramatically and can easily be removed from the linear system in a pre-processing (singleton rows) or post-processing (singleton columns) step.

The second step of this linear solution strategy is the permutation of the matrix to *block lower (or upper) triangular form*. This permutation is determined in two steps: first a maximum matching permutation to generate a matrix with a zero-free diagonal, and second a topological sort which finds the strongly-connected components of the associated directed graph. We leverage the fact that circuits often give many small diagonal blocks to use the BTF structure in a novel way. A condensed (block) graph is constructed by contracting all the vertices within each diagonal block into a single vertex. This results in a coarse representation of the original graph that is often much smaller.

The *matrix partitioning* is the third step in this linear solution strategy. We partition the condensed (block) graph into parts that are only loosely connected using hypergraph partitioning [4]. These three steps produce a global matrix reordering that is used to generate a block Jacobi preconditioner for GMRES.

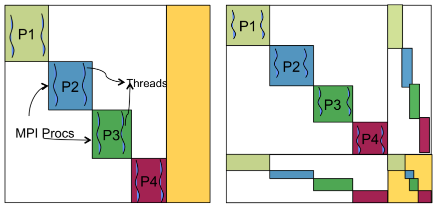

Fig. 2. Graph/Hypergraph based ordering of the sparse linear system for parallelism in ShyLU with unsymmetric ordering (left) and symmetric ordering (right).

The number of diagonal blocks in the preconditioner is the number of MPI processes used. Since a block Jacobi preconditioner only applies the inverse of these diagonal blocks, no parallel communication is required to perform the factorization and solve, which makes it a scalable preconditioning technique. However, the number of GMRES iterations needed to solve the linear system to a given tolerance will increase with the number of subdomains (MPI ranks). This effectively limits the number of MPI processes that can be used for any given problem. Therefore, it is necessary to take advantage of intra-node parallelism for accelerating local computations. We use the multithreaded kernels in the Epetra package of Trilinos for the multithreaded sparse matrix-vector multiplication.

ShyLU: ShyLU is a hybrid linear solver designed to be a black-box algebraic solver [9]. It is hybrid in both the parallel programming sense - using MPI and Threads - and in the mathematical sense - using features from direct and iterative methods. ShyLU was designed to be a subdomain solver in a domain decomposition framework within Trilinos [9]. However, it can also be used as a global Schur complement solver. We introduce hybrid parallelism in the Schur complement computation of ShyLU for it to be more scalable for large circuits.

Let $Ax = b$ be the system of interest. Suppose A has the form

$$A = \begin{pmatrix} D & C \\ R & G \end{pmatrix}, \tag{3}$$

where D and G are square and D is non-singular. The Schur complement after D is factored is $S = G - R * D^{-1} * C$. Solving $Ax = b$ then consists of solving

$$\begin{pmatrix} D & C \\ R & G \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix} \tag{4}$$

by solving $Dz = b_1$ and using the z in $Sx_2 = b_2 - Rz$ and $Dx_1 = b_1 - Cx_2$.

ShyLU uses hypergraph partitioning to permute the matrix into the bordered block diagonal form shown in Figure 2. Each block diagonal in the permuted matrix corresponds to a MPI rank and factored using a direct solver. An approximate Schur complement is used to compute a preconditioner for an iterative method to solve for the Schur complement. The approximation is computed using either dropping or a probing method for a fixed pattern. We will use the former in this paper.

The first expensive step in ShyLU is the factorization of block diagonals. For all the problems in this paper we use KLU [3] a simplicial direct solver to factor the block diagonals. The other expensive step in ShyLU is computing the approximate Schur complement. Computing the approximate Schur complement is completely local within an MPI rank in ShyLU and is a good candidate for hybrid parallelism. It involves triangular solves in $D^{-1} * C$ computation of the Schur complement and a matrix vector multiply. It can also be formulated as a matrix matrix multiply. ShyLU uses the former formulation. The triangular solve in this particular case has multiple right-hand sides where the right-hand sides are themselves sparse columns of C. We have modified KLU in order to compute the approximate Schur complement as described below.

The hybrid Schur approximation uses a triangular solve with multiple right-hand sides to compute a block column of the Schur complement in parallel. KLU's triangular solve was optimized for multiple right-hand sides using vectorization. We have introduced the multithreaded triangular solves for block columns in addition to the existing vectorization. The second change is to exploit the sparsity in the right-hand side by avoiding the additional floating point operations in the triangular solve. The importance of exploiting the sparsity in the triangular solves has been observed before [12]. However, it is also important to exploit the BTF structure in the factorization step for circuit problems. The BTF here is within the direct solver and different from BTF-based preconditioner in Section 3. As KLU uses the BTF structure in its factorization and triangular solve we exploit the sparsity within the triangular solve corresponding to the diagonal blocks of the BTF structure. Note that the numeric factorization of KLU is still sequential. In summary, we have introduced a multithreaded triangular solve with block right-hand sides that exploits sparsity in the right-hand side within the BTF structure.

4 Performance Results

This section presents results for the proposed hybrid approaches for parallel simulation of challenging problems. Results presented in this paper are generated using Xyce (post release 5.2.1) and Trilinos(10.10.1). The test machine is a capacity cluster, with 272 compute nodes, where each node has a 2.2 GHz AMD quad socket/quad core processor and 32GB RAM. Xyce, Trilinos, SuperLU v4.3 [6], and SuperLU_DIST v2.5 [7] are compiled using Intel 11.1 compilers, where the Intel MKL 11.1 provides the BLAS/LAPACK and PARDISO libraries. The integrated circuits selected for these tests are of varying scales and the simulation challenges even the sparse direct linear solvers. Table 1 partially describes the circuits used in the numerical experiments. All three of these are proprietary application-specific integrated circuits (ASICs).

Sparse Direct Solver Performance: The circuits selected for these experiments are small enough that sparse direct solvers are still practical. Therefore, we will start by looking at performance results from the state-of-the-art sparse solvers KLU, PARDISO, SuperLU, and SuperLU_DIST in Table 2. KLU and SuperLU are serial, while the parallel codes are run on 16 cores. The results illustrate the difficulty of these simulations. The only solver that consistently

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	116247	52552	69085	76079	137	0
ckt2	688838	93	222481	176	75	291761
ckt3	1944792	400234	211486	795827	36100	199992

Table 2. Total linear solve time (sec.) for various sparse direct solvers; (-) indicates simulation failed to complete. The number of threads/MPI processes is in parantheses.

	ckt1	ckt2	ckt3
KLU	9381.3	7060.8	**14222.7**
PARDISO (16)	**715.0**	**6690.5**	-
SuperLU	-	-	72176.8
SuperLU_Dist (16)	-	-	-

enables a transient simulation to complete is KLU. PARDISO performs well, beating KLU on ckt1, and ckt2, but fails to complete the DC analysis phase for ckt3. SuperLU_DIST is designed for problems with supernodal structure which is not present in any of our test cases. We believe its static pivoting choice causes the problems in completing the simulation. Note that these are representative simulations. Real simulations could be order of magnitude longer. We will compare our approaches to KLU in the rest of the paper.

Hybrid Linear Solver Performance: The linear solver dominates the simulation time for circuits ckt1 and ckt3 ($> 90\%$), while it is about half the total simulation time for ckt2. Thus, these circuits are the most useful in determining the effectiveness of the hybrid linear solvers. From past experience [10, 11] with MPI-only simulations the BTF-based preconditioner will be used for ckt1 and ckt2. ShyLU will provide a much more robust solver strategy for ckt3 , as it has a large irreducible conductance matrix. The BTF-based preconditioner is paired with the Epetra MPI+threads implementation in these experiments. For both linear solver strategies, KLU is chosen as the block diagonal solver.

A scaling study is performed using ckt1 to determine the number of MPI processes per node resulting in the best linear solver performance for the BTF-based preconditioner. The simulations are run for various numbers of MPI processes per node (ppn), 4, 8, or 16, from 1 to 8 nodes. The results in Table 3 indicate that using 4 processes per node enables the simulator to achieve a faster linear solver time than with 8 or 16 processes per node. In fact, both 8 and 16 processes per node achieve their peak linear solver performance with 16 MPI processes. The number of MPI processes is the same as the number of subdomains, as a result the preconditioner is more effective with fewer MPI ranks. The hybrid approach allows us to use the available cores effectively with fewer subdomains and a better preconditioner. For comparison the 4ppn configuration (with 32 MPI processes) results in 25% speedup over 16ppn configuration and results in a 9.5x speedup over PARDISO's time (Table 2) .

Table 3. Comparison of total linear solve time for ckt1 when the number of MPI processes per node (ppn) is varied with one thread.

MPI processes	4ppn	8ppn	16ppn
4	253.8	-	-
8	125.9	136.7	-
16	77.5	83.5	94.7
32	74.8	84.5	100.4

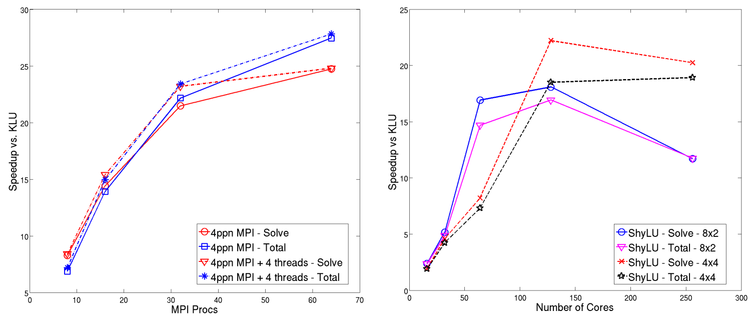

Fig. 3. Speedup of Xyce's simulation time and linear solve time for strong scaling experiments with different configurations of MPI Tasks X Threads per node using BTF (ckt2, left) and ShyLU (ckt3, right).

A larger scaling study is performed for ckt2, which generates a much larger linear system (Table 1). The strong scaling results for the linear solver time and total simulation time are presented in Fig. 3(left). They indicate that the BTF-based preconditioning technique achieves scalable performance up to 64 MPI processes, or 256 cores. At 32 MPI processes, the Epetra MPI+threads implementation provides an additional $2x$ speedup over KLU. Overall the total simulation time is $27x$ faster on 64 MPI processes for this circuit.

The largest and most challenging test case is ckt3. For single node runs, only KLU and SuperLU can finish this simulation and take 4 hours and 20 hours, respectively, for a transient time of 1 ns. Typical simulations require a transient time of 20 ns or longer, which would result in the simulation taking more than a week. The hybrid linear solver - ShyLU - is used to simulate ckt3 on up to 256 cores. The results, shown in Fig. 3 (right), indicate a significant speedup in the linear solve time, up to $22x$, and total simulation time, up to $19x$.

The importance of hybrid parallelism is illustrated in Fig. 3(right) by comparing different MPI tasks x threads per node (8x2 vs 4x4). The 4x4 configuration clearly wins in the larger core counts (128 and 256). At 256 cores, the number of MPI processes for the 8x2 configuration is 128, which is equal to the number of parts for ShyLU's partitioning (see Fig. 2). Experiments indicate that the ideal part size for ckt3 is 64. Using 128 parts results in a matrix that is imbalanced in the direct factorization phase. This results in the 8x2 case having the best performance with 128 cores (or 64 MPI processes). The 4x4 case achieves its best performance with 64 MPI processes for 256 cores. Thus, using four threads instead of two threads allowed ShyLU to speedup the simulation by an additional $4x$ (from $15x$ to $19x$) for higher core counts. An MPI-only version [11] peaks at just 64 cores because of the limited inherent parallelism in the linear problem.

5 Conclusion

This paper proposes hybrid techniques for enabling fast, parallel circuit simulation of large-scale ASICs on modern multicore platforms. These techniques are implemented in a MPI-based parallel circuit simulator, Xyce, and tested on a set of challenging integrated circuits. The results presented indicate that the hybrid linear solver strategies provide a significant improvement to Xyce's scalability. For a 500K device ASIC, the BTF-based preconditioned iterative method enables Xyce to achieve a $27x$ speedup on 256 cores. While, ShyLU, the Schur complement based hybrid linear solver, enables Xyce to achieve a $19x$ speedup on 256 cores for a 2 million device ASIC.

References

1. C. Baker, E. Boman, M. Heroux, E. Keiter, S. Rajamanickam, R. Schiek, and H. Thornquist. Enabling next-generation parallel circuit simulation with Trilinos. In *Euro-Par 2011: Parallel Processing Workshops*, volume 7155 of *LNCS*. 2012.
2. A. Basermann, U. Jaekel, M. Nordhausen, and K. Hachiya. Parallel iterative solvers for sparse linear systems in circuit simulation. *Future Gener. Comp. Sys.*, 21(8):1275–1284, January 2005.
3. T. A. Davis and E. P. Natarajan. Algorithm 907: KLU, a direct sparse solver for circuit simulation problems. *ACM Trans. Math. Softw.*, 37:36:1–36:17, Sept. 2010.
4. K.D. Devine, E.G. Boman, R.T. Heaphy, R.H. Bisseling, and Ü.V. Çatalyürek. Parallel hypergraph partitioning for scientific computing. In *Proc. of 20th International Parallel and Distributed Processing Symposium (IPDPS'06)*. IEEE, 2006.
5. E. R. Keiter, H. K. Thornquist, R. J. Hoekstra, T. V. Russo, R. L. Schiek, and E. L. Rankin. Parallel transistor-level circuit simulation. In *Advanced Simulation and Verification of Electronic and Biological Systems*, pages 1–21. Springer, 2011.
6. X. S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Softw.*, 31:302–325, September 2005.
7. X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Tran. on Math. Soft.*, 29(2):110–140, June 2003.
8. L. W. Nagel. SPICE2, a computer program to simulate semiconductor circuits. Technical Report ERL-M250, University of California, Berkeley, 1975.
9. S. Rajamanickam, E.G. Boman, and M.A. Heroux. ShyLU: A hybrid-hybrid solver for multicore platforms. In *IEEE 26th International Parallel Distributed Processing Symposium (IPDPS)*, pages 631 –643, may 2012.
10. H. K. Thornquist, E. R. Keiter, R. J. Hoekstra, D. M. Day, and E. G. Boman. A parallel preconditioning strategy for efficient transistor–level circuit simulation. In *Proceedings of the 2009 (ICCAD)*. ACM, Nov. 2009.
11. H. K. Thornquist, E. R. Keiter, and S. Rajamanickam. Electrical modeling and simulation for stockpile stewardship. *XRDS*, 19(3):18–22, March 2013.
12. I. Yamazaki and X. S. Li. On techniques to improve robustness and scalability of a parallel hybrid linear solver. In *Proc. of the 9th Int. conf. on High perf. comp. for comput. sci.*, VECPAR'10, Berlin, Heidelberg, 2011. Springer-Verlag.