# MAPS: Multi-Algorithm Parallel Circuit Simulation

Xiaoji Ye, Wei Dong, Peng Li
Department of ECE, Texas A&M University
College Station, TX 77843
Email: {yexiaoji,weidong, pli}@neo.tamu.edu

Sani Nassif
IBM Austin Research Laboratory
Austin, TX 78758
Email: nassif@us.ibm.com

***Abstract*— The emergence of multi-core and many-core processors has introduced new opportunities and challenges to EDA research and development. While the availability of increasing parallel computing power holds new promise to address many computing challenges in 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 Multi-Algorithm Parallel circuit Simulation approach (*MAPS*) and its multi-core implementation to expedite one of the most fundamental CAD applications: transistor-level transient circuit simulation. *MAPS* starts multiple simulation algorithms in parallel for a given simulation task. By properly synchronizing these algorithms on-the-fly, we exploit the diversity in simulation algorithms to achieve possibly superlinear overall speedup in transient simulation. In addition, our unique multi-algorithm framework allows unique safe exploration of simulation methods that are conventionally discarded due to convergence concerns. As a coarse grained parallel simulation approach, the implementation of *MAPS* demands a minimum of parallel programming effort and allows for reuse of existing serial simulation codes.**

## I. Introduction

The semiconductor industry is undergoing a shift from single-core processors to multi-core processors [1]–[3], which is changing the landscape of computing [4]. This shift also brings new opportunities and challenges to the CAD community [5]. Multi-core processors with increasing performance are widely accessible to designers who now have the amount of computing power that was only possible on expensive supercomputers and mainframes a decade ago [6]–[8]. Current industry trends clearly point to a continuing increase in the number of cores per processor, thus there is a strong need to develop new parallel computing tools which can fully utilize available hardware parallelism.

Parallel circuit simulation is by no means a new topic, and there exists a body of prior work such as [9], [10]. The most straightforward way is to parallelize the device evaluation and employ parallel matrix solver. However, it has been shown in [11], [12] that the runtime of parallel matrix solvers does not scale well with the number of processors. The waveform relaxation algorithm [10] is another powerful algorithm where parallel processing can be naturally applied, but it suffers from significant convergence problems in practice. Note that most prior work target traditional supercomputers and computer clusters, and may not be very applicable to current multi-core processors which exhibit significantly reduced inter-processor communication overhead. Recently, a so-called waveform pipelinging approach is proposed to exploit parallel computing for transient simulation on multi-core platforms [13].

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 steps within a single algorithm. This choice often leads to fine grained parallel processing and hence requires significant programming effort. In this work, we approach the problem from a somewhat unorthodox angle, we explore *Inter-algorithm Parallelism*. This unique perspective not only opens up new opportunities, but also allows us to explore advantages that are simply not possible when working within one fixed algorithm.

In our Multi-Algorithm Parallel circuit Simulation (*MAPS*) approach, multiple different simulation algorithms are initiated in parallel -using multi-threading- for a single simulation task [14]. These algorithms are synchronized on-the-fly during the simulation, and because they have a diverse CPU-time vs. error tradeoff, we can dynamically pick the best performing algorithm at every time point. In our initial implementation of MAPS, we included the standard SPICE-like algorithm as the solid backup solution, which guarantees that the worst case performance is no worse than a serial SPICE simulation. We also include some aggressive and, possibly non-robust, simulation algorithms which would normally not be considered in the typical single-algorithm implementation. In the end, this combination of algorithms in *MAPS* leads to superlinear runtime speedup in practical cases. We finally note that the approach applied in *MAPS* is largely independent of other parallelization approaches, for example, it can trivially be paired with more conventional approaches such as parallel device model evaluations and/or parallel matrix solvers.

## II. Overview of the Approach

### A. Algorithm diversity

In practice, a single simulation algorithm may behave differently within the entire simulation period. Take a simple circuit which consists of a nonlinear driver and its interconnect load as an example. We apply successive chord method [15], [16] to simulate the circuit. The voltage waveform at the driver output is shown in Fig. 1. 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, successive chord converges easily and moves fast during these two intervals. However, in time interval B, transistor operating conditions transit much faster. As a result, successive chord method is likely to converge slowly or even diverge.

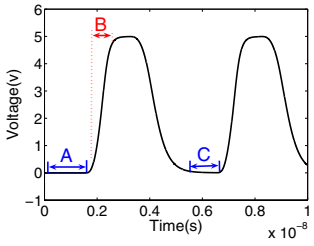



Fig. 1. Waveform at one node in a nonlinear circuit.

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

* This work was funded in part by SRC/GRC and NSF Career Award CCF-0747423.

parallel. Ideally, a pool of algorithms of diverse characteristics are desired. In this work, we pair various numerical integration methods with nonlinear solution methods to create a set of candidate simulation algorithms. We shall now consider how these algorithms should be integrated under a multi-algorithm (MA) framework. 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. Therefore, it is likely that its overall runtime performance may be neutralized by its *fast* and *slow* regions. In other words, the favorable performance of successive chord in intervals A and C is not well exploited. If other simulation methods encounter the same problem, the overall efficiency of the multi-algorithm approach is limited.

### B. Synchronization and granularity

The foregoing discussion reveals the importance of a key factor under the multi-algorithm context: inter-algorithm synchronization granularity. The simple multi-algorithm idea has a very coarse synchronization granularity - all the algorithms synchronize only once at the end of the simulation. This observation leads to a much more powerful MA concept, allowing for finer grained inter-algorithm synchronization. In *MAPS*, a flexible and conceptually clean synchronization model is adopted. Conceptually, the algorithms do not directly interact with each other, rather, they *asynchronously*, or *independently*, communicate with a *global synchronizer*. 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. Compared with the previous approach, the use of 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 V-B.

## III. MAPS: Diversity in Numerical integration

In this section, we exploit the possibility of incorporating a number of numerical integration methods into the proposed MAPS multi-algorithm 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 simulators usually include one step methods such as Backward Euler (BE), Trapezoidal (TR) and multi-step methods such as Gear method [17].

### A. One-step integration methods

Backward Euler and Trapezoidal 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

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

where $t_n \le \xi \le t_{n+1}$. Variable time steps are used in SPICE simulators to improve the runtime efficiency [17]. And local truncation error can be used to predict the variable time step during the simulation. Take Backward Euler 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 $h_{n+1} = t_{n+1} - t_n$, $\epsilon$ 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 re-computed. In Trapezoidal method, the same time step control mechanism is used except that equation (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 [17].

In comparison, 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. Multi-step integration methods

Gear methods [18] 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 [19]:

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

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

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

where $t_n \le \xi \le t_{n+1}$. In practise, 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 re-computations are needed so as to maintain a desirable accuracy level.

### C. Variable-order variable-stepsize integration methods

It is possible to integrate even more sophisticated high order methods into MAPS. 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 ODE's. 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 them, DASSL [20] is one of the most successful ones. DASSL uses the fixed leading coefficient BDF formulas [21] 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 $\omega_{n+1}^P$ is formed by interpolating solutions at the last $k+1$ time points: $(t_{n-k}, \cdots, t_{n-1}, t_n)$,

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

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

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

and $DD(x_n, x_{n-1}, \cdots, x_{n-i+1}), i > 1$ 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 $\omega_{n+1}^C$ is a polynomial which satisfies following conditions: first, it interpolates the predictor polynomial at $k$ equally spaced time points before $t_{n+1}$,

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

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 local truncation error as a measure to control the stepsize as well as the order. It estimates what the local truncation errors 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 re-compute $x_n$. 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 [20] 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 re-computed.
4) Predict the next timestep: $h_{n+1}$ if $x_n$ is accepted; a new $h_n$ if $x_n$ is re-computed.

### D. Exploiting diversity in numerical integration

Numerical integration methods vary in complexity, efficiency and robustness. The one-step 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. On the other hand, it is difficult to choose a single optimal numerical integration method for a simulation task *a priori*. The efficiency of a method is largely determined by the circuit and input stimulus, and it varies over the time as the circuit passes through different states. To this end, MAPS favorably allows for on-the-fly interaction of integration methods, at a controllable communication granularity, so as to achieve the optimal results via collaborative effort.

## IV. MAPS: Diversity in Nonlinear Iterative Methods

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

### A. Newton method

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

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

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

where at the $k$-th iteration, $\mathbf{J^k}$ is the *Jacobian matrix* of $\mathbf{F}$, which needs to be updated at every iteration; $\mathbf{\Delta v^{(k)}}$ is the solution update; $\mathbf{v^{(k)}}$ and $\mathbf{v^{(k+1)}}$ are the solution guesses at the $k$-th and $(k+1)$-th iterations, respectively. Despite its good robustness, Newton 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 the updated Jacobian matrix needs to be factorized, which is expensive, especially for large circuits.

### B. Successive chord method

Successive chord method is a constant Jacobian matrix type iterative method [22]. 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 during each nonlinear iteration. As a result, there is a significant

reduction in device model evaluation. Additionally, the fixed Jacobian can be factorized once and the LU factors can be reused to solve (13) efficiently. However, the downside of not updating the Jacobian matrix is that the convergence rate of successive chord method is linear instead of quadratic. The selection of the chord values (entries in the Jacobian) is very critical. Bad chord selection may lead to excessive number of iterations or even divergence. The convergence criteria of successive chord method [22] is

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

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

### C. Exploiting diversity in nonlinear iterative methods

Although there are a wide range of nonlinear iterative methods that can be used in MAPS, due to the scope of this paper, we only consider those two methods discussed above since they already show distinguishing tradeoffs between per-iteration cost vs. number of iterations required, and efficiency vs. robustness. Newton method has a higher per-iteration cost and the favorable quadratic convergence rate. Successive chord method has a lower per-iteration cost but a linear convergence rate. When the chord values are not chosen properly, successive chord may not even be able to converge. In terms of robustness, Newton method is more robust than successive chord method.

Being applied as a standard alone method, the weak convergence property of a non-robust iterative method can significantly constrain its application [15], [16]. In MAPS, since the standard Newton method is always chosen as a solid backup, other non-robust methods no longer have to converge during the entire simulation, significantly relaxing the convergence constraint. Moreover, non-robust methods are employed with a rather *different* objective in MAPS: 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 scaling of the parallel multi-algorithm framework.

## V. Algorithm Selection and Synchronization Scheme

### A. Algorithm selection

In the current implementation of MAPS, three numerical integration methods: BE, Gear2 and DASSL with independent dynamic time step control are paired with Newton method to form three complete simulation algorithms. The SPICE-like BE + Newton combination provides a basic guarantee for the success of the simulation. BE is paired with successive chord to create a fourth algorithm. To further enhance the runtime benefit of successive chord in transient analysis, a dynamic step step rounding technique [23] is used. Note that the exact Jacobian matrix also depends on the time step in numerical integration method. For example in Backward Euler, a grounded capacitor of value $c$ contributes a stamp $c/h$ to the Jacobian, where $h$ is the time step. 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 $\{h_{min}, 2h_{min}, 4h_{min}, \cdots, h_{max}\}$, where $h_{min}$ and $h_{max}$ are estimated min/max time steps computed by dynamic time step control [23]. The total number of discrete time steps is given by $1+\lceil log_2(h_{max}/h_{min})$ In this case, 10 discrete time points can cover a $1000X$ span of time step. During the simulation, the computed variable time step is always rounded down to the nearest smaller value in the predefined time step set. Then a pre-factorized Jacobian 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.

### B. Synchronization

To optimally synchronize all algorithms during the simulation, a number of guidelines are followed

- The fastest algorithm shall inform all the slower algorithms such that the slower algorithms can keep up with the fastest one as quickly as possible;
- Every algorithm has the chance to contribute in the collaborative effort as long as it completes certain useful work fast enough;
- Sufficient information shall be shared between all algorithms;
- Race condition must be avoided during synchronization;
- Synchronization shall be independent of the number and choice of algorithms (e.g. order of numerical integration).

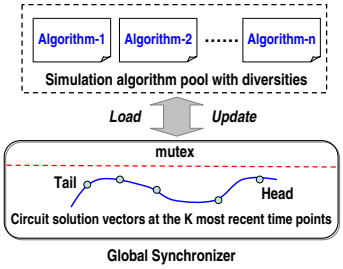



Fig. 2. Global synchronizer in MAPS

We achieve all of these goals with the aid of a *global synchronizer*, which is visible to all algorithms, as shown in Fig. 2. 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 MAPS, $k$ is set 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 $t_{tail}$ ($t_{head} > t_{tail}$), respectively. Each algorithm works on its own pace and independently, or *asynchronously* access the global synchronizer via a mutex. Hence, there is no direct interaction between the algorithms. If 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 $t_{alg.}$ is after the head of the global synchronizer, i.e. $t_{alg.} > t_{head}$, the head is updated by this algorithm and the tail of the global synchronizer is deleted. In this way, the synchronizer still maintains $k$ but updated solutions. 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.

One favorable feature of this synchronization scheme is that it provides a transparent interface between an arbitrary number of algorithms: 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 multi-algorithm transient simulation. The communication overhead of this scheme is quite small. Each algorithm accesses the global synchronizer only after solving the entire solution solution(s) at one (or several) time point(s). Moreover, no algorithm is idle at any given time in this scheme, which avoids the time wasted in waiting.

We summarize the overall structure of MAPS in Fig. 3. It shall be noted that the idea of multi-algorithm parallel paradigm is extendable

to other CAD applications.

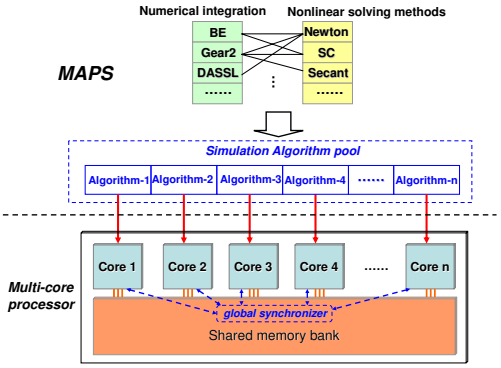



Fig. 3. Overall structure of MAPS.

## VI. Experimental Results

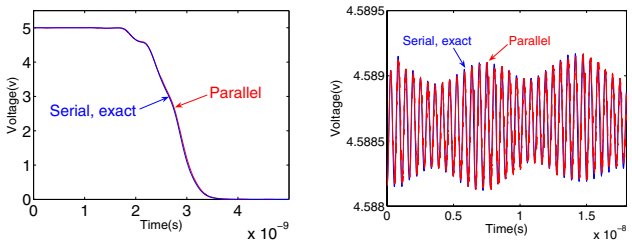
We demonstrate various aspects of MAPS including runtime speedup, accuracy, synchronization overhead, scalability and the global synchronizer. As discussed in Section V-A, we have four simulation algorithms in the current implementation of MAPS: Newton method + Backward Euler, Newton method + Gear2, Newton method + DASSL and successive chord + BE + dynamic time step rounding. They all have the same convergence criteria and LTE bound. Besides the four-algorithm MAPS, we also implement serial versions of these four algorithms as references. A set of test circuits with varying natures, ranging from passive element dominant circuits (clock meshes) to transistor dominant circuits (combinational logic and analog circuits), are used in the experiments. The parallel simulation code is multi-threaded using Pthreads on a Linux server with 8GB memory and two quad-core processors running at 2.33GHz. Our four-algorithm MAPS effectively utilizes four cores by assigning one algorithm per core.

### A. Runtime

Table. I summarizes the runtimes of MAPS and serial versions of the four algorithms. The runtime speedup of MAPS is with respect to the standard SPICE-like implementation: *Newton + BE*. For passive element dominant circuit like clock meshes, *successive chord + dynamic time step rounding* can significantly outperform other algorithms. The use of a few fixed Jacobian matrices has a significant advantage because of high cost of Jacobian matrix factorization in a large clock mesh. On the other hand, the multi-algorithm framework safeguards the robustness of the entire simulation even when successive chord method possibly diverges occasionally. As a result, MAPS achieves significant runtime speedups (up to 35x). For transistor dominant digital circuits (adder and ring oscillators), no individual algorithm can significantly outperform others. Successive chord method can not simulate the adder circuit successfully because it has relative large number of transistors and each one is going through some different operating region change. Under this case, MAPS still achieve superlinear runtime speedup (larger than 4X). For the last two analog circuits, successive chord method has a hard time in solving them because it takes excessive amout of iterations to converge. Thus for the 6th and 7th circuits, we effectively have only three useful algorithms in MAPS, which still achieve a superlinear runtime speedup (larger than 3X).

### B. Accuracy

Accuracy of MAPS is demonstrated in Fig. 4(a) and 4(b), where the transient waveforms simulated by MAPS are compared with those obtained through the serial simulation of Newton+BE algorithm. A minimum step size is chosen in the serial simulation such that the serial results can be considered as exact. The results computed by MAPS are indistinguishable from the exact.




(a) (b)

Fig. 4. (a) Accuracy of MAPS for a combinational logic circuit, (b) Accuracy of MAPS for a double-balanced mixer.

### C. Synchronization overhead

One of the major roadblocks to parallel computing is the synchronization/communication overhead. MAPS has low synchronization overhead due its coarse grained parallel computing nature. In Fig. 5, we compare the overall simulation synchronization cost and the computational cost other than synchronization. The synchronization usually takes about $1 \sim 2\%$ of the total runtime.

### D. Scalability

MAPS also exhibits excellent scalability. In Fig. 6, we show the runtime of the serial *Newton+BE*, and three versions of MAPS with the number of algorithms/threads ranging from two to four. It can be seen that the runtime of MAPS scales favorably as the number of the algorithms increases.

### E. Global synchronizer

We provide real-time profiling data to demonstrate the interactions between the four algorithms via the global synchronizer. Fig. 8 shows how often each individual algorithm updates the global synchronizer during the entire simulation. We can see that successive chord and DASSL update the global synchronizer more often than the other two

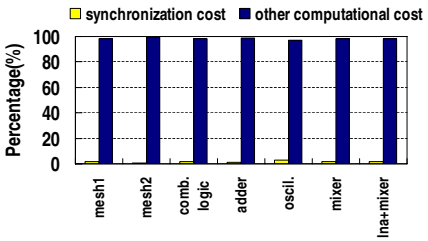



Fig. 5. Synchronization cost vs. other computational cost.

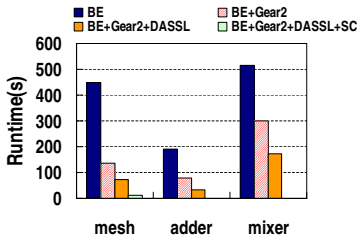


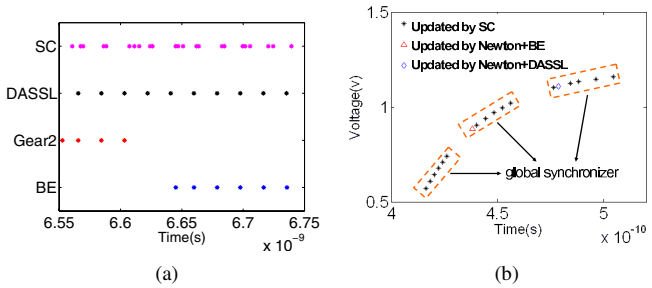

Fig. 6. Scalability of MAPS in the number of algorithms.

TABLE I

RUNTIME(S) COMPARISON BETWEEN FOUR-ALGORITHM MAPS AND FOUR SERIAL ALGORITHMS

| CKT | Description | # of lin. ele. | # of FETs | Newton+BE | Newton+Gear2 | Newton+DASSL | SC | Parallel (MAPS) | Speedup over Newton+BE |
|---|---|---|---|---|---|---|---|---|---|
| 1 | clock mesh1 | 10K | 26 | 447.07 | 259.09 | 139.84 | 14.74 | 13.06 | 34.23x |
| 2 | clock mesh2 | 20K | 40 | 796.06 | 736.65 | 531.69 | 40.77 | 38.29 | 20.79x |
| 3 | comb. logic | 13 | 26 | 35.70 | 28.03 | 26.32 | 12.82 | 10.41 | 3.43x |
| 4 | adder | 30 | 200 | 189.23 | 85.35 | 61.87 | N/A | 32.51 | 5.82x |
| 5 | ring oscillator | 15 | 30 | 284.73 | 84.68 | 69.84 | 123.05 | 63.30 | 4.50x |
| 6 | DB mixer | 20 | 6 | 512.35 | 358.15 | 192.79 | 882.52 | 172.44 | 2.97x |
| 7 | LNA+mixer | 50 | 12 | 626.53 | 326.18 | 210.05 | 1126.32 | 201.17 | 3.11x |

algorithms, hence they contribute most in MAPS. It shall be noted that variations do exist across different test circuits. Fig. 7(a) 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. 7(b), for the simulation of a clock mesh, we take three snapshots of the global synchronizer content with the relative time locations of the 6 most recent circuit solutions marked. As can be seen, the stored 6 solutions may be contributed by different algorithms and their relative locations evolve over the time.




(a) (b)

Fig. 7. (a) Synchronizer updates within a local time window, (b) Snapshot of the global synchronizer.

## VII. CONCLUSION

A novel multi-algorithm parallel simulation approach is 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 small parallel programming effort and is extendable to other CAD applications.

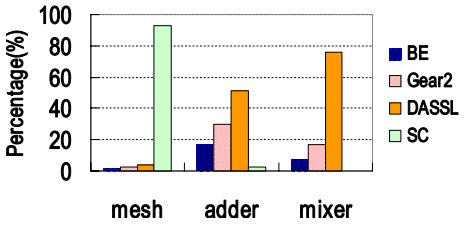



Fig. 8. Overall global synchronizer update breakdowns.

## REFERENCES


[1] C. McNairy and R. Bhatia. Montecito: a dua-core, dual-thread Itanium processor. *IEEE Micro*, 25(2):10–20, March 2005.

[2] J. Friedrich et al. Design of the Power6$^{TM}$ microprocessor. In *IEEE Int. Solid-State Circuits Conf.*, pages 96–97, February 2007.

[3] J. Dorsey et al. An integrated quad-core Opteron$^{TM}$ processor. In *IEEE Int. Solid-State Circuits Conf.*, pages 102–103, February 2007.

[4] J. Held, J. Bautisa, and S. Koehl. From a few cores to many: a tera-scale computing research overview. Intel Research White Paper.

[5] S. Borkar. Thousand core chips - a technology perspective. In *IEEE/ACM Design Automation Conf.*, pages 746–749, June 2007.

[6] S. Kravitz, R. Bryant, and R. Rutenbar. Massively parallel switch-level simulation: a feasibility study. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems of Integrated Circuits and Systems*, 10(7):871–894, July 1991.

[7] E. Carlson and R. Rutenbar. Mask verification on the CONNECTION MACHINE. In *IEEE/ACM Design Automation Conf.*, pages 134–140, June 1988.

[8] R. Saleh, K. Gallivan, M. Chang, I. Hajj, D. Smart, and T. Trick. Parallel circuit simulation on supercomputers. *Proc. of IEEE*, 77(12):1915–1931, December 1989.

[9] G. Yang. Paraspice: a parallel circuit simulator for shared-memory multiprocessors. In *ACM/IEEE Design Automation Conf.*, pages 400–405, 1991.

[10] J. White and A. Sangiovanni-Vincentelli. *Relaxation techniques for the simulation of the VLSI circuits*. Kluwer Academic Publishers, Boston, 1987.

[11] S. Markus et al. Performance evaluation of mpi implementations and mpi basedparallel ellpack solvers. In *MPI Developer's Conference*, pages 162–169, July 1996.

[12] H. Kotakemori, H. Hasegawa, and A. Nishida. Performance evaluation of a parallel iterative method library using openmp. In *Eighth International Conference on High-Performance Computing in Asia-Pacific Region*, pages 5–10, Dec. 2005.

[13] W. Dong, P. Li, and X. Ye. Wavepipe: parallel transient simulation of analog and digital circuits on multi-core shared memory machines. In *IEEE/ACM Design Automation Conf.*, pages 238–243, June 2008.

[14] X. Ye, W. Dong, and P. Li. A multi-algorithm approach to parallel circuit simulation. In *IEEE/ACM TAU workshop*.

[15] F. Dartu and L. Pilleggi. Teta: Transistor-level engine for timing analysis. In *IEEE/ACM Design Automation Conference*, pages 595–598, Jun. 1998.

[16] P. Li and L. Pilleggi. A linear-centric modeling approach to harmonic balance analysis. In *Design, Automation and Test in Europe*, pages 634–639, March 2002.

[17] Laurence W. Nagel. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. PhD thesis, EECS Department, University of California, Berkeley, 1975.

[18] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1971.

[19] H. Shichman. Integration system of a nonlinear network analysis program. *IEEE Trans. on Circuit Theory*, CT-17(3):378–386, August 1970.

[20] K. E. Brenan, S. L. Campbell, and L. R. Petzold. *Numerical Solutions of Initial-Value Problems in Differential-Algebraic Equations*. Elsevier Science Publishing Co., Inc., New York, USA, 1989.

[21] K. R. Jackson and R. Sacks-Davis. An alternative implementation of variable step-size multistep formulas for stiff odes. *ACM Trans. Math. Software*, (6):295–318, 1980.

[22] J. Ortega and W. Rheinboldt. *Iterative solution of nonlinear equations in several variables*. Academic Press, 1970.

[23] 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 *Intl. Symp. on Quality Electronic Design*, pages 627–632, March 2008.