Hierarchical Multialgorithm Parallel Circuit Simulation

Xiaoji Ye, Wei Dong, *Member, IEEE,* Peng Li, *Senior Member, IEEE,* and Sani Nassif, *Fellow, IEEE*

***Abstract*—The emergence of multicore and many-core processors has introduced new opportunities and challenges to electronic design automation research and development. While the availability of increasing parallel computing power holds new promise to address many challenges in computer-aided design (CAD), the leverage of hardware parallelism can only be possible with a new generation of parallel CAD applications. In this paper, we propose a novel hierarchical multialgorithm (MA) parallel circuit simulation approach and its multicore implementation to expedite one of the most fundamental CAD applications: transistor-level transient circuit simulation. In our parallel circuit simulation approach, we create two levels of parallelism. At the higher level of parallelism, we start multiple simulation algorithms in parallel for a given simulation task. Interalgorithm communication is established to enable simulation algorithms to exchange useful information so that they could advance faster than without doing so. At the lower level of parallelism, each algorithm within the MA framework utilizes fine-grained parallel techniques such as parallel device evaluation and parallel matrix solve to fully harness the available hardware resources. By combining the two levels of parallelism, the computing power of the multicore or many-core processor platforms can be fully utilized to achieve superlinear speedup in circuit simulation.**

***Index Terms*—Circuit simulation, parallel processing.**

I. Introduction

THE LANDSCAPE of computing is changing [1] with the ongoing shift from single-core processors to multicore processors [2]–[5] in the semiconductor industry. This change brings new opportunities and challenges to the computer-aided design (CAD) community [6]. Multicore processors are now widely accessible to designers who can have the amount of computing power that was only possible on expensive supercomputers and mainframes a decade ago [7]–[9]. However, the development of parallel CAD algorithms and applications that fully utilize the available parallel hardware computing power remains a significant challenge.

Manuscript received August 24, 2009; revised February 4, 2010 and May 12, 2010; accepted July 12, 2010. Date of current version December 17, 2010. This work was supported in part by SRC and the Texas Analog Center for Excellence, under Contract 2008-HC-1836, and NSF Career Award CCF-0747423. This paper was recommended by Associate Editor L. M. Silveira.

X. Ye and P. Li are with the Department of Electrical and Computer Engineering, Texas A&M University, College Station, TX 77843 USA (e-mail: yexiaoji@tamu.edu; pli@tamu.edu).

W. Dong is with the Texas Instruments, Dallas, TX 75243 USA (e-mail: weidong@ti.com).

S. Nassif is with the IBM Austin Research Laboratory, Austin, TX 78758 USA (e-mail: nassif@us.ibm.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2010.2067870

In this paper, we parallelize one of the most used yet expensive CAD applications: transistor-level time domain circuit simulation. Parallel circuit simulation is not a completely new topic. Prior work [10]–[12] attempted to realize parallel circuit simulation from a variety of angles. A practical way to parallelize a SPICE-like circuit simulator is to parallelize the device evaluation and matrix solving. To test the efficiency of parallel matrix solver, we have conducted our own experiments using an available parallel matrix solver [13]. In Fig. 1, we show that the runtime for factorizing the matrix indeed does not scale linearly with the number of cores on a high-end 8-core shared memory server. The two matrices used in Fig. 1 are large sparse matrices extracted from circuit simulation. The solid line represents the runtime curve of a symmetric positive definite matrix, the dotted line represents the runtime curve of an unsymmetric matrix. In fact, performance improvement saturates when the number of cores used is greater than three. Although parallel device evaluation can reduce the time spent on device evaluation, it introduces additional overhead related to thread creation, termination, and synchronization, etc. Therefore, if only parallel matrix solving and parallel device evaluation are employed in a circuit simulator, runtime speedup may stagnate once the number of processors reaches certain point. Multilevel Newton algorithm [11] and waveform relaxation algorithm [12] are based on circuit decomposition. As a result of circuit decomposition, some subcircuits can be naturally solved in parallel. Decomposition-based circuit simulation algorithms are guaranteed to converge under certain assumptions of the circuit. There has been a successful implementation of the variant of the multilevel Newton–Raphson method APLAC [14] that uses convergence aiding methods to enhance the convergence properties. Note that most prior work targets traditional supercomputers and computer clusters, and do not explore the favorable characteristics of the current multicore processors such as shared-memory-based communication scheme, reduced interprocessor communication overhead, uniformed computing power among cores, etc. Recently, a so-called waveform pipelining approach is proposed to exploit parallel computing for transient simulation on multicore platforms [15].

One key observation is that most of the existing parallel circuit simulation approaches can be viewed as *intra-algorithm parallelism*, meaning that parallel computing is only applied to expedite intermediate computational steps within a single algorithm. This choice often leads to fine grained parallel algorithms which require a significant amount of data

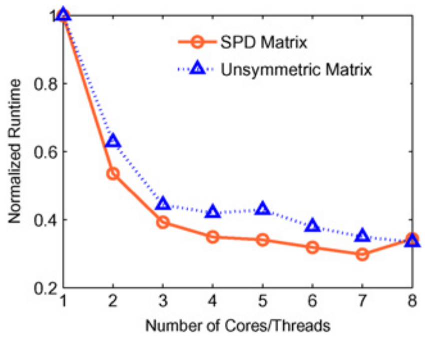


Fig. 1. Performance evaluation of a parallel matrix solver.

dependency analysis and programming effort. In this paper, we approach the problem from a somewhat unorthodox angle, we explore *interalgorithm parallelism* as well as *intra-algorithm parallelism*. This combination of different levels of parallelism not only opens up new opportunities, but also allows us to explore advantages that are simply not possible when working within one fixed algorithm.

The presented hierarchical multialgorithm parallel simulation (*HMAPS*) approach extends our earlier preliminary work [16], [17] along a similar direction. Multiple different simulation algorithms are initiated in parallel using multi-threading for a single simulation task. These algorithms are synchronized on-the-fly during the simulation. Because they have a diverse CPU-time vs. convergence property tradeoff, we pick the best performing algorithm at every time point. We include the standard SPICE-like algorithm as a solid backup solution which guarantees that the worst case performance of *HMAPS* is no worse than a serial SPICE simulation. We also include some aggressive, and possibly nonrobust, simulation algorithms that would normally not be considered in the typical single-algorithm circuit simulator. In the end, this combination of algorithms in *HMAPS* leads to favorable, sometimes, even superlinear speedup in practical cases. In addition to exploiting diversities in algorithms, the multialgorithm (MA) framework also allows algorithms to share some useful realtime information of the circuit in order to achieve better runtime performance.

Since the basic MA framework is largely independent of other parallelization techniques, we also uses more conventional approaches such as parallel device model evaluations and parallel matrix solvers to further reduce the runtime. This combination of high-level MA parallelism and low-level intra-algorithm parallelism forms the hierarchical MAPS approach which not only utilizes the hardware resources better but also achieves better runtime performance than MAPS [17].

II. Overview of the Approach

Before going into the details, we provide some observations of various parallel circuit simulation approaches. Fig. 2 illustrates four computation models and their corresponding computing platforms. The scheme on the top left is a sequential computing model where a single task/algorithm is running on a single core CPU. Traditional sequential SPICE

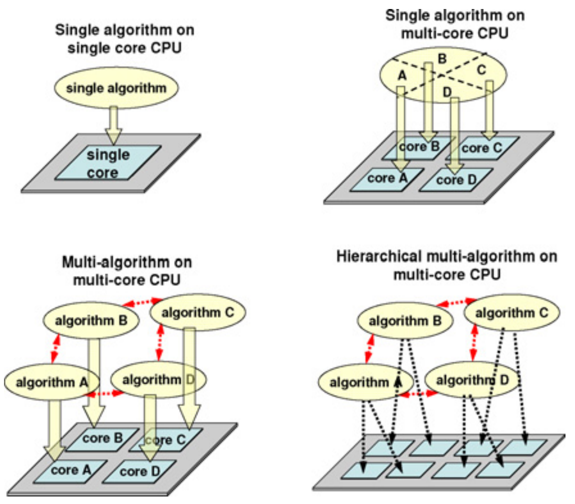


Fig. 2. Four different computing models of circuit simulation approaches.

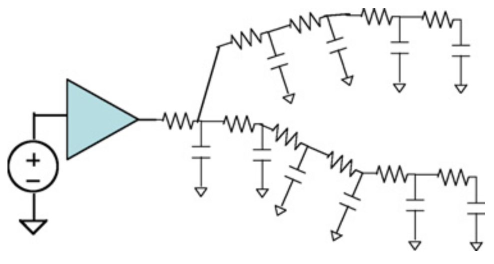


Fig. 3. Example circuit.

simulation falls into this category. The scheme on the top right is a parallel computing model where one task is divided into smaller subtasks, and each subtask is running on a core of a multicore CPU. Intra-algorithm parallel approaches such as parallel device evaluation and parallel matrix solve fall into this category. The scheme on the bottom left is another model for parallel computing where multiple algorithms are being executed for a single simulation task on multiple cores and communication is allowed between different algorithms. The MAPS approach [16], [17] belongs to this category. The fourth model is the combination of the second and third model. The hierarchical MAPS approach proposed in this paper implements this model.

A. Algorithm Diversity

In practice, a single simulation algorithm may behave differently within the entire simulation period. Take the circuit in Fig. 3 as an example. We apply a nonlinear iterative method, namely successive chord (SC) method [18], [19], to simulate the circuit. The voltage waveform at the driver output is shown in Fig. 4. During time intervals A and C, the output waveform is smooth and transistor operating conditions remain largely unchanged. Thus, as a constant-Jacobian type method, SC converges easily and moves fast during these two intervals. However, in time interval B, transistor operating conditions transit much faster. As a result, SC method is likely to converge slowly or even diverge.

The on-the-fly performance variation of a single algorithm suggests the potential benefit gained from running multiple

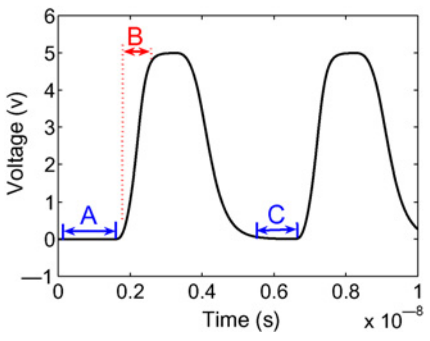


Fig. 4. Waveform at one node in a nonlinear circuit.

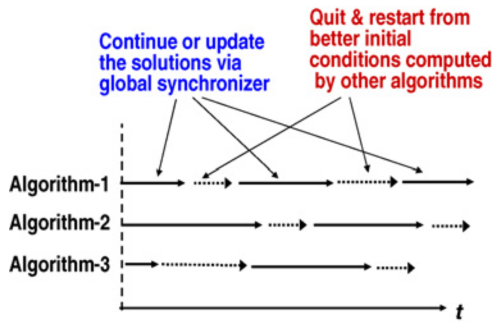

Continue or update the solutions via global synchronizer
Quit & restart from better initial conditions computed by other algorithms
Algorithm-1
Algorithm-2
Algorithm-3
t

Fig. 5. Synchronization scheme in HMAPS.

TABLE I
COMPARISONS OF MAPS AND HMAPS

Parallel Techniques	MAPS [17]	Hierarchical MAPS
Multialgorithm	Yes	Yes
Global synchronizer	Yes	Yes
Low-level matrix sharing between algorithms	No	Yes
Parallel device evaluation	No	Yes
Parallel matrix solver	No	Yes

algorithms in parallel. Ideally, a pool of algorithms of diverse characteristics are desired. In this paper, we pair various numerical integration methods with nonlinear solving methods to create a set of candidate simulation algorithms. We now consider how these algorithms should be integrated under an MA framework where algorithm diversities can be well exploited. First, consider a simple MA simulation approach where multiple algorithms are running independently in parallel and the entire simulation ends whenever the fastest simulation algorithm completes the simulation. Take the above circuit for example. Successive chord method moves fast in intervals A and C, but may be slower or even diverge in interval B. Most likely its overall runtime performance is not good due to the neutralization of its *fast* and *slow* regions. In other words, the favorable performance of SC in intervals A and C is not exploited. If other aggressive but nonrobust simulation methods encounter the same problem, which is likely in practice, the overall efficiency of the MA approach can be rather limited.

B. Synchronization and Granularity

In *HMAPS*, a flexible and conceptually clean synchronization model is adopted. Algorithms do not directly interact with each other, rather, they *asynchronously*, or *independently*, communicate with a *global synchronizer* as shown in Fig. 5. The global synchronizer stores the circuit solutions at the k most time points, where k is determined by the highest order of numerical integration formula used. With a communication granularity controlled on a individual algorithm basis, each algorithm independently updates the circuit solutions stored in the global synchronizer contingent upon the timeliness of its work, and/or loads the most updated initial condition from the synchronizer to start new work. The use of the global synchronizer provides a more transparent interface between multiple algorithms and has a clear advantage when high-order integration methods are employed, as detailed in Section VI.

The global synchronizer in the above MA approach is essentially a coarse-grained communication scheme which enables solution sharing between algorithms. Algorithm diversity is naturally exploited when the best performing algorithm at every time point writes its latest solution into the global synchronizer. Every algorithm reads the global synchronizer to get the latest solution as its initial condition for future time points. Since the write/read operations in the shared-memory-based communication scheme is easy to implement and inexpensive, we can introduce more beneficial cooperations between algorithms. For example, the selection of the fixed Jacobian matrix is the most critical point to the performance of the SC method, which is one of the simulation algorithms included under HMAPS. Since Newton's method is also used in our MA approach, and it consistently updates its Jacobian matrix, SC method can use the latest Jacobian matrix computed by Newton's method. The latest Jacobian matrix will make the SC method more likely to converge or converge in less number of iterations. This interalgorithm matrix sharing is another communication scheme besides the solution sharing in HMAPS.

Since there is a predictable trend that the number of cores of multicore processors will keep increasing, we can accommodate intra-algorithm fine-grained parallel approaches into the MA framework so that the hardware resources can be fully utilized and the performance can be further increased. Each algorithm in the MA approach uses parallel matrix solver [13] to solve the linear equation system during the nonlinear iterations and parallel device evaluation when evaluating the expensive MOSFET model for large number of transistors. This combination of high-level MA parallelism and low-level fine-grained parallel approaches forms the hierarchical MAPS. Table I compares MAPS [17] and HMAPS. Note that the parallel efficiency achieved through interalgorithm parallelisms comes at the expense of more memory consumption. That is, multiple copies of circuit data structures are needed to support the simultaneous application of multiple algorithms. For large circuits, this memory overhead justifies the consideration of the balance between interalgorithm and intra-algorithm parallelisms, as being explored in HMAPS.

III. HMAPS: Diversity in Numerical Integration

In this section, we exploit the possibility of incorporating a number of numerical integration methods with varying characteristics into the proposed HMAPS framework. A nonlinear circuit can be described by the following MNA circuit equations:

$$\frac{d}{dt}q(x) + f(x) = u(t) \tag{1}$$

where $x(t) \in R^N$ is the vector of circuit unknowns, q and f are nonlinear functions representing nonlinear dynamic and static circuit elements, $u(t) \in R^M$ is the input vector. To solve the above differential equations numerically, a numerical integration method is applied. Numerical integration methods employed in SPICE-type simulators usually include one-step methods such as backward Euler (BE), trapezoidal (TR), and multistep methods such as Gear method [20]. Additionally, variable-order variable-step methods have also been proposed to solve general ordinary differential equations (ODE) [21]. We examine the varying characteristics of these methods and outline their potential for MA simulation.

A. One-Step Integration Methods

BE and TR are one-step integration methods in that they rely on the availability of the circuit solution at one preceding time point. They are defined as , $x_{n+1} = x_n + h_{n+1}x'_{n+1}$ and $x_{n+1} = x_n + \frac{h_{n+1}}{2}(x'_n + x'_{n+1})$, respectively.

The local truncation errors (LTEs) at time t_{n+1} introduced by BE and TR are given as

$$\text{LTE}_{\text{BE}} = -h_{n+1}^2\frac{x''(\xi)}{2}, \quad \text{LTE}_{\text{TR}} = -h_{n+1}^3\frac{x'''(\xi)}{12} \tag{2}$$

where $t_n \le \xi \le t_{n+1}$. Variable time steps are used in SPICE simulators to improve the runtime efficiency [20]. And LTE can be used to predict the variable time step during the simulation. Take BE method, for example, if solutions at t_n are computed, the next time step h_{n+1} can be computed as

$$h_{n+1} = \sqrt{\frac{2\epsilon}{x''(\xi)}} \tag{3}$$

where $h_{n+1} = t_{n+1} - t_n$, ϵ is the user-defined bound for LTE, and $x''(\xi)$ is computed by the second order divided difference $DD_2(t_n)$ since solutions at t_n are available. After x_{n+1} is computed using h_{n+1}, we can again use LTE formula (2) to decide whether it should be accepted or recomputed. If LTE at t_{n+1} is within the given bound, x_{n+1} is accepted, otherwise x_{n+1} needs to be recomputed using a smaller timestep. In trapezoidal method, the same time step control mechanism may be used except that (3) should be replaced accordingly. For nonlinear circuits, slight modification of the error bound in (3) is needed in order to avoid the timestep "lock up" situation. Readers may refer to [20] for detailed explanation.

In comparison, we note that TR tends to have larger time steps than BE given the same error bound. However, TR may cause self-oscillation and for stiff circuits the timestep may need to be reduced. In some cases, numerical integration has to be switched from TR to the more robust BE to maintain stability.

B. Multistep Integration Methods

Gear methods [22] provide a different speed vs. robustness tradeoff compared to the two methods described above. It has been shown that the first and second-order Gear methods are stiffly stable, hence they do not cause self-oscillation. Gear methods are a family of multistep methods which rely on the circuit solutions at multiple preceding time points. For example, the fixed time step size second-order Gear method (Gear2) is given by $x_{n+1} = \frac{4}{3}x_n - \frac{1}{3}x_{n-1} + \frac{2}{3}h_{n+1}x'_{n+1}$.

If variable timesteps are used, the coefficients in the above formula will be decided dynamically. The variable timestep Gear2 formula is [23]

$$\begin{aligned} x_{n+1} = &-x_{n-1}\frac{h_{n+1}^2}{h_n(2h_{n+1}+h_n)} + x_n\frac{(h_{n+1}+h_n)^2}{h_n(2h_{n+1}+h_n)} \\ &+x'_{n+1}\frac{h_{n+1}(h_{n+1}+h_n)}{2h_{n+1}+h_n} \end{aligned} \tag{4}$$

where $h_{n+1} = t_{n+1} - t_n$, $h_n = t_n - t_{n-1}$. The LTE of (4) is

$$LTE_{\text{Gear2}} = -\frac{h_{n+1}^2(h_{n+1}+h_n)^2}{6(2h_{n+1}+h_n)}x'''(\xi) \tag{5}$$

where $t_n \le \xi \le t_{n+1}$. In practice, if the magnitude of LTE exceeds an upper bound, the stepsize is halved and solutions at x_{n+1} is recomputed; if the magnitude of LTE is less than a lower bound, the stepsize is doubled. The lower and upper bound have to be chosen carefully so that less solution recomputations are needed so as to maintain a desirable accuracy level [23].

C. Variable-Order Variable-Stepsize Integration Methods

It is possible to integrate even more sophisticated high order methods into HMAPS. Since only the first and second-order Gear method are stiffly stable, higher order Gear methods are not usually used in SPICE. However, there do exist other robust high order integration methods. Despite the less familiarity to the CAD community, they have gained great success in the area of numerical analysis and scientific computing. High order integration methods (order higher than two) could potentially produce large time steps. However, it is well known that high order methods are unstable for some ODEs. If a constant high order integration method, say fifth order, is used to solve a stiff system, the step size could be reduced to be very small in order to maintain stability. Hence, most of the efficient high order integration methods have certain mechanisms to dynamically vary the order as well as time step. Among these, DASSL [21] is one of the most successful ones. DASSL uses the fixed leading coefficient backward differentiation (BDF) formulas [24] to solve differential equations.

DASSL incorporates a predictor and corrector to solve an ODE system. The predictor essentially provides an initial guess for the solution and its derivative at a new time point t_{n+1}. For a kth order DASSL formula, a predictor polynomial ω_{n+1}^P is formed by interpolating solutions at the last $k+1$ time points $(t_{n-k}, \ldots, t_{n-1}, t_n)$

$$\omega_{n+1}^P(t_{n-i}) = x_{n-i} \quad i = 0, 1, \ldots, k. \tag{6}$$

The predictor of x and x' at t_{n+1} are obtained by evaluating the predictor polynomial at t_{n+1}

$$x_{n+1}^{(0)} = \omega_{n+1}^{P}(t_{n+1}) \quad x_{n+1}^{\prime(0)} = \omega_{n+1}^{\prime P}(t_{n+1}). \tag{7}$$

The predictor of x_{n+1} and x'_{n+1} are specially given by the following somewhat involved interpolation scheme:

$$x_{n+1}^{(0)} = \sum_{i=1}^{k+1} \phi_i^*(n) \quad x_{n+1}^{\prime(0)} = \sum_{i=1}^{k+1} \gamma_i(n+1)\phi_i^*(n) \tag{8}$$

where

$$\begin{aligned}
\psi_i(n+1) &= t_{n+1} - t_{n+1-i} \quad i \geq 1\\
\alpha_i(n+1) &= h_{n+1}/\psi_i(n+1) \quad i \geq 1\\
\beta_1(n+1) &= 1\\
\beta_i(n+1) &= \frac{\psi_1(n+1)\psi_2 n+1 \cdots \psi_{i-1}(n+1)}{\psi_1(n)\psi_2 n \cdots \psi_{i-1}(n)} \quad i > 1\\
\phi_1(n) &= x_n\\
\phi_i(n) &= \psi_1(n)\psi_2(n)\cdots\psi_{i-1}(n)DD(x_n, x_{n-1}, \ldots\\
&\quad , x_{n-i+1}), \qquad i > 1\\
\gamma_1(n+1) &= 0\\
\phi_i^*(n) &= \beta_i(n+1)\phi_i(n)\\
\gamma_i(n+1) &= \gamma_{i-1}(n+1) + \alpha_{i-1}(n+1)/h_{n+1} \quad i > 1
\end{aligned}$$

and $DD(x_n, x_{n-1}, \ldots, x_{n-i+1}), i > 1$ is the ith divided difference. The above intermediate variables are computed by using the solutions and time points before t_{n+1}.

The corrector polynomial ω_{n+1}^{C} is a polynomial that satisfies following conditions: first, it interpolates the predictor polynomial at k equally spaced time points before t_{n+1}

$$\omega_{n+1}^{C}(t_{n+1} - ih_{n+1}) = \omega_{n+1}^{P}(t_{n+1} - ih_{n+1}) \quad 1 \leq i \leq k \tag{9}$$

where h_{n+1} is the predicted timestep for t_{n+1}; second, the solution of the corrector formula is the solution at t_{n+1}

$$\omega_{n+1}^{C}(t_{n+1}) = x_{n+1}. \tag{10}$$

By following the above two conditions, the corrector of the kth order DASSL formula is given by

$$\alpha_s(x_{n+1} - x_{n+1}^{(0)}) + h_{n+1}(x'_{n+1} - x_{n+1}^{\prime(0)}) = 0 \tag{11}$$

where $\alpha_s = \sum_{j=1}^{k} \frac{1}{j}$.

Then (11) is solved together with (12) to get the solutions at t_{n+1}

$$F(t_{n+1}, \omega_{n+1}^{C}(t_{n+1}), \omega_{n+1}^{\prime C}(t_{n+1})) = 0. \tag{12}$$

DASSL uses LTE as a measure to control the stepsize as well as the order. It estimates what the LTEs at t_n would have been if the step to x_n were taken at orders $k-2$, $k-1$, k, and $k+1$, respectively. Based on these error estimates, DASSL decides the order k' for the next time step. If x_n is accepted, k' will be used to compute the solutions at the future time point t_{n+1}; if x_n is rejected, k' will be used to recompute x_n. Due to the page limit and topic of interest of this paper, we will not discuss the order and stepsize selection strategy of DASSL in detail. Readers may refer to [21] for the complete discussion.

The basic procedure of DASSL can be stated as:

1) calculate solutions at t_n using predicted timestep h_n, where $h_n = t_n - t_{n-1}$;
2) based on the error estimates at t_n, decide order k' for the next step;
3) based on LTE at t_n, decide whether x_n should be accepted or recomputed;
4) predict the next timestep: h_{n+1} if x_n is accepted; a new h_n if x_n is recomputed.

D. Exploiting Diversity in Numerical Integration

From the foregoing discussion, it is evident that numerical integration methods vary in complexity, speed, and robustness. The one-step first-order BE method, is robust, but has large LTEs. Variable-order variable-step size methods (e.g., DASSL) have much smaller LTEs and potentially lead to much lager time steps. However, they are significantly more complex and require numerous additional computations and checks to maintain accuracy and stability. For stiff circuits or stiff periods of the simulation, variable-order methods may decrease their orders to first order in order to ensure stability after attempting to stay at higher orders. Under these cases, a large amount of computed work may be rejected and wasted.

On the other hand, in practice it is difficult to choose a single optimal numerical integration method for a simulation task *a priori*. The efficiency of a method is decided by the nature of the circuits and input stimulus, and it varies over the time as the circuit passes through various regimes. To this end, HMAPS favorably allows for on-the-fly interaction of integration methods with varying order and time step control, at a controllable communication granularity, so as to achieve the optimal results via collaborative effort dynamically. In particular, as will be seen in Section VI, our algorithm synchronization scheme is completely transparent regardless of the choice of numerical integration and allows for the integration of arbitrary numerical integration methods.

IV. HMAPS: Diversity in Nonlinear Iterative Methods

The nonlinear iterative methods are essential to nonlinear (e.g., transistor) circuit analysis. Besides the standard Newton–Raphson method, a variety of other choices exist, providing orthogonal algorithm diversity to numerical integration algorithms that can be exploited in HMAPS.

A. Newton's Method

The widely used Newton's method solves a set of nonlinear circuit equations $\mathbf{F}(\mathbf{v}) = \mathbf{0}$ iteratively as follows:

$$\mathbf{J}^{(\mathbf{k})}\Delta\mathbf{v}^{(\mathbf{k})} = -\mathbf{F}(\mathbf{v}^{(\mathbf{k})}) \tag{13}$$

$$\mathbf{v}^{(\mathbf{k+1})} = \mathbf{v}^{(\mathbf{k})} + \Delta\mathbf{v}^{(\mathbf{k})} \tag{14}$$

where at the kth iteration, $\mathbf{J}^{(\mathbf{k})}$ is the *Jacobian matrix* of $\mathbf{F}$, which needs to be updated at every iteration; $\Delta\mathbf{v}^{(\mathbf{k})}$ is the solution increment; $\mathbf{v}^{(\mathbf{k})}$ and $\mathbf{v}^{(\mathbf{k+1})}$ are the solution guesses at the kth and $(k+1)$th iterations, respectively. Despite its good robustness, Newton's method tends to be expensive. At each iteration, a new Jacobian matrix $\mathbf{J^{(k)}}$ is assembled, which requires expensive computation of device model derivatives. Note that derivative computation is much more expensive than the evaluation of device equations and dominates the overall device model evaluation. Moreover, at each iteration a new matrix solve is required to factorize the updated Jacobian matrix $\mathbf{J^{(k)}}$, which is expensive, especially for large circuits.

B. Successive Chord Method

Different from Newton's method, SC method is a constant Jacobian matrix type iterative method [25]. Since a fixed Jacobian matrix, $J_{sc} \in \mathbf{R}^{N\times N}$, is constructed only once and then used throughout the simulation, no device model derivatives need to be computed to update the Jacobian matrix during each nonlinear iteration. As a result, there is a significant reduction in device model evaluation. Additionally, the fixed Jacobian matrix J_{sc} is only factorized once and the lower upper triangular (LU) factors can be reused to solve (13) efficiently. However, the downside of not updating the Jacobian matrix is that the convergence rate of SC method is linear, which is inferior to the quadratic convergence rate of Newton's method. The selection of chord values (entries in the Jacobian matrix corresponding to transistors) is very critical to the performance of SC method. Bad chord selection may lead to excessive number of iterations or even divergence. The convergence criterion of SC method [25] is

$$\left\|\mathbf{I} - \mathbf{J}_{\text{sc}}^{-1}\mathbf{J_F}(\mathbf{v}^*)\right\| \leq 1 \tag{15}$$

where $\mathbf{I}$ is the $N \times N$ identity matrix, $\mathbf{J}_{\text{sc}}$ is the constant Jacobian matrix used in SC method, $\mathbf{J_F}(\mathbf{v}^*) \in \mathbf{R}^{N\times N}$ is the exact Jacobian matrix at solution $\mathbf{v}^*$, of $\mathbf{F}(\mathbf{v}) = \mathbf{0}$.

C. Secant Method

In principle, secant method provides a different efficiency vs. complexity tradeoff compared with the above two methods. Secant method does form a new Jacobian matrix at each iteration, but does so approximately. The Jacobian matrix at the kth iteration is approximated by A_k in (17)

$$\mathbf{A_0} = \mathbf{J}(\mathbf{v^{(0)}}) \tag{16}$$

$$\mathbf{A_k} = \mathbf{A_{k-1}} + \frac{1}{\vec{\mathbf{S}}^{\mathbf{T}}\vec{\mathbf{S}}}(\vec{\mathbf{Y}} - \mathbf{A_{k-1}}\vec{\mathbf{S}})\vec{\mathbf{S}}^{\mathbf{T}} \tag{17}$$

where $\vec{\mathbf{S}} = \mathbf{v^{(k)}} - \mathbf{v^{(k-1)}}$ and $\vec{\mathbf{Y}} = \mathbf{F}(\mathbf{v^{(k)}}) - \mathbf{F}(\mathbf{v^{(k-1)}})$, $\mathbf{v^{(k)}}$ and $\mathbf{v^{(k+1)}}$ are the solution guesses at the kth and $(k+1)$th iterations. Secant method also avoids the need for device model derivative computation. However, a new factorization of $\mathbf{A_k}$ is still needed at every iteration. Secant method has a superlinear convergence rate which is also inferior to the quadratic convergence rate of Newton's method.

D. Exploiting Diversity in Nonlinear Iterative Methods

Although other types of nonlinear iterative methods (e.g., nonlinear relaxation methods) can also be considered, the three methods discussed above already show distinguishing tradeoffs between per iteration cost vs. number of iterations, and efficiency vs. robustness. Newton's method has the highest per-iteration cost: computation of device model derivatives and solve of a new linear system. However, it has the favorable quadratic convergence rate, which helps reduce the total number of iterations required for convergence. At each iteration, secant method relaxes the need for device model derivatives, but it only has a superlinear convergence rate. Successive chord has the lowest per-iteration cost as it relaxes the need for both the device model derivatives computation and factorization of a new Jacobian matrix. However, it has a linear convergence rate that corresponds to a larger number of iterations required to reach convergence. When the chord values are not chosen properly, SC may not even be able to converge. In terms of robustness, Newton's method is the most robust while SC is the least robust.

Again, in practice it is difficult to choose a single optimal nonlinear iterative method *a priori*. The relative performance of a method is determined by a complex tradeoff between all the above factors in addition to the dependency on the circuit type, mode of the circuit, and input excitations applied. In addition, different method is likely to prevail during different phases of a transient simulation. HMAPS allows for a dynamic exploration of superior performances of multiple nonlinear iterative methods occurring in different phases of the simulation, contributing to the overall efficiency of the MA approach. We further emphasize the following key points. Being applied as a standard alone method, the weak convergence property of a nonrobust iterative method can significantly constrain its application [18], [19]. For example, in SC method it is difficult to find near optimal chord values that achieve good efficiency while guaranteeing the convergence for the entire simulation. As a results, nonrobust methods are usually discarded for general robust circuit simulation. In HMAPS, since the standard Newton's method is always chosen as a solid backup, other nonrobust methods no longer have to converge during the entire simulation, significantly relaxing their convergence constraints. Moreover, nonrobust methods are employed with a rather *different* objective in HMAPS: they are purposely controlled in an *aggressive* or *risky* way to possibly gain large runtime speedups during certain phases of the simulation. This unique *opportunism* contributes to possible superlinear runtime speedup of the parallel MA framework.

In the current implementation of HMAPS, various numerical integration methods are paired with nonlinear iterative methods to create a pool of simulation algorithms. In terms of numerical integration methods, BE, Gear2 and our implementation of DASSL are included. In terms of nonlinear iterative methods, Newton's method is chosen as a solid backup and SC is included to gain opportunistic speedup. It is experimentally found that secant method has weak convergence property and still requires factorizing a new approximated Jacobian matrix at each iteration. Since it does not provide significant runtime benefit, secant method is currently not adopted in HMAPS. The three numerical integration methods with independent dynamic time step control are all paired with Newton's method to form three complete simulation algorithms. Hence, the SPICE-like BE + Newton combination is selected, which provides a basic guarantee for the success of the simulation.

BE is paired with SC to create the fourth algorithm. To further enhance the runtime benefit of SC in transient

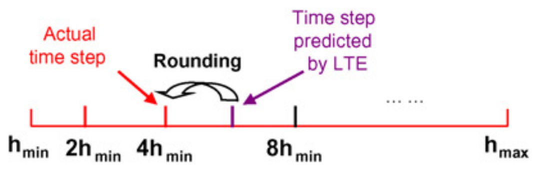


Fig. 6. Dynamic time step rounding.

simulation, a dynamic time step rounding technique [26] is used. The use of a constant Jacobian matrix in SC method reduces the number of matrix factorizations to one for the complete nonlinear solve at each time point. Note that the exact Jacobian matrix also depends on the time step in numerical integration method. For example, in BE, a grounded capacitor of value c contributes a stamp c/h to the Jacobian matrix, where h is the time step. As h is dynamically changed according to dynamic time step control, the Jacobian matrix varies over the time. To avoid frequent Jacobian matrix factorizations along the entire time axis, a set of fixed Jacobian matrices are pre-factorized before the simulation starts at a few geometrically spaced time steps $\{h_{\min}, 2h_{\min}, 4h_{\min}, \ldots, h_{\max}\}$, where $h_{\min}$ and $h_{\max}$ are estimated min/max time steps computed by dynamic time step control [26]. The total number of discrete time steps is given by

$$1 + \lceil \log_2(h_{\max}/h_{\min}) \rceil. \tag{18}$$

In this case, only 10 discrete time points are needed to cover a $1000X$ span of time step. As a result, only a limited number of Jacobian matrix factorizations are needed. During the simulation, the variable time step that is predicted by the LTE is always rounded down to the nearest smaller value in the predefined time step set as shown in Fig. 6. In this way, a pre-factorized Jacobian matrix is reused and the LTE is always satisfactory. Ideally, the time step reduction caused by rounding is no more than $2X$ because those predefined time steps are geometrically spaced.

V. Intra-Algorithm Parallelism

As mentioned earlier, device evaluation and matrix solve can be parallelized. Therefore, we incorporate the conventional intra-algorithm parallel simulation techniques into the MA framework. There are a number of reasons for this addition: first, interalgorithm approach is completely orthogonal to intra-algorithm methods, which means they can be used together without problem. Second, intra-algorithm parallel simulation algorithms can improve the efficiency of each individual algorithm in the MA framework; thus improve the overall speedup of the MA simulation. Third, the combination of interalgorithm and intra-algorithm parallelism creates more parallelism and is capable of utilizing more cores than interalgorithm parallelism alone. By combining the MA framework and intra-algorithm parallel simulation techniques, we form the complete HMAPS approach.

In parallel device evaluation, the cost of evaluating circuit elements is divided equally among cores used. In HMAPS, each simulation algorithm uses multiple threads to do parallel device evaluation. The total number of threads used in parallel device evaluation by all algorithms does not exceed the number of cores on the machine.

In HMAPS, we use parallel matrix solver SuperLU [13] to solve the system of linear equations. SuperLU is a general purpose parallel matrix solver with parallel computing capability. It has three versions: SuperLU for sequential machines; SuperLU_MT for shared memory parallel machines; SuperLU_DIST for distributed memory. Based on our thread-based HMAPS implementation, we choose SuperLU_MT for parallel matrix solve. Although SuperLU_MT is not specifically built for circuit simulation, it is sufficient for the purpose of verifying our proposed ideas and algorithms. We can easily incorporate any new parallel matrix solver in our simulation framework. In HMAPS, SuperLU needs to be used with extreme caution since global and static variables may cause false data sharing between different simulation algorithms when multiple algorithms are calling the parallel matrix routines simultaneously. Again, the total number of threads used in parallel matrix solve by all algorithms does not exceed the number of cores on the machine.

A. *Flexibility in Intra-Algorithm Parallelism*

An interesting problem in parallel computing is load balancing and resource allocation. Parallel algorithm designers face challenge of optimally assigning hardware resources to different tasks in a parallel algorithm. In HMAPS, we need to allocate the cores to each algorithm optimally to achieve good results. We favor the more effective algorithms when allocating the cores. We follow the experimental observations to decide which algorithm is likely to contribute more in HMAPS if given more cores for its low-level parallelism. This algorithm will be given more cores than other algorithms in HMAPS in the hope that this specific core allocation will result in a overall better performance of HMAPS. Right now, the core allocation in HMAPS is done manually. Automatic core assignment is possible if we can predict the performance of HMAPS with any core assignment. However, this would require us to build the performance model of HMAPS. This is an interesting future research direction.

B. *Tradeoff Between Interalgorithm and Intra-Algorithm Parallelism*

Since the hardware resources (e.g., number of cores, memory) on multicore processor computers are limited, tradeoff between interalgorithm and intra-algorithm parallelism exists. In thread-based implementation of HMAPS, each algorithm is initiated by one thread and it has to use it own private data structure including device list, matrices, device models, etc. Otherwise, false data sharing will happen. Therefore, the memory usage of the 4-algorithm HMAPS is larger than a sequential algorithm. For circuits of very large size, memory could become a bottleneck. In this case, it would be better to use a lower number of algorithms in HMAPS and assign more cores to each algorithm. On the other hand, if the circuit size is small, intra-algorithm parallelism may not be as beneficial and memory storage is not a limiting factor, more emphasis can be put on the interalgorithm parallelism.

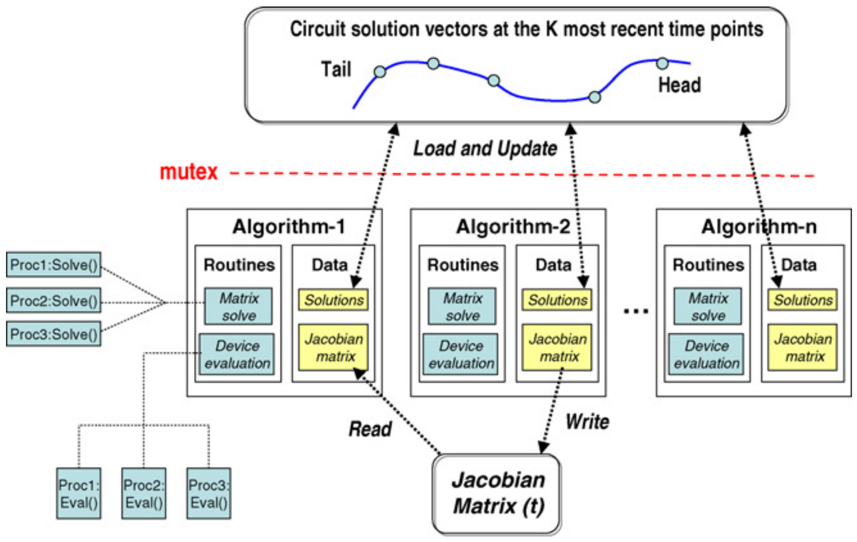


Fig. 7. Communication scheme in HMAPS.

Another important issue is the contention on cache and memory bus bandwidth. If several threads in HMAPS are doing potentially nonuseful work, they would still compete for the cache and memory bandwidth. This contention would deteriorate the cache and memory condition of all threads in the system, therefore, deteriorate the performance of HMAPS as a whole.

All the above issues have to be considered when making the proper selection of algorithms and core assignment in HMAPS. Based on our previous simulation experiments, SC method is likely to contribute the most in HMAPS. Therefore, we assign four cores to it in HMAPS. All the other three algorithms use one core each. Automatic algorithm selection and core assignment is possible with the performance model of HMAPS.

VI. Communications in HMAPS

To ensure algorithm diversities are well exploited during the transient simulation and all algorithms are properly synchronized, a number of guidelines are followed as given below.

1) The latest circuit solutions computed by the fastest algorithm shall be passed to all the slower algorithms as quickly as possible so that slower algorithms can use them as initial conditions and keep up with the fastest algorithm.
2) Every algorithm has the chance to contribute to the overall performance of HMAPS as long as it completes certain useful work fast enough.
3) Sufficient information shall be shared among all algorithms so that every algorithm has the initial conditions it needs to move forward.
4) Synchronization shall be independent of the number and choice of algorithms (e.g., the order of the numerical integration method).
5) Race condition must be avoided during synchronization.

We achieve all of these goals with the aid of a *global synchronizer*, which is visible to all algorithms as shown in Fig. 7. It contains circuit solutions at k most recent time points, where k is decided by the highest numerical integration order used among all the algorithms. In HMAPS, k is set to be 6 since the highest integration order used in our DASSL implementation is 5. The head and tail of these k time points are denoted as t_{head} and t_{tail} ($t_{\text{head}} > t_{\text{tail}}$), respectively. Each algorithm works on its own pace, and independently or *asynchronously* accesses the global synchronizer via a mutex guard which prevents the potential race condition. Hence, there is no direct interaction between the algorithms. When one algorithm finishes solving one time point, it will access the global synchronizer (the frequency of access can be tuned). If its current time point $t_{\text{alg.}}$ is ahead of the head of the global synchronizer, i.e., $t_{\text{alg.}} > t_{\text{head}}$, the head is updated by this algorithm to $t_{\text{alg.}}$ and the tail of the global synchronizer is deleted. In this way, the global synchronizer still maintains k time points and circuit solutions associated with them. If this algorithm does not reach as far as the head, but it reaches a time point that is ahead of the tail, the new solution is still inserted into the synchronizer and the tail is deleted. Additionally, before each algorithm starts to compute the next new time point, it also checks the global synchronizer to load the most recent initial conditions stored in the synchronizer so as to move down the time axis as fast as it can.

The pseudocode of the synchronization algorithm is listed below Algorithm 1. Every simulation algorithm in HMAPS uses Algorithm 1.

One favorable feature of this synchronization scheme is that it provides a transparent interface between an arbitrary number of algorithms with varying characteristics (e.g., independent dynamic time step control and varying numerical integration order): the algorithms do not *talk* to each other directly, rather, through the global synchronizer, they assist each other in a best possible way so as to collectively advance the MA transient simulation. The communication overhead of the scheme is quite low. Each algorithm accesses the global synchronizer only after solving the entire solution(s) at one (several) time point(s). Moreover, no algorithm is idle at any given time in

Algorithm 1 Synchronization algorithm

```
1:  while simulation not over do
2:      Mutex lock.
3:      if $t_{\text{alg.}} > t_{\text{head}}$ then
4:          Update global synchronizer.
5:      else if $t_{\text{head}} > t_{\text{alg.}} > t_{\text{tail}}$ then
6:          Insert solution into global synchronizer.
7:          Read global synchronizer as initial condition.
8:      else
9:          Read global synchronizer as initial condition.
10:     end if
11:     Mutex unlock.
12:     Solve for next time point.
13:     Update $t_{\text{alg.}}$ and its solution.
14: end while
```

this scheme, which avoids the time wasted in waiting, possibly in a direct synchronization scheme.

The above communication scheme is essentially interalgorithm solution sharing. Since the communication cost is very low on the shared-memory-based platform, we can have more beneficial interactions between algorithms to further improve the overall performance of HMAPS. According to (15), the fixed Jacobian matrix J_{sc} is critical to the performance of SC method. If J_{sc} is close to the current true Jacobian matrix, SC method will converge, otherwise it will probably diverge. During the transient simulation, circuit responses as well as the true Jacobian matrix are changing. For example, within interval A and C in Fig. 4, the circuit responses as well as the Jacobian matrix are not changing, therefore, SC method will proceed very fast. Within interval B, since the circuit responses and the Jacobian matrix is changing rapidly, SC method is likely to slow down or diverge. So using a fixed Jacobian matrix J_{sc} for the entire simulation period in SC method is not good. Since Newton's method is used in HMAPS, SC method can use the latest Jacobian matrix computed by Newton's method if its performance starts to degrade. In this way, SC method can always use a better J_{sc} on-the-fly and there is no need to manually choose chord values.

We also summarize the overall structure of HMAPS in Fig. 9. As a reference, the overall structure of MAPS is shown in Fig. 8 where only coarser-grained interalgorithm parallelism is used. It shall be noted this MA parallel paradigm is not only applicable to circuit simulation. It is possible to extend it to exploit algorithm diversity in a variety of parallel CAD applications.

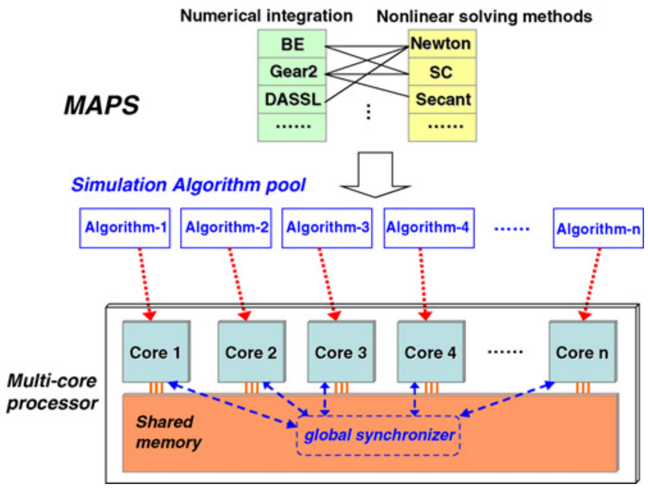


Fig. 8. Overall structure of MAPS.

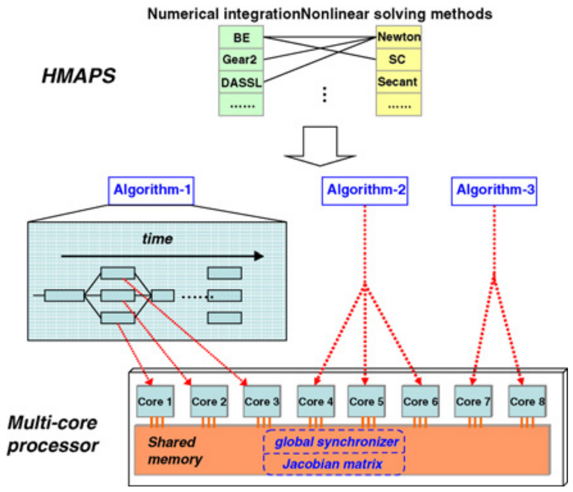


Fig. 9. Overall structure of HMAPS.

VII. Experimental Results

We demonstrate various aspects of HMAPS including runtime speedup, accuracy, synchronization overhead and the global synchronizer. As discussed earlier, we have four simulation algorithms in the current implementation of HMAPS: Newton's method + BE, Newton's method + Gear2, Newton's method + DASSL and SC method + dynamic time step rounding. In HMAPS, each algorithm is also capable of utilizing parallel device evaluation and parallel matrix solvers. Besides the four-algorithm HMAPS, we also implement the sequential version of these four simulation algorithms and HMAPS with interalgorithm parallelism only as references.

We have three different types of circuits in the experiments. The first type of circuit (CKT 1, 2, 3) is the transistor dominant combinational circuit; the second type (CKT 4, 5, 6) is the mesh-based clock distribution circuit [27]; the third type (CKT 7, 8) is analog-type circuit. For the first type, they have very regular structure and large size. They are dominated by MOSFET transistors. Mesh-based clock distribution circuits have similar structure but different sizes. They have large number of linear elements and small number of transistors. Analog-type circuits are small and have irregular structure. They have higher accuracy requirement compared with the other two types of circuits.

The parallel simulation code is multithreaded using Pthreads. Experiments are conducted on a Linux server with 8 GB memory and two quad-core processors running at 2.33 GHz. It is a symmetric multiprocessing system (SMP).

A. Runtime

Table II summarizes the runtime (in seconds) of four sequential algorithms and HMAPS with interalgorithm

TABLE II
RUNTIME (IN SECONDS) OF FOUR SEQUENTIAL ALGORITHMS AND HMAPS WITH INTERALGORITHM PARALLELISM ONLY (USING 4 THREADS)

CKT	Description	No. of Lin. ele.	No. of FETs	Newton+BE	Newton+Gear2	Newton+DASSL	SC	HMAPS w/Interalgorithm Parallelism Only	Speedup Over Newton+BE
1	Comb. logic 1	200	400	105.1	25.1	45.8	80.3	33.8	3.11x
2	Comb. logic 2	1000	2000	947.5	579.5	837.1	123.5	128.3	7.39x
3	Comb. logic 3	4000	8000	1230.0	879.8	626.0	83.3	94.6	13.0x
4	Clock mesh 1	10k	20	120.8	34.7	N/A	10.5	11.4	10.6x
5	Clock mesh 2	20k	40	1231.3	489.0	N/A	62.5	63.6	19.36x
6	Clock mesh 3	25k	50	4227.0	1547.5	N/A	210.3	220.3	19.19x
7	DB mixer	20	6	494.8	197.8	100.9	1623.0	151.6	3.26x
8	LNA + mixer	50	12	459.7	135.9	48.6	N/A	63.7	7.22x

TABLE III
HMAPS IMPLEMENTATION 1 (INTERALGORITHM PARALLELISM ONLY, USING FOUR THREADS) VS. HMAPS IMPLEMENTATION 2 (INTER AND INTRA-ALGORITHM PARALLELISM, USING EIGHT THREADS)

CKT	Description	No. of Threads Used in HMAPS Implementation 1	No. of Threads Used in HMAPS Implementation 2	HMAPS implementation 1 Runtime (s)	HMAPS Implementation 2 Runtime (s)	HMAPS Implementation 2 Speedup
1	Comb. logic 1	4	8	33.8	37.3	0.91×
2	Comb. logic 2	4	8	128.3	94.9	1.35×
3	Comb. logic 3	4	8	94.6	53.2	1.78×
4	Clock mesh 1	4	8	11.4	19.2	0.59×
5	Clock mesh 2	4	8	63.6	32.5	1.96×
6	Clock mesh 3	4	8	220.3	106.5	2.07×
7	DB mixer	4	8	151.6	171.3	0.88×
8	LNA + mixer	4	8	63.7	64.2	0.99×

parallelism only. It demonstrates the benefit of interalgorithm parallelism in the MA framework. There is no intra-algorithm parallelism in this implementation of HMAPS. Each algorithm is initiated by one thread, and each thread is running on one CPU core. Therefore, this implementation of HMAPS uses four CPU cores. The runtime speedup of HMAPS is with respect to the standard SPICE-like implementation: *Newton + BE*. For six out of eight examples (circuits $2, 3, 4, 5, 6, 8$), HMAPS achieves superlinear speedup (larger than $4\times$).

For larger circuits (circuits $2, 3, 5, 6$), SC method is the fastest sequential algorithm. This is because SC method avoids the costs of repeatedly factorizing large Jacobian matrices and evaluating device model derivatives, which are especially hight for large circuits. For smaller circuits (circuits $1, 4, 7, 8$), the advantage of SC method is smaller since the matrix size is small and there is smaller number of devices, and the contribution in HMAPS is mainly from advanced numerical integration methods (Gear2 and DASSL). In circuits 7 and 8 from Table II, one may note that HMAPS's performance is worse than the performance of Newton + DASSL. This discrepancy is due to the overhead in HMAPS. Example circuits 7 and 8 are the smallest circuits. Running more threads/algorithms on the system would deteriorate the cache condition and memory bandwidth. This deterioration becomes quite apparent for these two small circuits since memory/cache access takes a large portion of their overall simulation time. This is why HMAPS is apparently slower for these two circuits than the best sequential algorithm. HMAPS also shows great scalability in the sense that it achieves larger speedup for larger circuits.

In Table III, we demonstrates the runtime and speedup of HMAPS with interalgorithm parallelism only (*HMAPS implementation 1*) and HMAPS with both interalgorithm parallelism and intra-algorithm parallelism (*HMAPS implementation 2*). HMAPS implementation 1 has four algorithms and uses four cores. In principle, each algorithm in HMAPS implementation 2 can utilize many cores to do intra-algorithm parallelism. Due to the limited number of cores on the machine, we have to assign the available cores wisely. According to the principle mentioned in Section V-A, we assign four extra threads/cores for SC method to do parallel device evaluation and parallel matrix solve. Other three algorithms only use one thread/core, respectively. Therefore, HMAPS implementation 2 fully utilizes all eight cores on the machine.

We can see from Table III both implementations of HMAPS achieves good speedup in general. By creating more parallelism, HMAPS implementation 2 achieves larger speedup than HMAPS implementation 1 for four largest circuits (circuit $2, 3, 5, 6$). However, for smaller circuits (circuit $1, 4, 7, 8$), HMAPS implementation 2 does not improve the runtime compared with HMAPS implementation 1, sometimes, there is even a slow down. This "unexpected" slow down in runtime brings up an interesting problem in parallel circuit simulation, and even in parallel computing in general, that is, more parallelism is not always better.

For our particular parallel circuit simulation problem, this situation can be more carefully analyzed. Firstly, the parallelizability of a simulation algorithm can be limited due to the nature of the algorithm or implementation issues. Take the SC

TABLE IV
HMAPS IMPLEMENTATION 1 (INTERALGORITHM PARALLELISM ONLY, USING FOUR THREADS) VS. NEWTON+GEAR2

CKT	Description	Newton+Gear2 w. 1 Thread Runtime (s)	Newton+Gear2 w. 4 Threads Runtime (s)	HMAPS Implementation 1 Runtime (s)	HMAPS Implementation 1 Speedup w.r.t. Gear2 w. 1T	HMAPS Implementation 1 Speedup w.r.t. Gear2 w. 4T
1	Comb. logic 1	25.1	21.6	33.8	0.74×	0.64×
2	Comb. logic 2	579.5	195.3	128.3	4.52×	1.52×
3	Comb. logic 3	879.8	340.2	94.6	9.30×	3.60×
4	Clock mesh 1	34.7	39.4	11.4	3.04×	3.46×
5	Clock mesh 2	489.0	226.7	63.6	7.69×	3.56×
6	Clock mesh 3	1547.5	732.5	220.3	7.02×	3.33×
7	DB mixer	197.8	295.3	151.6	1.30×	1.95×
8	LNA + mixer	135.9	176.4	63.7	2.13×	2.77×

TABLE V
HMAPS IMPLEMENTATION 2 (INTER AND INTRA-ALGORITHM PARALLELISM, USING EIGHT THREADS) VS. NEWTON+GEAR2

CKT	Description	Newton+Gear2 w. 1 Thread Runtime (s)	Newton+Gear2 w. 8 Threads Runtime (s)	HMAPS Implementation 2 Runtime (s)	HMAPS Implementation 2 Speedup w.r.t. Gear2 w. 1T	HMAPS Implementation 2 Speedup w.r.t. Gear2 w. 8T
1	Comb. logic 1	25.1	61.9	37.3	0.67×	1.66×
2	Comb. logic 2	579.5	247.0	94.9	6.11×	2.60×
3	Comb. logic 3	879.8	239.8	53.2	16.54×	4.51×
4	Clock mesh 1	34.7	122.1	19.2	1.81×	6.36×
5	Clock mesh 2	489.0	183.5	32.5	15.05×	5.65×
6	Clock mesh 3	1547.5	552.3	106.5	14.53×	5.19×
7	DB mixer	197.8	523.8	171.3	1.15×	3.06×
8	LNA + mixer	135.9	271.2	64.2	2.12×	4.22×

method as an example, the parallel matrix solver [13] we use in HMAPS implementation 2 does not parallelize the matrix resolve routine which is a major cost in SC method. Therefore, this implementation issue affects the parallelizability of SC method so that its runtime does not scale well with the number of cores it uses. In principle, the matrix resolve routine can be parallelized just as the matrix factorization routine. If there is a parallel matrix solver available which can do parallel matrix resolve, we can improve the performance of SC method as well as HMAPS. Second, creating parallelism introduces overhead. Each thread is associated with its creation and termination cost. More threads doing fine-grained parallelism means more memory access. For smaller circuits, these overhead can not be neglected compared with the computational cost.

Results in Table III reveal the limitation of low-level parallelism which again justifies the usefulness of interalgorithm parallelism. Interalgorithm parallelism creates more opportunities for parallel circuit simulation which are impossible to find in intra-algorithm parallelism.

In Tables IV and V, we compare HMAPS with the parallel version of Newton+Gear2 algorithm. In Table IV, we compare HMAPS implementation 1 (Interalgorithm parallelism only, using four threads) with Newton+Gear2 algorithm using 1 and 4 threads. Newton+Gear2 using 4 threads can be viewed as a standard way of parallel circuit simulation. We can see from the last column that HMAPS implementation 1 gets reasonable speedup against Newton+Gear2 using 4 threads. In Table V, we compare HMAPS implementation 2 (inter and Intra-algorithm parallelism, using 8 threads) with Newton+Gear2 algorithm using 1 and 8 threads. Again, from the speedup numbers in the last column we can see that HMAPS implementation 2 gets reasonable speedup against Newton+Gear2 using 8 threads.

TABLE VI
COMPUTATIONAL COMPONENT COST (IN SECONDS) BREAKDOWN FOR EACH EXAMPLE CIRCUIT

CKT	Description	Jacobian Matrix Evaluation	Matrix Solve	Matrix Resolve
1	Comb. logic 1	9.8*e*-3	3.8*e*-3	6.8*e*-5
2	Comb. logic 2	7.0*e*-2	2.9*e*-1	3.3*e*-3
3	Comb. logic 3	6.1*e*-1	2.2e1	5.7*e*-2
4	Clock mesh 1	1.3*e*-2	1.6*e*-2	4.7*e*-4
5	Clock mesh 2	1.0*e*-1	1.5e1	4.0*e*-2
6	Clock mesh 3	1.8*e*-1	5.1e1	9.6*e*-2
7	DB mixer	1.6*e*-4	1.9*e*-4	2.0*e*-6
8	LNA + mixer	3.6*e*-4	2.9*e*-4	5.0*e*-6

In the reference algorithm (Newton+Gear2), we implement parallel device evaluation and parallel matrix solve. For small circuits, the cost of device evaluation and matrix solve are small, if we use 4 or 8 threads to parallelize the device evaluation and matrix solve, the overhead could be larger than the benefits, therefore, the overall simulation time will be slowed down. The slowdown comes from two sources: 1) SuperLU itself does not have good speedup for small matrices. Sometimes it can even be slowed down for small matrices, and 2) in thread-based parallel device evaluation,

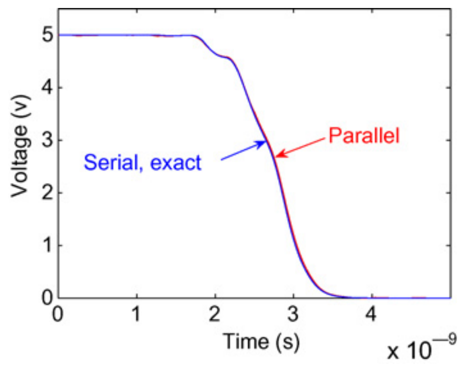


Fig. 10. Accuracy of HMAPS for a combinational logic circuit.

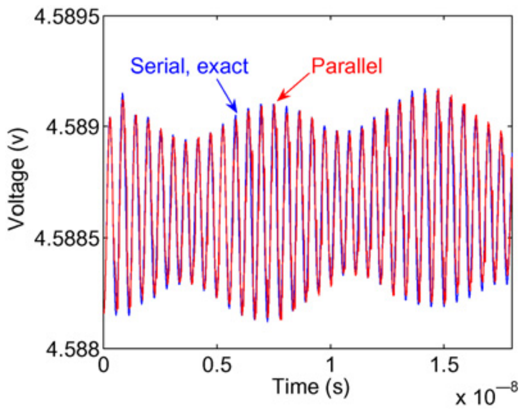


Fig. 11. Accuracy of HMAPS for a double-balanced mixer.

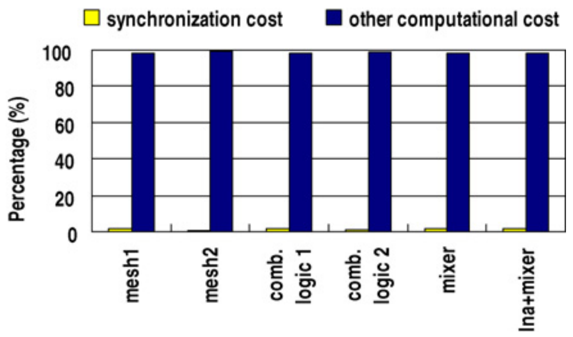


Fig. 12. Synchronization cost vs. other computational cost.

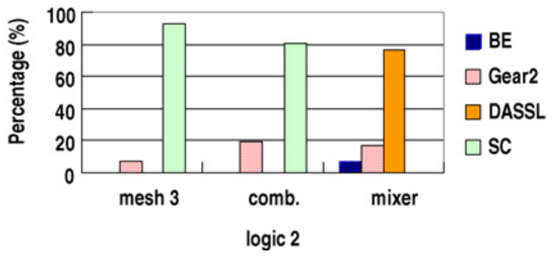


Fig. 13. Overall global synchronizer update breakdowns.

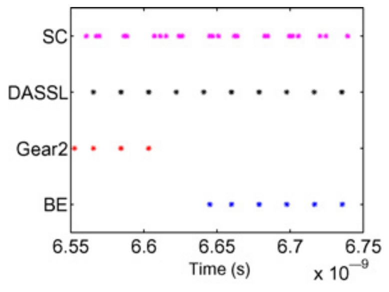


Fig. 14. Synchronizer updates within a local time window.

every thread is operating on its own unique set of devices, which is a subset of all devices. After each thread has evaluated its own set of devices, there is a merge-like operation which assembles the matrix fill-ins from all threads to form the final Jacobian matrix. There is inevitable overhead associated with this merge operation. If the original cost of device evaluation is small, parallel device evaluation would not give speedup, it could even slow down the device evaluation. For larger circuits, since the computational cost of device evaluation and matrix solve is large, the benefit of parallel computing is more obvious. For a better understanding of the performance modeling aspect of parallel circuit simulation, readers may refer to [28].

In Table VI, we list the cost breakdown for each circuit in terms of device evaluation, matrix solve, and matrix resolve.

B. Accuracy

We demonstrate the accuracy of HMAPS in Figs. 10 and 11, where the transient circuit waveforms simulated by HMAPS are compared with those obtained through the serial simulation of Newton+BE algorithm. A minimum step size is purposely chosen in the serial simulation such that the results may be considered as exact. The results computed by HMAPS are indistinguishable from the exact.

Each simulation algorithm used in HMAPS has the same convergence checking mechanism to ensure the computed results are accurate. We use both relative tolerance and absolute tolerance to check the norm of the residual of the system of nonlinear equations. Therefore, only accurate results are written into the global synchronizer. The simulation result of HMAPS is always accurate.

C. Synchronization

Interalgorithm parallelism in HMAPS has low synchronization overhead due its coarse-grained nature. In Fig. 12, we compare the overall synchronization cost associated with the interalgorithm parallelism and the computational cost. The synchronization usually takes about 1–2% of the total runtime.

We provide real-time profiling data to demonstrate the interactions between the four algorithms via the global synchronizer. Fig. 13 shows how often each individual algorithm updates the global synchronizer during the entire simulation in HMAPS. We can see that each algorithm has the chance to contribute to the global synchronizer. Variations exist across different test circuits. Fig. 14 is a local view of the global synchronization update within a time window. The y-axis marks the algorithm that updates the global synchronizer at each time point.

In Fig. 15, for the simulation of a clock mesh, we take three snapshots of the global synchronizer content with the relative

TABLE VII
MEMORY USAGE FOR EACH SIMULATION

CKT	Description	Newton+BE w. 1T (mB)	SC w. 1T	Newton+DASSL w. 1T (mB)	Newton+Gear2 w. 1T (mB)	Newton+Gear2 w. 4T (mB)	Newton+Gear2 w. 8T (mB)	HMAPS impl. 1	HMAPS impl. 2
1	Comb. logic 1	22	24 mB	22	22	23	23	25 mB	26 mB
2	Comb. logic 2	56	94 mB	56	56	57	60	106 mB	115 mB
3	Comb. logic 3	330	801 mB	330	330	335	340	997 mB	1.1 gB
4	Clock mesh 1	122	128 mB	122	122	122	129	135 mB	137 mB
5	Clock mesh 2	480	815 mB	480	480	498	501	1.1 gB	1.1 gB
6	Clock mesh 3	750	1.5 gB	750	750	757	765	2.0 gB	2.0 gB
7	DB mixer	14mB	14 mB	14	14	14	14	14 mB	14 mB
8	LNA + mixer	14	14 mB	14	14	15	15	15 mB	15 mB

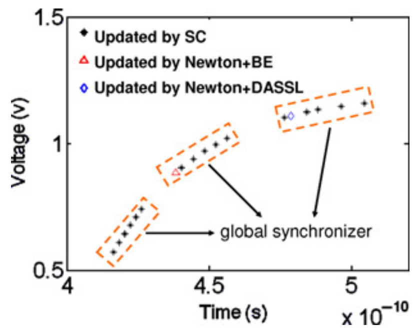


Fig. 15. Snapshot of the global synchronizer.

time locations of the six most recent circuit solutions marked. As can be seen, the stored six solutions may be contributed by different algorithms and their relative locations evolve over the time.

D. Memory Usage

In Table VII, we list the memory usage for every simulation. We can see that for a single simulation algorithm, SC method has the largest memory usage. This is because SC method needs to store pre-factorized Jacobian matrices. HMAPS has the largest memory usage among all simulation runs. This is expected since four algorithms in HMAPS use their own private data structure. However, the memory cost is still under control. For very large circuits, the challenge on the memory storage could be a potential limitation for HMAPS. On the other and, there are also lots of practical circuits, the storage requirement is not too demanding, yet the simulation needs to be speeded up. HMAPS would be a nice fit for such cases. Also, due to the inherent low communication overhead, HMAPS may be able to solve very large circuits over the network, where each node has sufficient memory to keep a separate copy of data structures.

E. Debugging Process

There is a lack of debugging tools that fully support multithreaded based programs. We take the divide-and-conquer approach in our debugging process. We first make sure that individual algorithm used in HMAPS are implemented correctly, then the communication scheme and interactions between algorithms.

VIII. CONCLUSION

A novel HMAPS approach was presented to achieve efficient coarse-grained parallel computing via exploration of algorithm diversity. The unique nature of the approach makes it possible to achieve superlinear runtime speedup and opens up new opportunities to utilize increasingly parallel computing hardware. Additionally, our approach requires minimum parallel programming effort and allows for reuse of existing serial simulation codes.

REFERENCES

[1] J. Held, J. Bautisa, and S. Koehl, "From a few cores to many: A terascale computing research overview," Intel Research White Paper, 2006.

[2] C. McNairy and R. Bhatia, "Montecito: A dual-core, dual-thread Itanium processor," *IEEE Micro*, vol. 25, no. 2, pp. 10–20, Mar. 2005.

[3] J. Friedrich, B. McCredie, N. James, B. Huott, B. Curran, E. Fluhr, G. Mittal, E. Chan, Y. Chan, D. Plass, C. Sam, L. Hung, L. Clark, J. Ripley, S. Taylor, J. Dilullo, and M. Lanzerotti, "Design of the Power6 microprocessor," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2007, pp. 96–97.

[4] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar, "An integrated quad-core Opteron processor," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2007, pp. 102–103.

[5] U. G. Nawathe, M. Hassan, L. Warriner, K. Yen, B. Upputuri, D. Greenhill, A. Kumar, and H. Park, "An 8-core 64-thread 64b power-efficient SPARC SoC," in *Proc. IEEE Int. Solid-State Circuits Conf.*, Feb. 2007, pp. 108–109.

[6] S. Borkar, "Thousand core chips: A technology perspective," in *Proc. IEEE/ACM Design Autom. Conf.*, Jun. 2007, pp. 746–749.

[7] S. Kravitz, R. Bryant, and R. Rutenbar, "Massively parallel switch-level simulation: A feasibility study," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst. Integr. Circuits Syst.*, vol. 10, no. 7, pp. 871–894, Jul. 1991.

[8] E. Carlson and R. Rutenbar, "Mask verification on the connection machine," in *Proc. IEEE/ACM Design Autom. Conf.*, Jun. 1988, pp. 134–140.

[9] R. Saleh, K. Gallivan, M. Chang, I. Hajj, D. Smart, and T. Trick, "Parallel circuit simulation on supercomputers," *Proc. IEEE*, vol. 77, no. 12, pp. 1915–1931, Dec. 1989.

[10] G. Yang, "Paraspice: A parallel circuit simulator for shared-memory multiprocessors," in *Proc. ACM/IEEE Design Autom. Conf.*, Jun. 1991, pp. 400–405.

[11] N. Rabbat, A. Sangiovanni-Vincentelli, and H. Hsieh, "A multilevel Newton algorithm with macromodeling and latency for the analysis of large-scale nonlinear circuits in the time domain," *IEEE Trans. Circuits Syst.*, vol. 26, no. 9, pp. 733–741, Sep. 1979.

[12] J. White and A. Sangiovanni-Vincentelli, *Relaxation Techniques for the Simulation of the VLSI Circuits*. Boston, MA: Kluwer, 1987.

[13] J. W. Demmel, J. R. Gilbert, and X. S. Li, "An asynchronous parallel supernodal algorithm for sparse Gaussian elimination," *SIAM J. Matrix Anal. Appl.*, vol 20, no. 4, pp. 915–952, 1999.

[14] M. Honkala, J. Roos, and M. Valtonen, "New multilevel Newton–Raphson method for parallel circuit simulation," in *Proc. Eur. Conf. Circuit Theory Design*, Aug. 2001, pp. 113–116.

[15] W. Dong, P. Li, and X. Ye, "Wavepipe: Parallel transient simulation of analog and digital circuits on multi-core shared memory machines," in *Proc. IEEE/ACM Design Autom. Conf.*, Jun. 2008, pp. 238–243.

[16] X. Ye, W. Dong, and P. Li, "A multi-algorithm approach to parallel circuit simulation," in *Proc. IEEE/ACM TAU Workshop*, Feb. 2008, pp. 73–78.

[17] X. Ye, W. Dong, P. Li, and S. Nassif, "Maps: Multi-algorithm parallel circuit simulation," in *Proc. Comput.-Aided Design, Int. Conf.*, 2008, pp. 73–78.

[18] F. Dartu and L. Pilleggi, "Teta: Transistor-level engine for timing analysis," in *Proc. IEEE/ACM Design Autom. Conf.*, Jun. 1998, pp. 595–598.

[19] P. Li and L. Pilleggi, "A linear-centric modeling approach to harmonic balance analysis," in *Proc. Design Autom. Test Europe*, Mar. 2002, pp. 634–639.

[20] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Ph.D. dissertation, EECS Dept., Univ. California, Berkeley, 1975.

[21] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *Numerical Solutions of Initial-Value Problems in Differential-Algebraic Equations*. New York: Elsevier, 1989.

[22] C. W. Gear, *Numerical Initial Value Problems in Ordinary Differential Equations*. Englewood Cliffs, NJ: Prentice-Hall, 1971.

[23] H. Shichman, "Integration system of a nonlinear network analysis program," *IEEE Trans. Circuit Theory*, vol. CT-17, no. 3, pp. 378–386, Aug. 1970.

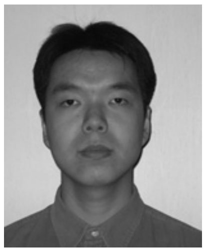
[24] K. R. Jackson and R. Sacks-Davis, "An alternative implementation of variable step-size multistep formulas for stiff odes," *ACM Trans. Math. Softw.*, vol. 6, no. 3, pp. 295–318, 1980.

[25] J. Ortega and W. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*. New York: Academic Press, 1970.

[26] X. Ye, M. Zhao, R. Panda, P. Li, and J. Hu, "Accelerating clock mesh simulation using matrix-level macromodels and dynamic time step rounding," in *Proc. Int. Symp. Quality Electron. Design*, Mar. 2008, pp. 627–632.

[27] P. J. Restle, T. G. McNamara, D. A. Webber, P. J. Camporese, K. F. Eng, K. A. Jenkins, D. H. Allen, M. J.Rohn, M. P. Quaranta, D. W. Boerstler, C. J. Alpert, C. A. Carter, R. N. Bailey, J. G. Petrovick, B. L. Krauter, and B. D. McCredie, "A clock distribution network for microprocessors," *IEEE J. Solid-State Circuits*, vol. 36, no. 5, pp. 792–799, May 2001.

[28] X. Ye and P. Li, "Parallel program performance modeling for runtime optimization of multi-algorithm circuit simulation," in *Proc. IEEE/ACM Design Autom. Conf.*, Jun. 2010, pp. 561–566.



Xiaoji Ye received the B.E. degree in electronic information science and technology from Wuhan University, Wuhan, China, in 2004, and the M.S. degree in computer engineering from Texas A&M University, College Station, in 2007. He is currently pursuing the Ph.D. degree in computer engineering from the Department of Electrical and Computer Engineering, Texas A&M University.

His current research interests include circuit simulation and analysis, interconnect modeling, timing/leakage analysis and optimization, clock network analysis and optimization, parallel circuit simulation and optimization, and organic transistor modeling.

Mr. Ye received the Best Paper Award from the 2008 Design Automation Conference and the Best in Session Award in SRC Techcon 2009.

Wei Dong (S'06–M'09) received the B.S. and M.S. degrees in electrical engineering from Xi'an Jiaotong University, Xi'an, China, and Shanghai Jiao Tong University, Shanghai, China, in 2000 and 2003, respectively, and the Ph.D. degree in electrical and computer engineering from Texas A&M University, College Station, in 2009.

Since 2009, he has been a Research Engineer with Texas Instruments, Dallas. His current research interests include VLSI circuit design and computer-aided design with an emphasis on high-performance parallel circuit simulation techniques.

Dr. Dong received the 2008 Design Automation Conference Best Paper Award, the 2009 SRC Techcon Best Paper in Session Award, and the 2008 International Conference of Computer-Aided Design Best Paper Award nomination.

Peng Li (S'02–M'04–SM'09) received the B.Eng. degree in information engineering and the M.Eng. degree in systems engineering from Xi'an Jiaotong University, Xi'an, China, in 1994 and 1997, respectively, and the Ph.D. degree in electrical and computer engineering from Carnegie Mellon University, Pittsburgh, PA, in 2003.

Since August 2004, he has been on the faculty of the Department of Electrical and Computer Engineering, Texas A&M Univeristy, College Station, where he is presently an Associate Professor. His past and present work has been focusing on analog and mixed-signal computer-aided design and testing, circuit simulation, design and analysis of power and clock distributions, interconnect modeling, statistical circuit design, parallel algorithms and implementations for solving large-scale computing problems in VLSI computer-aided design, and emerging applications. His current research interests include the general areas of VLSI systems, design automation, and parallel computing.

Dr. Li's work has been recognized by various awards, including two Design Automation Conference Best Paper Awards in 2003 and 2008, two SRC Inventor Recognition Awards in 2001 and 2004, the MARCO Inventor Recognition Award in 2006, the National Science Foundation CAREER Award in 2008, and the ECE Outstanding Professor Award from Texas A&M University in 2008. He is an Associate Editor for the IEEE TRANSACTIONS on COMPUTER-AIDED DESIGN and the IEEE TRANSACTIONS on CIRCUITS and SYSTEMS II. He has been on the committees of many international conferences and workshops, including ICCAD, ISQED, ISCAS, TAU, and VLSI-DAT, as well as the selection committee for the ACM Outstanding Ph.D. Dissertation Award in Electronic Design Automation. He served as the Technical Program Committee Chair for the ACM TAU Workshop in 2009 and the General Chair for the 2010 Workshop.

Sani Nassif (F'08) received the Bachelors degree from the American University of Beirut, Beirut, Lebanon, in 1980, and the Masters and Ph.D. degrees from Carnegie-Mellon University, Pittsburgh, PA, in 1981 and 1985, respectively.

He was with Bell Laboratories, Madison, WI, until 1996, and then joined the IBM Austin Research Laboratory, Armonk, NY, where he currently manages the Silicon Analytics Department. He has authored numerous conference and journal publications.

Dr. Nassif has received five Best Paper Awards (IEEE TCAD, ICCAD, DAC, ISQED, and ICCD), authored invited papers to ISSCC, IEDM, ISLPED, HOTCHIPS, and CICC, and given keynote and plenary presentations at Sasimi, ESSCIRC, BMAS, SISPAD, SEMICON, PATMOS, and ASAP. He is a member of the IBM Academy of Technology, a member of the ACM, and has a total of 44 patents.