SAND REPORT

SAND2010-0248
Unlimited Release
Printed January 2010

Parallel Algorithm Strategies for Circuit Simulation

Eric R. Keiter, Richard L. Schiek, and Heidi K. Thornquist

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

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.

Approved for public release; further dissemination unlimited.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/ordering.htm

SAND2010-0248
Unlimited Release
Printed January 2010

Parallel Algorithm Strategies for Circuit Simulation

Eric R. Keiter, Richard L. Schiek, and Heidi K. Thornquist,
Electrical and Microsystems Modeling

Sandia National Laboratories
P.O. Box 5800
Mail Stop 0316
Albuquerque, NM 87185-0316

Abstract

Circuit simulation tools (e.g., **SPICE**) have become invaluable in the development and design of electronic circuits. However, they have been pushed to their performance limits in addressing circuit design challenges that come from the technology drivers of smaller feature scales and higher integration. Improving the performance of circuit simulation tools through exploiting new opportunities in widely-available multi-processor architectures is a logical next step. Unfortunately, not all traditional simulation applications are inherently parallel, and quickly adapting mature application codes (even codes designed to parallel applications) to new parallel paradigms can be prohibitively difficult. In general, performance is influenced by many choices: hardware platform, runtime environment, languages and compilers used, algorithm choice and implementation, and more. In this complicated environment, the use of mini-applications small self-contained proxies for real applications is an excellent approach for rapidly exploring the parameter space of all these choices [7]. In this report we present a multi-core performance study of Xyce, a transistor-level circuit simulation tool, and describe the future development of a mini-application for circuit simulation.

This page is left intentionally blank

Contents

1 Introduction 9
1.1 State of the Field 9
1.2 Accomplishments 10
2 Parallel Approach 11
3 Results and Discussion 12
3.1 Compute Platform 13
3.2 Circuits 13
3.3 Discussion 13
4 Conclusions 16
4.1 Future Work: MiniXyce 17
References 19

This page is left intentionally blank

Figures

1 Solver structure 10
2 MPI-based parallel load/solve structure 11

Tables

1 Circuits: matrix size(N), capacitors(C), MOSFETs(M), resistors(R), inductors (L), voltage sources(V), current sources (I), expression-based sources (B), diodes (D), mutual inductors (K), BJTs (Q), controlled switches (S). 13
2 Multi-core performance results; D=#devices, N=matrix dimension, H=Xyce before performance improvements, T=Xyce after performance improvents without threading. 16

This page is left intentionally blank

1 Introduction

Circuit simulation tools (e.g., **SPICE** [14]) have become invaluable in the development and design of electronic circuits. However, they have been pushed to their performance limits in addressing circuit design challenges that come from the technology drivers of smaller feature scales and higher integration. Improving the performance of circuit simulation tools through exploiting new opportunities in multi-processor architectures is a logical next step. With the advent of multi-core technology, supercomputing has become democratized, and large-scale parallel platforms are becoming widely available.

Unfortunately, not all traditional simulation applications are inherently parallel, and quickly adapting mature application codes (even codes designed to parallel applications) to new parallel paradigms can be prohibitively difficult. Application code performance is influenced by many choices: hardware platform, runtime environment, languages and compilers used, algorithm choice and implementation, and more. However, it if often uncertain which choices have the most impact on application performance. In this complicated environment, the use of mini-applications small self-contained proxies for real applications is an excellent approach for rapidly exploring the parameter space of all these choices [7].

1.1 State of the Field

Various methods of analog circuit simulation are well described by Vlach and Singal [18], among others [1, 5, 19]. The circuit to be simulated is represented as a system of coupled DAE's, which are obtained from the enforcement of Kirchhoff's current and voltage laws (KCL and KVL, respectively) across an electrical network. The resulting system of differential algebraic equations (DAE) has the following form:

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

Simulation of this transient equation results in linear systems of the form:

$$(G + Q/\delta t)\delta x = (b - f)/\delta t \tag{2}$$

involving the conductance matrix $G(t) = \frac{df}{dx}(x(t))$, and the capacitance matrix $Q(t) = \frac{dq}{dx}(x(t))$. In general, there are several mathematical formulations that may be used to produce this DAE system, but in practice, nearly all circuit simulators such as SPICE [12], use the "modified KCL" formulation [15]. This is also the formulation used by **Xyce** [9].

The system of equations (1) is solved by implicit methods, resulting in the nested solver loop of Fig. 1. In the initial DC operating point (DCOP) problem, the q terms are not present, so equation (1) is reduced to the nonlinear equation $f(x) = 0$, and the linear system is simplified to:

$$G\ \delta x = -f(x). \tag{3}$$

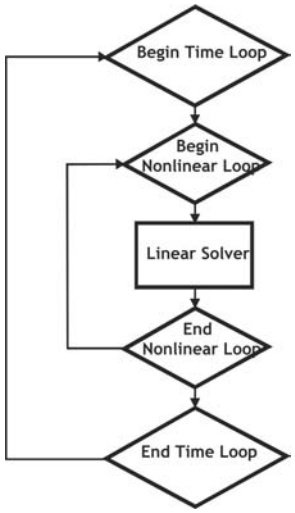

Figure 1. Solver structure

Linear systems resulting from circuit simulation are sparse, typically have heterogeneous non-symmetric structure, and are often ill-conditioned. Direct sparse linear solvers [6], [10], are the industry standard approach for solving these linear systems due to their reliability and ease of use in practice. Additionally, for smaller problems they are usually faster than iterative methods. However, direct solvers suffer from poor scaling and become impractical when the linear system has hundreds of thousands of unknowns or more.

By comparison, iterative solvers have the potential to be a scalable solution method for large-scale linear systems with a lower algorithmic complexity. The down-side is that they are not as easy to use as direct solvers because their effectiveness is dependent upon the appropriate choice of filtering, reordering, partitioning, and preconditioning.

There has been some progress on the use of iterative methods for circuit simulation, notably Basermann [2, 3] and Bomhof [4], both of whom relied on distributed Schur-complement based preconditioners. Additionally, Thornquist [17] successfully applied pre-condioners based on block-triangular factorization. Matrix-free implementations of iterative methods are commonly used in Harmonic Balance simulators. Multi-grid methods have successfully been applied to power-grid simulation by several authors [13], [16], [20]. However, for conventional transient full-system simulation, which includes functional blocks, iterative matrix solvers have yet to be widely used.

1.2 Accomplishments

The focus of the work presented in this report is on improving the performance of the device evaluation phase within Xyce. While the matrix solution phase is more ripe with opportunities for performance improvements, the device evaluation phase is more straightforward. This allows us to more quickly assess different approaches for improving performance. In

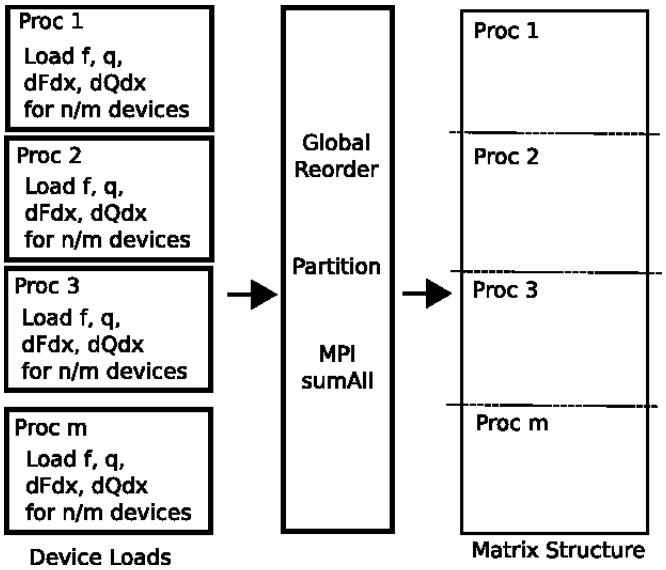

Figure 2. MPI-based parallel load/solve structure

particular, the results presented here are for cases in which multi-threading is integrated into the device load procedure.

2 Parallel Approach

Xyce is already parallelized using a message-passing programming paradigm (MPI), where the number of devices are split (almost) evenly across the number of processors (figure 2). However, each individual device evaluation is carried out serially on each processor or compute node and then loaded into a globally distributed matrix. This is an effective strategy that allows Xyce to scale well across multiple processors. However, modern desktop computers and the compute nodes of parallel computers often contain multiple CPU's per workstation or compute node. Running Xyce in parallel via MPI on such an architecture effectively loads multiple copies of Xyce which then use MPI to send messages all within a shared memory system. This reduces memory efficiency and ignores multi-threading optimizations designed into modern CPU's. Thus, we seek a more fine-grained parallelism that uses threading to better utilize multiple CPU's or cores. A good target for threading is the individual device loads.

To obtain this more fine-grained parallelism through threading, we used the OpenMP standard. Unfortunately, this type of parallelism is invasive, requiring the modification of each device with parallel directives so that it can take advantage of multi-threading. Caution needs to be taken with the insertion of directives because threading is a shared-memory programming paradigm, so multiple threads could be modifying the same place in memory. Thus it is necessary to identify and address computations where "race conditions" could occur, resulting in memory overwrites.

Our first approach consisted of having the device-loads dynamically split among the number of threads used during a given run. In this scenario, devices are divided up among the available threads in sets. As each thread finishes with a given set of devices, it requests a new set to work on. Dynamic splitting of the devices is better than static or linear division as the magnitude of calculations that must be done for various devices can differ greatly. For example, a resistor or capacitor device have minimal calculations that need to be done for a load while a modern transistor model like the BSIM3 can have thousands of calculations to do to prepare for its load.

Once a device has calculated its updates to the jacobian and residual, it must load or add the result into a memory space that is shared among all the threads. To safely do this addition, threading locks must be used to avoid thread conflicts. Augmenting the code to make such locking feasible proved difficult and slowed the resulting threaded application considerably.

An improved approach would be to duplicate the jacobian and residual for each thread, so that the threads could add to these memory spaces without first obtaining locks. When all the devices are done loading, each thread's copy of the jacobian and residual could then be summed into the final value. When running in parallel, the threading layer operates below the MPI parallel layer. Specifically, when Xyce is running in parallel the threading occurs within one of the copies of Xyce. The data to be duplicated on the thread level would only be the data that already exists on that parallel copy of Xyce. Thus, no additional parallel communication would be need to keep execution synchronized. The threading would complete before the next MPI level communication would normally occur.

Duplicating some data on the thread level avoids all the complicated locking that must be done at the cost of allocating an extra jacobian and residual vectors (e.g. F, Q and their voltage limited counterparts.). This improved approach has not yet been implemented, but is planned for Xyce. Finally, while we have studied multi-threading the device loads, we have not yet looked at threading in other areas of the codes such as in the linear solver. Since the device loads are fully within Xyce it's an area of code that we can fully modify. Linear solver technology resides in Trilinos and is thus somewhat outside the development scope of this work.

3 Results and Discussion

In this section we will present the performance results from integrating multi-threading into the device load procedure. All the simulations are performed on the compute platform described in section 3.1, using Intel 10.1 compilers. The test circuits we use to obtain these results, as described in section 3.2, have a wide variety of problem size and complexity. To fairly discern the performance gains or losses, we include performance results obtained from **Xyce** before any modifications were made to the device load procedure. Furthermore, we compare the performance with the modified version of **Xyce** that is not compiled to use any multi-threading. Lastly, we study the threading overhead by reporting the performance

Table 1. Circuits: matrix size(N), capacitors(C), MOSFETs(M), resistors(R), inductors (L), voltage sources(V), current sources (I), expression-based sources (B), diodes (D), mutual inductors (K), BJTs (Q), controlled switches (S).

Circuit	N	C	M	R	L	V	I	B	D	K	Q	S
ckt1	1191	66	875	250	20	15	9	12	225	4	57	5
ckt2	15622	7507	10173	11057	0	29	0	0	0	0	0	0
ckt3	572	236	240	342	0	12	0	0	0	0	0	0
ckt4	366	36	0	108	10	2	8	0	40	4	50	0
ckt5	25187	0	71097	0	0	264	0	0	0	0	0	0
ckt6	1220	0	432	17	0	66	0	0	0	0	0	0
ckt7	3003	3001	0	3001	0	1	0	0	0	0	0	0
ckt8	5109	0	2002	0	0	2	0	0	0	0	0	200
ckt9	1815	32	4063	4	0	10	0	0	0	0	0	0
ckt10	63761	208236	11732	51947	0	56	0	0	0	0	0	0

of the modified version of **Xyce** compiled for multi-threading, but only given a single thread at runtime.

3.1 Compute Platform

All computations are done on a 2.2 GHz AMD quad socket/quad core processor, using the OFED software stack. This processor has 32 GB DDR2 RAM and a total of sixteen cores. If only two, four, or eight of the sixteen cores are used, the memory is evenly divided between the cores.

3.2 Circuits

The experiments are based on transient simulations, and provide measure of both the efficiency and the reliability of the multi-core approach. The goal of this work is to examine how the computation work scales under different conditions, so for the purposes of testing, circuits were chosen that had a wide variety of problem size and complexity.

Table 1 partially describes the circuits used in the numerical experiments. All the circuits with ‘M’ devices use the BSIM3 [8] model, with a few exceptions. The ‘M’ devices in ckt2 are based on the SPICE level$= 1$ and level$= 3$ MOSFET models, as well as the [8] model. The ‘M’ devices in ckt11 are based on the BSIM4 [11] model. As clarification of the ‘K’ devices, the mutual inductors in ckt5 are all nonlinear and in ckt2 are both linear and nonlinear.

3.3 Discussion

The performance results for this study are found in table 2. To fairly discern the performance gains or losses, we include performance results obtained from a serial build of **Xyce** before any modifications were made to the device load procedure. These results can

be found in the first row for each circuit, and is indicated by “1(H)” in the “# Cores” column. Furthermore, we compare the performance with the modified version of **Xyce** that is not compiled to use any multi-threading. These results can be found in the second row for each circuit, and is indicated by “1(T)” in the “# Cores” column. Lastly, we study the threading overhead by reporting the performance of the modified version of **Xyce** compiled for multi-threading, but only given a single thread at runtime. These results can be found in the third row for each circuit, and is indicated by “1” in the “# Cores” column.

The performance results indicate that, in all but one circuit, using multi-threading in the device loads results in a speedup of the load time. Furthermore, in most cases the speedup increases with the number of cores. The net effect of the load speedup on the total simulation time was lessened in all cases. This can be seen by comparing the load speedup with the load and linear solve speedup. This is because the sparse linear solver used in these tests is not threaded, and thus will not scale with the number of cores. When the Jacobian and residual load time dominates the cost of the simulation, as with ckt8 and ckt9, then the multi-threaded loads result in a 2- to 3-times speedup in total simulation time. Even when the linear solver time dominates the cost of the simulation, as with ckt6, the simulation time is still decreases by 30 percent.

Given ten circuits with a wide variety of problem size and complexity, only ckt1 and ckt4 suffered from performance issues. These are the only two circuits that include diodes, BJTs, and mutual inductors, which may indicate that multi-threading is not effective in one or more of those devices.

Circuit	# Cores	Read-in and setup time	DCOP time	Transient Stepping time	Transient Linear Solve Time	# Successful Steps	# Failed Steps	# Linear Solves	# Residual Evaluations	Total Residual Load Time	Total Jacobian Load Time	Total Linear Solve Time	Total Elapsed Run Time	Speedup of Load	Speedup of Load+Solve
ckt1	1(H)														
D=1539	1(T)														
N=1191	1														
	2	1	1	5307	966	228117	28752	618683	875560	2567	1012	966	5308		
	4	1	1	4278	974	228557	28843	619418	876826	1843	739	974	4280		
	8	1	1	3875	912	228131	28715	618533	875387	1631	632	912	3876		
	16	1	1	4240	841	228409	28844	619299	876560	1955	798	842	4242		
ckt2	1(H)	13	13	288	52	364	39	1585	1988	158	58	54	314	1.00	1.00
D=28766	1(T)	13	11	232	51	359	34	1550	1943	128	35	53	255	1.24	1.23
N=15622	1	13	12	278	51	362	37	1536	1935	141	70	53	303	1.12	1.04
	2	13	10	207	51	354	41	1565	1960	87	46	54	231	1.83	1.36
	4	13	8	171	51	365	38	1580	1983	63	37	53	192	2.53	1.64
	8	13	7	143	48	351	30	1494	1875	46	29	50	163	3.45	1.93
	16	13	7	148	50	383	48	1649	2080	44	32	52	169	3.60	1.86

Circuit	# Cores	Read-in and setup time	DCOP time	Transient Stepping time	Transient Linear Solve Time	# Successful Steps	# Failed Steps	# Linear Solves	# Residual Evaluations	Total Residual Load Time	Total Jacobian Load Time	Total Linear Solve Time	Total Elapsed Run Time	Speedup of Load	Speedup of Load+Solve
ckt3	1(H)	1	0	8	2	920	76	2037	3033	4	1	2	9	1.00	1.00
D=830	1(T)	1	0	7	2	923	83	2056	3062	3	0	2	8	1.17	1.09
N=572	1	1	0	8	2	936	79	2079	3094	4	1	2	9	1.04	0.96
	2	1	0	6	2	951	80	2111	3142	2	1	2	7	1.96	1.26
	4	1	0	5	2	930	75	2056	3061	1	1	2	6	2.78	1.41
	8	1	0	5	2	933	80	2070	3083	1	1	2	6	3.02	1.40
	16	1	0	5	2	935	78	2073	3086	1	1	2	6	3.24	1.56
ckt4	1(H)	0	0	194	66	162557	17087	222850	402494	63	18	66	195	1.00	1.00
D=258	1(T)	1	0	180	67	163462	17599	224752	405813	52	13	67	181	1.21	1.08
N=366	1	0	0	208	67	163462	17599	224752	405813	68	24	67	208	0.92	0.93
	2	0	0	285	67	163884	17070	224662	405616	123	35	67	286	0.51	0.68
	4	0	0	335	68	166650	17662	228750	413062	175	28	68	335	0.36	0.58
	8	1	0	418	70	169458	17575	230615	417648	258	28	70	418	0.24	0.47
	16	0	0	553	68	168916	17702	231121	417739	401	34	68	553	0.16	0.35
ckt5	1(H)	57	366	952	262	288	17	936	1263	592	208	345	1375	1.00	1.00
D=71361	1(T)	57	346	862	268	288	17	936	1263	543	132	353	1266	1.09	1.09
N=25187	1	58	365	946	263	288	17	936	1263	538	258	346	1370	1.10	1.00
	2	58	290	706	260	288	17	934	1261	340	157	341	1054	1.74	1.30
	4	58	250	581	261	288	17	927	1254	229	107	340	889	2.59	1.55
	8	57	235	514	260	288	17	933	1260	175	78	341	806	3.39	1.70
	16	58	242	507	258	288	17	935	1262	174	74	341	807	3.41	1.70
ckt6	1(H)	1	3	2361	1565	87719	7512	194223	289454	546	85	1568	2365	1.00	1.00
D=515	1(T)	1	4	2330	1540	87995	7618	195016	290629	513	106	1543	2334	1.06	1.01
N=1220	1	1	5	2461	1529	87813	7511	194449	289773	564	199	1532	2467	0.97	0.96
	2	1	4	2111	1521	87829	7553	194570	289952	313	105	1525	2115	1.74	1.12
	4	1	4	1984	1534	87619	7533	194040	289192	206	74	1537	1989	2.65	1.19
	8	1	4	1915	1540	87716	7559	194267	289542	154	51	1544	1919	3.55	1.23
	16	1	3	1688	1341	87845	7536	194551	289932	140	48	1344	1692	3.91	1.40
ckt7	1(H)	3	0	399	57	26522	5	26526	53053	159	55	57	403	1.00	1.00
D=6003	1(T)	3	0	324	57	26522	5	26526	53053	78	34	57	328	2.04	1.23
N=3003	1	2	0	364	57	26522	5	26526	53053	92	58	57	366	1.73	1.10
	2	2	1	337	58	26522	5	26526	53053	77	50	58	339	2.07	1.19
	4	2	1	324	58	26522	5	26526	53053	75	45	58	327	2.12	1.23
	8	2	0	333	57	26522	5	26526	53053	82	48	57	335	1.94	1.20
	16	2	0	338	57	26522	5	26526	53053	94	55	57	340	1.68	1.19

Circuit	# Cores	Read-in and setup time	DCOP time	Transient Stepping time	Transient Linear Solve Time	# Successful Steps	# Failed Steps	# Linear Solves	# Residual Evaluations	Total Residual Load Time	Total Jacobian Load Time	Total Linear Solve Time	Total Elapsed Run Time	Speedup of Load	Speedup of Load+Solve
ckt8	1(H)	174	2	2497	303	15501	0	46461	61962	1736	333	303	2673	1.00	1.00
D=2204	1(T)	174	0	2321	304	15501	0	46461	61962	1625	263	304	2495	1.07	1.07
N=5109	1	174	1	2683	303	15501	0	46461	61962	1752	500	303	2858	0.99	0.94
	2	175	0	1786	296	15501	0	46461	61962	1060	299	296	1962	1.64	1.36
	4	175	0	1344	310	15501	0	46480	61981	692	208	310	1519	2.51	1.76
	8	174	0	1048	296	15501	0	46480	61981	475	148	296	1222	3.65	2.19
	16	174	0	950	296	15501	0	46480	61981	367	157	296	1124	4.73	2.38
ckt9	1(H)	3	3	381	28	3026	277	6808	10111	260	56	28	386	1.00	1.00
D=4109	1(T)	3	2	326	27	3028	278	6815	10121	230	37	28	331	1.13	1.17
N=1815	1	3	3	385	27	2973	285	6734	9992	245	82	27	391	1.06	0.99
	2	3	2	293	27	3027	286	6840	10153	172	62	28	298	1.51	1.30
	4	3	2	215	28	3002	279	6781	10062	111	42	28	219	2.33	1.77
	8	3	1	145	27	2986	264	6713	9963	62	24	27	149	4.18	2.59
	16	3	1	123	25	2984	277	6738	9999	42	23	26	127	6.11	3.05
ckt10	1(H)	549	37	231	79	74	2	177	258	104	40	92	818	1.00	1.00
D=271971	1(T)	554	32	190	80	74	2	177	258	71	26	93	776	1.45	1.05
N=63761	1	557	33	197	78	74	2	177	258	71	39	91	787	1.47	1.04
	2	562	28	162	78	74	2	177	258	44	25	91	752	2.38	1.09
	4	565	26	144	79	74	2	177	258	31	20	91	735	3.38	1.11
	8	555	25	137	78	74	2	177	258	26	17	91	717	4.05	1.14
	16	556	24	133	78	74	2	177	258	24	15	90	713	4.32	1.15

Table 2: Multi-core performance results; D=#devices, N=matrix dimension, H=Xyce before performance improvements, T=Xyce after performance improvents without threading.

4 Conclusions

In this paper we discussed parallel circuit simulation, particularly for mixed-architecture parallelism, in which combinations of threading and message-passing may be components of a optimal parallel simulation strategy. Experimental results indicate that, even if only threading parallelism is used, a significant speedup can be expected with Xyce. While the combination of threading and message-passing was not explored here, it is reasonable to assume that the coarser-grained device distribution (via MPI) will be enhanced by the finer-grained device evaluation parallelism (via threading).

The performance study presented in this paper, has been enlightening, but time consuming. Furthermore, have focused on only one piece of functionality, device loading, that is required for transistor-level circuit simulation. A full simulation performance study would be too involved for even once choice of architecture. This is the motivating factor for mini-applications, like MiniXyce, so that computational kernels pertinent to a particular application can be easily and quickly assessed.

4.1 Future Work: MiniXyce

Throughout this performance study, a considerable amount of thought has been put into creating a mini-application for transistor-level circuit simulation. Even though the mini-application, MiniXyce, has not been implemented, it is the future tool for prototyping new parallel paradigms for Xyce.

Xyce, the original code upon which MiniXyce would be based, consists of over 500,000 lines of C++ code. However, much of the source is required to support capabilities that are not needed for MiniXyce. For example, the Xyce input file (IO) parser is very complex, supporting user-defined expressions, hierarchical subcircuiting, as well as the physics (compact device) model library. For large circuit simulations, the input file itself can be so large as to exceed the memory constraints of a single processor. As a result, it is necessary for Xyce's parsing to be conducted in parallel.

In addition to the IO parser support, a large fraction of the Xyce source is devoted to the library of device models. In circuit simulation, device models are used to enforce KCL equations, by applying Ohmic relationships to discrete electrical components to branches of the circuit graph. Typical examples of such components include transistors, diodes, resistors, and capacitors. While some device models, such as the resistor, are quite simple, modern transistor models can be extremely complex. It is common for modern CMOS based transistor models to consist of over 10,000 lines of C/C++ code.

For MiniXyce, both source code burdens (IO and device models) can be avoided or mitigated. The approach taken for MiniXyce will be based on the following ideas. (1) Most large circuits that could benefit from parallel computing methods will be CMOS integrated circuits. (2) CMOS integrated circuit designs can be divided into a few general categories and/or building blocks, such as memory, PLL, ADC, DAC, power grids and multipliers. As such, only a handful of device models are necessary: resistor, capacitor, voltage source, and a simplified MOSFET model. Additionally, a traditional circuit parser is not necessary, as the connectivity structure of many building blocks can be hard-coded with repeated unit cells. Realistic circuits will have more variability than can possibly be represented with such an approach, but this should be sufficient to investigate performance and scalability.

References

[1] P. Antognetti and G. Massobrio. *Semiconductor Device Modeling with SPICE*. McGraw-Hill, 1988.

[2] A. Basermann, U. Jaekel, and K. Hachiy. Preconditioning parallel sparse iterative solvers for circuit simulation. In *Proceedings of the 8th SIAM Proceedings on Applied Linear Algebra*, Williamsburg VA, 2003.

[3] A. Basermann, U. Jaekel, and M. Nordhausen. Parallel iterative solvers for sparse linear systems in circuit simulation. 21(8):1275–1284, 2005.

[4] C. Bomhof and H. vanderVorst. A parallel linear system solver for circuit simulation problems. 7(7-8):649–665, 2000.

[5] Leon O. Chua and Pen-Min Lin. *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques*. Prentice-Hall, Englewood Cliffs, New Jersey, 1975.

[6] T. A. Davis. *Direct Methods for Sparse Linear System*. SIAM, 2006.

[7] Michael A. Heroux, Douglas W. Doerfler, Paul S. Crozier, James M. Willenbring, H. Carter Edwards, Alan Williams, Mahesh Rajan, Eric R. Keiter, Heidi K. Thornquist, and Robert W. Numrich. Improving performance via mini-applications. Technical report SAND2009-5574, Sandia National Laboratories, Albuquerque, New Mexico 87185, 2009.

[8] J.H. Huang, Z.H. Liu, M.C. Jeng, K. Hui, M. Chan, P.K. KO, and C. Hu. BSIM3 Manual. Technical report, Department of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720.

[9] Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra, Lon J. Waters, and Thomas V. Russo. Xyce parallel electronic simulator design: Mathematical formulation, version 2.0. Technical Report SAND2004-2283, Sandia National Laboratories, Albuquerque, NM, June 2004.

[10] K.S. Kundert. Sparse matrix techniques. In A.E. Ruehli, editor, *Circuit Analysis, Simulation and Design*. North - Holland, 1987.

[11] William Liu. *MOSFET Models for SPICE Simulation including BSIM3v3 and BSIM4*. John Wiley and Sons, New York, 2001.

[12] L. W. Nagel. Spice 2, a computer program to simulate semiconductor circuits. Technical Report Memorandum ERL-M250, 1975.

[13] J. N. Kozhaya S. R. Nassif and F. N. Najm. A multigrid-like technique for power grid analysis. *IEEE Transactions on Computer-aided Design*, 21:11481160, Oct. 2002.

[14] Tom Quarles. Spice3f5 users' guide. Technical report, University of California-Berkeley, Berkeley, California, 1994.

[15] C. W. Ho A. E. Ruehli and P. A. Brennan. The modified nodal approach to network analysis. *IEEE Trans. Circuits Systems*, 22:505–509, 1988.

[16] Kai Sun, Quming Zhou, Kartik Mohanram, and Danny C. Sorensen. Parallel domain decomposition for simulation of large-scale power grids. *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 54–59, November 2007.

[17] Heidi K. Thornquist, Eric R. Keiter, Robert J. Hoekstra, David M. Day, and Erik G. Boman. A parallel preconditioning strategy for efficient transistor-level circuit simulation. *International Conference on Computer-Aided Design*, pages 410–417, November 2009.

[18] Jiri Vlach and Kishore Singal. *Computer Methods for Circuit Analysis and Design*. Chapman and Hall, New York, 1994.

[19] Tor A. Fjeldly Trond Ytterdal and Michael Shur. *Introduction to Device Modeling and Circuit Simulation*. John Wiley and Sons, New York, 1988.

[20] Cheng Zhuo, Jiang Hu, Min Zhao, and Kangsheng Chen. Power grid analysis and optimization using algebraic multigrid. *IEEE Transactions on Computer-aided Design*, 27:738–751, April 2008.

This page is left intentionally blank

DISTRIBUTION:

10 MS 0316
Eric R. Keiter, 09233

1 MS 0316
Robert J. Hoekstra, 01437

1 MS 0316
Richard Schiek, 01437

1 MS 0316
Heidi Thornquist, 01437

1 MS 1320
Mike Heroux, 01416

1 MS 0123
Donna L. Chavez, 01011

1 MS 0899
Technical Library, 09536