Scalable Trajectory Methods for On-Demand Analog Macromodel Extraction

Saurabh K Tiwary
Carnegie Mellon University
Pittsburgh,PA USA
stiwary@ece.cmu.edu

Rob A Rutenbar
Carnegie Mellon University
Pittsburgh,PA USA
rutenbar@ece.cmu.edu

ABSTRACT

Trajectory methods sample the state trajectory of a circuit as it simulates in the time domain, and build macromodels by reducing and interpolating among the linearizations created at a suitably spaced subset of the time points visited during training simulations. Unfortunately, moving from simple to industrial circuits requires more extensive training, which creates models too large to interpolate efficiently. To make trajectory methods practical, we describe a scalable interpolation architecture, and the first implementation of a complete trajectory "infrastructure" inside a full SPICE engine. The approach supports arbitrarily large training runs, automatically prunes redundant trajectory samples, supports limited hierarchy, enables incremental macromodel updates, and gives 3-10X speedups for larger circuits.

Categories and Subject Descriptors

I.6.5 [**Simulation and Modeling**]: Model Development

General Terms

Algorithms, Design

Keywords

Circuit, trajectory method, analog, macromodel, SPICE

1. INTRODUCTION

In the digital world, a hierarchy of established modeling abstractions allows us to move with relative ease from RTL to logic to circuit, and back up for verification. The situation in the analog world is much less satisfactory. Today, we design and verify at system level using simple functional models [1] [2], and can employ recent synthesis tools to render completed analog circuits for each block [3]. But we cannot reassemble and simulate the entire system at the device level to verify it: these problems are much too large. Nevertheless, some form of detailed system verification is essential: we cannot predict how subtle analog non-idealities may conspire to create problems until all blocks are assembled.

The standard solution is to use analog *macromodels*. Macromodels are simplified circuits which capture just the essential behaviors of some target circuit, and are fast enough to support full system simulation. Today, the tools we have to create such macromodels are extremely *ad hoc*. The most common strategy parameterizes a simple circuit *template* via curve-fitting to match relevant behaviors of the target circuit. Unfortunately, just as analog circuits themselves are most often created by experts, so too are their macromodels. Indeed, the larger problem is that for any given circuit–and especially custom circuits–we often lack a suitable template, fitting recipe, or modeling expert. In an ideal world, we should be able to extract macromodels *on demand*, as needed.

The increasing number of mixed-signal designs only magnifies this problem. The essential difficulty is that we seek reduced models of nonlinear behaviors. We have today a rigorous foundation for reduced order linear modeling [4] [5]. However, we lack any unified theory for the general nonlinear case, although there is promising work for important sub-problems, e.g., Volterra models of weakly nonlinear behavior [6] [7] [8].

In this paper, we focus on a class of macromodels called *trajectory methods* [9] [10] [11] [12] [13]. Trajectory methods sample the state trajectory of a circuit as it is simulated in the time domain, and build a macromodel by reducing the linearizations created at an appropriately chosen subset of the time points visited during training simulations, and then interpolating among them. Interpolation combines these reduced linearizations to predict the dynamic behavior of the circuit at any new point in the state space not visited during prior training. Trajectory methods can build macromodels *on demand*: both the "template" and the "fitting" come directly from training runs. Results to date have been extremely promising. However, there are several significant obstacles to practical application. First, existing methods scale poorly, as we move from simple to industrial circuits. Larger circuits require more training, and can visit trajectories requiring 100X more linearizations, overwhelming the existing interpolation algorithms. Second, no prior methods have been demonstrated inside a complete SPICE engine, with full support for modern device models. Analog methodologies rely heavily upon carefully qualified device models and simulators; to be regarded as trustworthy, we must demonstrate that trajectory models can be integrated in the same simulators.

In this paper, we describe a novel, scalable trajectory method for analog macromodeling, its implementation in a full SPICE engine (Berkeley SPICE3f5 [14]), and the essential numerical issues involved. Sec.2 gives some basic background on trajectory methods, and the features missing in prior efforts. Sec.3 describes the elements of our scalable formulation: fast nearest neighbor (NN) based interpolation, automatic trajectory sample pruning, low-overhead incremental model updating, and hierarchical invocation of trajectory models. Sec.4 shows experimental results for a range of circuits. Finally, Sec.5 offers concluding remarks.

2. BACKGROUND

2.1 Trajectory-Based Models

Circuit simulators represent a circuit as a set of nonlinear differential equations. In state-space form, these can be written as:

$$\frac{dg(x)}{dt} = f(x) + B(x)u; \qquad y = C^T x \tag{1}$$

where $x = x(t) \in R^N$ is a time-varying state vector representing node voltages and branch currents; $g : R^N \rightarrow R^N$ and $f : R^N \rightarrow R^N$ represent nonlinear charge/flux and current elements, respectively; $B = B(x)$ is a state-dependent $N \times M$ input matrix (but often taken as constant, independent of system state); $u \in R^M$ is a time-varying vector of inputs to the system; and C is an $N \times K$ matrix mapping internal state to the time-varying output vector $y \in R^K$. We simulate the circuit by solving this differential equation, for a given input signal $u = u(t)$, on a specified time interval $t \in [0, T]$, for a given initial state x_0, thereby computing $y(t)$.

The twin difficulties of simulating complex circuits are: (a) the order (N) of the resulting equations is large, and (b) the nonlinearities associated with each transistor require many expensive model evaluations as the circuit moves through its state-space. Together, these make large circuits with complex models slow to simulate. We can use trajectory methods to attack these problems in two ways.

First, we reduce the dimensionality of the overall system of equations. We approximate the real N-dimensional state-vector x with a much smaller vector z of order $q << N$. The idea is to approximate the original system by carefully selecting a reduced subspace wherein most of the dynamics of the system occur. This can be done via projection methods, by constructing a suitable reduction matrix V of size $N \times q$ whose columns define a basis in the reduced state-space. So, we approximate states x from the original state space by reduced states z: $x \simeq \hat{x} = Vz$. Reduction techniques combining Krylov [15] and TBR [9] methods can be employed here.

Second, we replace expensive nonlinear model evaluations with simpler lookups and interpolation. Since evaluating $f(.)$ and $g(.)$ is expensive, we approximate these as a simple first-order Taylor series, expanded around some state x_0, for example:

$$f(x) \simeq f(x_0) + A_0(x - x_0); \quad g(x) \simeq g(x_0) + G_0(x - x_0) \tag{2}$$

where A_0, G_0 are the Jacobians of f and g respectively at x_0. (One can also choose a second-order expansion, at additional complexity; see [11].) Expressing Equation (1) now in terms of the reduced state vector z, and this Taylor expansion, the final reduced system becomes:

$$\begin{cases} G_{0r}\frac{dz}{dt} = V^T f(x_0) + A_{0r}(z - z_0) + B_{0r}u \\ \qquad y = C_r^T z \end{cases} \tag{3}$$

where $A_{0r} = V^T A_0 V$;$G_{0r} = V^T G_0 V$;$B_{0r} = V^T B(x_0)$;$C_r = V^T C$. Figure 1 illustrates how the interpolation mechanism works. When a nonlinear system is excited by an input signal $u(t)$, it moves along a path in its state-space called a *trajectory*. The system is sampled at different points lying on its trajectory, thereby generating local linearizations (i.e., Jacobians) that are easily obtained during simulation. These capture the dynamics of the system around each sampled state-space point.

If we linearize at appropriately spaced points on the trajectory, and then reduce the order of these linearizations, we can approximate the dynamics at any new point in the space by interpolating among these saved linearized reduced order equations. Figure 1 shows a 3-dimensional state-space ($N = 3$); the spheres are the regions in which (we hope) the sampled linearizations are effective in capturing the dynamics of the circuit; we reduce each of these to order two ($q = 2$). As the system moves through the reduced state space, we re-evaluate (interpolate) the state-space equation and solve it to evolve the system to a new point in the state-space.

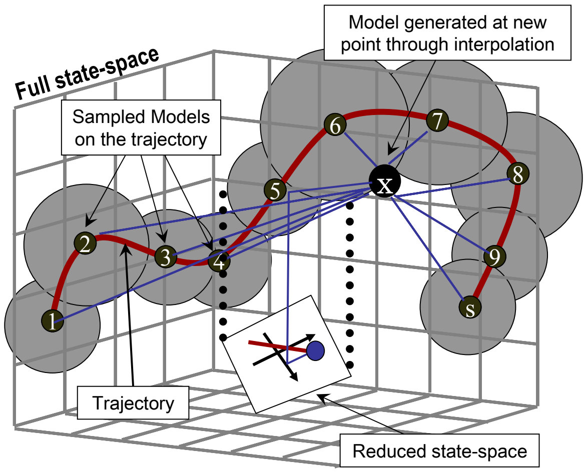

Full state-space
Model generated at new point through interpolation
Sampled Models on the trajectory
Trajectory
Reduced state-space
1
2
3
4
5
6
7
8
9
s
x

Figure 1: Models generated at sampled points on the trajectory of a system in its state-space. State-space equation at a new point in state-space is generate through interpolation of the reduced order sampled models.

Suppose, reduced linearizations have been generated at s points on the trajectory. Interpolation builds a single effective linearization for the new point x in state space as a weighted sum of these s stored linearizations.

$$w_i(z) = \frac{(exp(z_i - z)^2)^{-k}}{\sum_{i=1}^{s}(exp(z_i - z)^2)^{-k}} \quad \text{for i=1,2,..,s} \tag{4}$$

where, $w_i(z)$ is the weight contribution of the i^{th} linearization on the trajectory, for the point z's state-space matrix in the reduced order state-space. We used a value of $k = 10$ in our implementation. The weighting function [10][16] (Equation 4) is based on the heuristic that the state-space equations for the points lying inside the spheres in Figure 1 are similar to the equation for the linearized point at the center of the respective spheres. Interpolation lets us approximate the overall state-space equation as

$$\begin{cases} \frac{d}{dt}((\sum_{i=1}^{s} w_i(z)G_{ir})z + \sum_{i=1}^{s} V^T(g(x_i) - G_i x_i)w_i(z)) \\ = (\sum_{i=1}^{s} w_i(z)A_{ir})z + \sum_{i=1}^{s} V^T(f(x_i) - A_i x_i)w_i(z) + Bu \\ \qquad y = C_r^T z \end{cases} \tag{5}$$

where,

$$G_{ir} = V^T G_i V,\ A_{ir} = V^T A_i V,\ C_r = C^T V \text{ and } \sum_{i=1}^{s} w_i(z) = 1.$$

This formulation has been referred to as *piecewise linear trajectory-based model order reduction* [10]. The "piecewise linear" derives from the first-order Taylor expansion and the weighted linear combination form for the interpolation. The strategy works well when we have enough training trajectories to create a sufficient density of "overlapping" linearizations, as represented by the spheres in Figure 1, and when the dynamics of the circuit can be approximated with much reduced versions of these linearizations. In several experiments to date, these seem to be viable assumptions.

2.2 Challenges for Trajectory Methods

We have implemented some of these methods, and in the course of applying them to several circuits, we discovered a variety of practical shortcomings:

- *Fragility*: Using a simplistic training strategy to generate trajectory macromodels can create very fragile models. For example, most prior efforts train with at most a handful of simple waveforms. It is not difficult to find a test-case where such generated macromodels break. Figure 2 shows an example where the output of the macromodel differs significantly from that of the circuit for a test input waveform which was different from the training waveform.

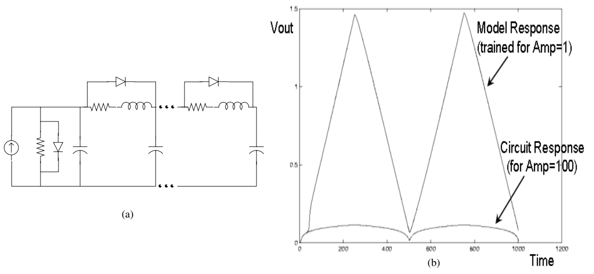


Figure 2: (a)Non-linear transmission line circuit schematic. (b) Waveform comparing the model and the circuit's response for a test input of 100A. The model was trained for an input of 1A.

- *Scalability*: The obvious answer to the above problem is to use a more rigorous training scheme. We find this works well, but the problem with a richer training set is that it can create a much larger set of linearized trajectory points. It is easily possible to generate 10^4 such points, i.e., 100x more than previous training methods. Earlier approaches that interpolate using *all* the sampled linearizations are easily overwhelmed. We need an interpolation scheme that has complexity much lower than $O(N)$.
- *Hierarchy*: We macromodel so that circuit-level models can be inserted in system-level contexts and simulated. Previous approaches demonstrated the feasibility of building trajectory models for an individual circuit, but not of inserting such models back into the simulator in a system context. We need an approach to handle such hierarchy.

We address all these problems in the following section.

3. SCALABLE TRAJECTORY MODELING

We have implemented a scalable trajectory modeling framework inside Berkeley-SPICE3f5 ("SPICE3") with full BSIM3 support. A simulateable circuit netlist is all that is required from the user to generate the macromodel for a given circuit. The generated macromodel can be used in another SPICE netlist as a new SPICE element ready to be simulated along with other circuit elements. We describe the essential pieces of this formulation in this section. Algorithm 1 shows the pseudo-code for the overall macromodeling flow.

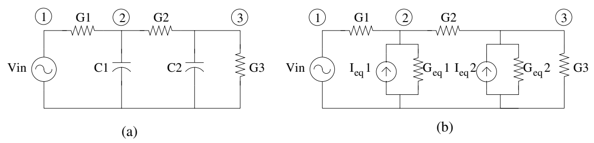
3.1 Data Extraction from Simulator

Our first problem is mainly one of bookkeeping: we need a classical state-space formulation in order to build the reduction matrices we need. Most SPICE engines, including SPICE3, do not create the necessary matrices directly. We must remedy this.

As listed in lines 3-5 of Algorithm 1, we extract the conductance(G) and capacitance(C) matrix for the circuit at each Newton-Raphson converged time-point during the transient simulation. SPICE3 does not use the state-space formulation to solve the circuit equation; it uses Modified Nodal Analysis (MNA) instead. Thus, capacitors and inductors are represented using their Thevenin/Norton equivalent companion models for trapezoidal integration approximation [17]. To extract the linearized G and C matrices, we visit the model evaluation files for all the circuit elements (*e.g.*, resistor, capacitor, transistor model (BSIM3) etc.) *before* they are stamped into a $Yv = J$ format. The mechanics are straightforward; roughly speaking, we visit each element in a circuit netlist at the successful convergence of a time-step during transient analysis and stamp the values of capacitance and conductance for that particular element into separate G and C matrices. We also stamp the current flowing at each node in the circuit into a vector. Figure 3 shows the implementation differences between a SPICE-like MNA matrix setup and the state-space based version we need.

Algorithm 1 Pseudo-code for macromodel generation

```
 1: for all input training waveforms do
 2:     perform transient simulation in SPICE3f5
 3:     for all Newton-Raphson converged time-steps do
 4:         extract G and C matrix and I vector for ckt
 5:     end for
 6: end for
 7: create reduction matrix(V)
 8: for all sampled state-space point do
 9:     reduce order of state-space equation using V
10: end for
11: prune the number of models
12: create database for efficient nearest-neighbor look-up
```



$$\begin{pmatrix} C1 & 0 \\ 0 & C2 \end{pmatrix} \frac{d}{dt} \begin{pmatrix} v2 \\ v3 \end{pmatrix} = - \begin{pmatrix} G1+G2 & -G2 \\ -G2 & G2+G3 \end{pmatrix} \begin{pmatrix} v2 \\ v3 \end{pmatrix} + \begin{pmatrix} G1 \\ 0 \end{pmatrix} vin \qquad \text{(c)}$$

$$\begin{pmatrix} G1 & -G1 & 0 & 1 \\ -G1 & (G1+G2+Geq1) & -G1 & 0 \\ 0 & -G2 & (G2+G3+Geq2) & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} v1 \\ v2 \\ v3 \\ I_{in} \end{pmatrix} = \begin{pmatrix} 0 \\ Ieq1 \\ Ieq2 \\ Vin \end{pmatrix} \qquad \text{(d)}$$

where,

$$I_{eq}k = i(t) + \frac{2Ck}{\Delta t} v_{Ck}(t) \quad G_{eq}k = \frac{2Ck}{\Delta t}$$

Figure 3: (a)An R-C network. (b)Its equivalent circuit with companion models for capacitors used for stamping of the MNA matrix. (c)State-space equation for the R-C network. The states are the voltage values at the nodes of the capacitor. (d) MNA matrix corresponding to (b).

After extracting the models, we generate the reduction matrix(V) using Krylov and TBR based methods [9] (line 7 of Algorithm 1). and generate the reduced order models (lines 8-10).

3.2 Model Pruning After Training

At this point, we have reduced linearizations for each of the sampled trajectory points from training. Our problem is that we may have *too many* such points for efficient interpolation. Earlier efforts typically show results with a few tens of such points; we routinely generate 10,000 such points with more rigorous training. Thus, as a next step, we propose to *prune* these linearizations, removing those we are less likely to find useful.

After the generation of the reduced linearizations (Steps 8-10 of Algorithm 1), we look for *similar* linearizations in our training database. We define two linearizations (i.e., two reduced matrices) to be *similar* if (a) the trajectory points about which they were generated are sufficiently close, and (b) the normalized distance between their state-space matrices (L2 norm) is less than some value ϵ. More precisely, linearizations matrices G_{ir}, A_{ir} and G_{jr}, A_{jr} corresponding to two sample points x_i and x_j are said to be similar if

$$\frac{||x_i^r - x_j^r||}{||x_i^r||} \leq \epsilon \quad and \tag{6}$$

$$\frac{||G_{ir} - G_{jr}||}{||G_{ir}||} \leq \delta_1, \frac{||A_{ir} - A_{jr}||}{||A_{ir}||} \leq \delta_2 \quad for \quad \epsilon, \delta_1, \delta_2 > 0 \tag{7}$$

where $|| \cdot ||$ is the standard L_2 vector/matrix norm, i.e. the componentwise sum of squared element values. Even when we train a large set of systematically generated waveforms, there are inevitably many instances when the circuit moves through an already visited region of the state space. Hence, sample-points in such regions provide no new information about the circuit dynamics. Model pruning is intended to reduce this redundant information. To prune, we look for *similar* linearizations in the trajectory database. Given a well-defined similarity metric, this is just a standard clustering/classification problem, solvable via a variety of data mining techniques. The result of the clustering step is a medium sized set of clusters (eg. for 10,000 points, 20-30 large clusters.) This is done via Gaussian Mixture Models based on Bayes classifiers and expectation maximization [18]. Roughly speaking, Gaussian mixture models (GMMs) optimize the likelihood that any given collection of data points are generated by a mixture of Gaussian distributions located appropriately in the state space. We choose how many mixtures we want to use; the algorithm places/orients/shapes the Gaussians so that the natural clusters in the data are each covered by one Gaussian.

Once the data-points are clustered into chunks of size 300-500, each of the clusters is reclassified into smaller clusters of size n where n is usually between 2-10. This is done using *agglomerative clustering*. It is a method for hierarchical clustering where we start with single element clusters and then progressively refine each cluster by merging the closest elements using a distance metric (*Euclidean* in our case). The method is similar to the well known Kruskal's algorithm for generating a minimum spanning tree by greedy merging of forests. We use the *average linkage* form of agglomerative clustering, wherein we represent a cluster having more than one element by a point which is the centroid of its member elements. We use this two-step strategy - GMM based clustering into a few large clusters, followed by fine-grain agglomerative clustering because the quadratic complexity $O(N^2)$ of purely agglomerative clustering is too inefficient for large sets of trajectory points.

Once we have these smaller clusters, we try replacing all the points of these small clusters by a new linearization whose sample point and state-space matrices are the simple, element-wise arithmetic means of all the linearizations in that particular cluster. We add this *average* sample point to our macromodel and check for *similarity* criteria between this new point and all the members of its corresponding cluster. If the *similarity* criteria is met, we throw away the member sample-point, otherwise, we keep it as a part of our macromodel.

3.3 Efficient Interpolation

Even after the model pruning step, the number of linearizations in our macromodel is large. Hence, it is computationally inefficient to use Equation 5 in its direct form for simulation of the macromodel. Computing $w_i's$ would take $O(N)$ time which means that the model evaluation time would grow linearly with respect to the number of linearized models. Another point worth noting in Equation 4 is that the final weights are very skewed towards the points very close to z_0. In fact, during simulations it was observed that only the first 5-10 closest points had any observable numerical weights associated with them. The rest all have their $w_i's$ set to 0 due to the highly centered kernel weighting function (Equation 4).

Therefore, it is much more efficient to compute the weights and generate the state-space equations using *a few nearest neighbors*, rather that using all of the s linearized models. In other words, since the distance weighting using Equation 4 already "zeroes" most of the trajectory linearizations that are sufficiently far away from new state-space point x, we propose, for efficiency, that we should instead only interpolate from a suitably chosen set of, say, k-nearest neighbors. High-dimensional nearest neighbor (NN) lookup is a well studied area. Indeed, the most obvious solution is to employ a *k-d* tree [19] to reduce the interpolation complexity from linear to logarithmic. However, we can go further, and reduce the practical complexity even more by employing *approximate nearest neighbor* (ANN) lookup strategies. ANN schemes trade-off determinism for speed: quickly returns all nearest neighbors within distance d if we allow some "tolerance" on the value of d. To be precise, given a set of n points $P = \{p_1, ..., p_n\}$ in d-dimensional space, any Minkowski distance metric $d(p, q)$ denoting the distance between two points and any real $\epsilon > 0$; $p \in P$ is defined as an ϵ-approximate nearest neighbor of the query point q if

$$d(p, q) \leq (1 + \epsilon)d(p', q) \quad \forall p' \in P \tag{8}$$

It has been shown [20] that the ϵ-approximate k-nearest neighbors of q can be computed in $O(klogn)$ for $k \geq 1$ but the hidden costs in this complexity analysis are much smaller.

In our case, since, only the k ($\sim$ 5-10) nearest neighbors have non-zero weights associated with them. Thus, if we query ANN for $k'(\sim 20) > k$ nearest neighbors, the result of this conservative query will include the k nearest neighbors as well, with very high probability. We can thus use Equation 4 with s set to k' to compute the weights and then to finally generate the state-space equation at z_0 using Equation 5. During implementation, this approach turned out to be very efficient. For a 10-fold increase in the number of linearized models, the search time for k-nearest neighbor increased by less than a factor of 2.

3.4 Incremental Model Update

An attractive, and somewhat surprising side-benefit of the trajectory based methodology is the ability to support *incremental* model updates at negligible cost. That is, if the macromodel's output fails to produce the same waveform as that of the target circuit for a particular input waveform due to insufficient training, we can simply *add* the linearization points for the problematic input waveform to our model. This will *update* the macromodel to handle the relevant non-visited regions of the circuit's state-space during inadequate prior training. We do not have to retrain and rebuild the entire macromodel from scratch.

3.5 Handling Hierarchy

Our implementation of the macromodeling methodology into SPICE3 also allows for hierarchical simulation. Once a macromodel for a particular circuit has been extracted, it can be inserted back into a new system-level SPICE netlist as a replacement to the transistor level circuit. For this, a new SPICE element similar to resistor, capacitor etc. has been created for the macromodel. The macromodel element is treated like a Voltage Controlled Voltage Source (VCVS) by the primary SPICE engine during stamping of the MNA matrix [16]. Figure 4 shows the block-diagram for the macromodel's model evaluation function. The primary SPICE engine passes the input voltage and present time to the model evaluation function. The macromodel stores its position in the state-space at the previous successful time-point. With the help of Approximate Nearest Neighbor queries and Equation 5, it computes the interpolated state-space equation. Then, using the time-step information from the primary SPICE engine, it solves the differential equation based on trapezoidal integration using a custom dense matrix solver. The solution pushes the current state-vector to a new point in the state-space and produces an output. Also, since Equation 5 is a function of the current position in the state-space as well, thus, we need to iterate to converge to a solution for a particular time-point t. Once, convergence is achieved, the output voltage value is passed back to the SPICE engine to be stamped in its MNA matrix. In case of non-convergence, a flag is set which forces the primary SPICE engine to reduce its time step. [16]

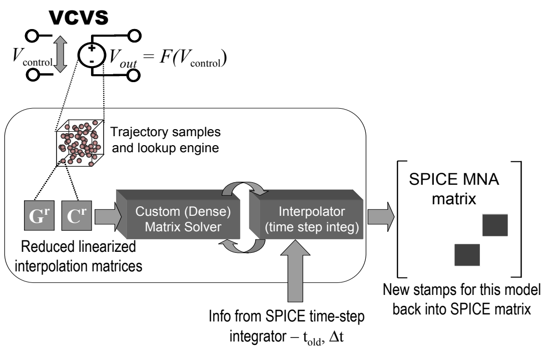

VCVS
$V_{control}$
$V_{out} = F(V_{control})$
Trajectory samples and lookup engine
G^r
C^r
Custom (Dense) Matrix Solver
Interpolator (time step integ)
SPICE MNA matrix
Reduced linearized interpolation matrices
New stamps for this model back into SPICE matrix
Info from SPICE time-step integrator – t_{old}, Δt

Figure 4: Block diagram showing how a generated macromodel gets simulated as a circuit element inside SPICE.

The present implementation of the model for system-level simulation does have one notable gap: we do not support models for input/output loading for the re-inserted VCVS element. Our current work focuses on efficient lookup/interpolation schemes for Z_{in} and Z_{out} as a function of the circuit's location in its state-space.

4. EXPERIMENTAL RESULTS

The complete macromodeling methodology as has been discussed in the previous sections has been implemented into SPICE3. To the best of our knowledge, this is the first SPICE-level implementation of trajectory methods with full BSIM3 support. Also, this is the first time where the user can re-insert the generated trajectory macromodel into a SPICE netlist for using the macromodel for faster system-level simulations in the same SPICE engine.

We present three trajectory macromodel results in this section: a complex opamp, a hierarchical circuit in which the opamp is replaced by its macromodel and a small example of easy incremental model update. First, however, we return to the still open problem of how to train these models in a more systematic way. We need a rigorous training scheme to create a robust macromodel. For this, we use a series of input waveforms with different shapes (*viz.* sinusoids, square-waves and chirps), frequencies and amplitudes. The amplitudes and frequencies for training are selected by predicting the range of waveforms that the circuit would encounter in practice. For example, an opamp designed for a 2.5V power supply would be trained across a range of input waveforms from 1μV to 1.25V (assuming it is biased at the mid-point). Prior knowledge of more restrictive input waveform would result in fewer sampled points in state-space (linearized models) and hence better speed-efficient macromodels.

The circuit under test (Figure 5) is a differential folded-cascode opamp with common-mode feedback stage (CMFB). Figure 6 compares the output of the circuit and the macromodel under the presence of power supply noise. The trained macromodel had $\sim$10000 linearization points which were reduced to $\sim$5000 through model

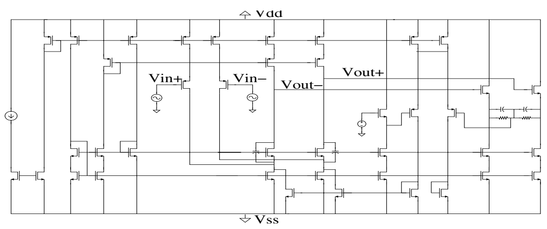


Figure 5: Circuit schematic of the folded-cascode opamp.

pruning. The model generation and model pruning took around 2 hours on a 1.6GHz machine with 256MB of RAM. The input to the circuit is a 0.02V, 100kHz saw-tooth test-waveform. To stress the model, we add power supply noise as a 40mV sinusoid at 10MHz. As can be seen from the zoomed-in picture (Figure 6), the output waveform produced by the macromodel matches that of the original transistor level circuit almost perfectly. We observed a speedup of $\sim$9.4X. The transistor circuit used 11.3% of the machine's 256MB RAM. The maximum memory usage by the macromodel with the complete set of ($\sim$10K) linearization points was 55.0%.

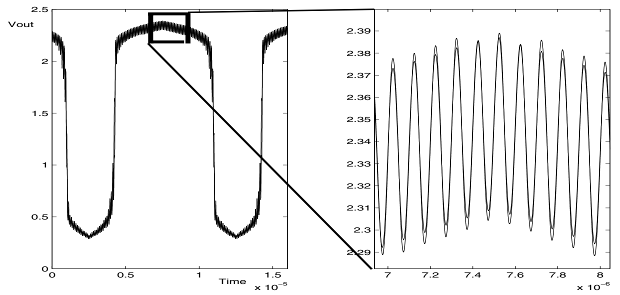


Figure 6: Plot comparing the output waveforms for the folded-cascode opamp of Figure 5 and its macromodel in the presence of power-supply noise. The zoomed in plot on the right shows good match between the two waveforms.

Table 1 highlights the importance of a scalable interpolation scheme. We show the inner loop time (in milliseconds (ms)) taken to interpolate the dynamics for one time point in the previous opamp circuit for three differently sized trajectory databases. The raw training data produces $\sim$ 10k points, we can use our pruning to reduce this to $\sim$ 5k points. A different, much more abbreviated training run is used to generate the model with $\sim$ 1k points. As we can see, the approximate nearest neighbor strategy is about one order of magnitude faster than the conventional *k-d* tree and three orders of magnitude faster than the linear lookup proposed in the original development of the trajectory method.

Table 1: CPU-time comparisons for the interpolation schemes.

Number of Points	Linear Interpolation (ms)	k-d Tree (ms)	ANN (ms)
975	28.67	1.31	0.23
4833	142.79	2.89	0.25
9653	272.28	4.86	0.30

As the second experiment, we now show that a macromodel can be used as a replacement for the transistor-level circuit in a system level context. The circuit which is used is a sample-and-hold circuit from a pipelined analog to digital converter. It uses the CMFB opamp (Figure 5) as one of its constituent blocks. Two simulations are performed. One with transistors in all the blocks and the other with all circuit elements kept the same except for the opamp which is replaced by our trajectory macromodel. The waveforms at the output of the sample-and-hold block have been plotted in Figure

7. As can be seen, there is very close agreement between the two simulation outputs.

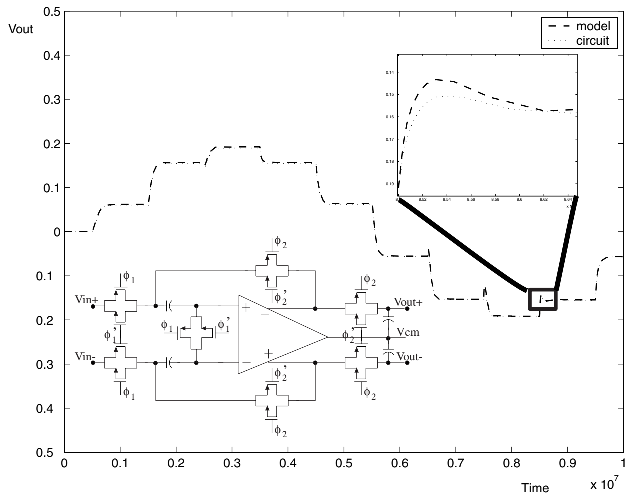


Figure 7: Waveform comparisons for the sample and hold output with the opamp replaced by its macromodel in one of the circuits. The waveforms are almost indistinguishable.

Due to the presence of other circuit elements along with both the circuit and transistor level opamp in this system-level simulation example, smaller speed-up gains ($\sim 3.8X$) were observed. The simulation speed-up numbers are expected to increase with the increase in the size of the circuit being macromodelled. Also, improvements in our dense matrix solver should result in better simulation run-times. Presently, around 80% of the simulation time is spent in model interpolation and matrix solve.

For our final experiment, we trained a macromodel for a simple two-stage opamp circuit (Figure 8(a)) with input waveforms with frequencies upto 100kHz. However, during testing, we used an input sinusoid of frequency 10MHz. As can be seen from Figure 8(b), the output waveforms differ for the circuit and the macromodel. This was because the high frequency test waveform excited the circuit to regions of its state-space where the macromodel had no representative linearization points. In the second pass, we added the linearization points corresponding to the failing high frequency input waveform to our macromodel and then simulated both the circuit and the macromodel. From Figure 8(c), we can observe that the two waveforms are indistinguishable.

5. CONCLUSIONS

We have presented a scalable trajectory-based modeling methodology for generating on-demand macromodels for analog circuits, and the first implementation of a complete trajectory "infrastructure" inside a SPICE engine with full BSIM3 support. The approach supports arbitrarily large training runs, automatically prunes redundant trajectory samples, supports limited hierarchy, enables incremental macromodel updates, and gives 3-10X speedups for larger circuits. Our ongoing work focuses on both algorithmic and engineering implementation improvements to our core trajectory engine. We believe, this work will be an excellent platform from which to propagate trajectory models into more widespread use.

6. ACKNOWLEDGMENTS

We would like to thank Jacob White and Dmitry Vasilyev (MIT) and Paolo Nenzi (University of Rome) for the fruitful discussions that have helped shape this work to its present form.

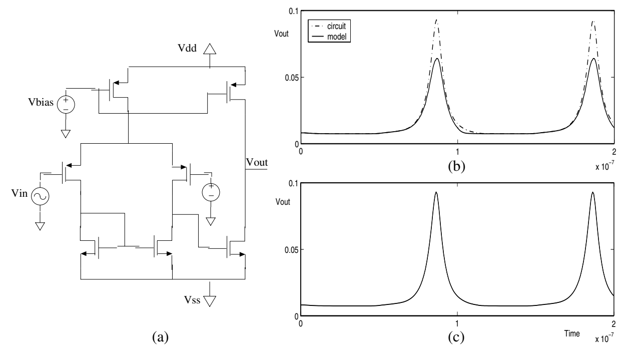


Figure 8: (a) Simple two-stage opamp circuit schematic (b) Circuit and macromodel's output waveform for incomplete training (c) Circuit and macromodel's output waveform after incremental model update.

7. REFERENCES

[1] M.Takahashi, K.Ogawa, and K.Kundert. VCO jitter simulation and its comparision with measurements. In *ASP-DAC*, 1999.

[2] S.K.Tiwary, S.Velu, R.A.Rutenbar, and T.Mukherjee. Pareto optimal modeling for efficient PLL optimization. In *Modeling and Simulation of Microsystems, Nanotech*, pages 195–198, 2004.

[3] G.G.E.Gielen and R.A.Rutenbar. Computer-aided design of analog and mixed-signal integrated circuits. In *Proc. of IEEE, Vol:88 Issue:12*, pages 1825–1854, 2000.

[4] L.T.Pillage and R.A.Rohrer. Asymptotic waveform evaluation. In *TCAD*, pages 352–366, 1990.

[5] A. Odabasioglu, M. Celik, and L.T. Pileggi. PRIMA: Passive reduced-order interconnect macromodeling algorithm. In *TCAD, Vol 17, No 8*, pages 645–654, 1998.

[6] P.Wambacq, G.Gielen, and W.Sansen. Interactive symbolic distortion analysis of analogue integrated circuits. *EDAC*, pages 484–488, 1991.

[7] Joel Phillips. Projection frameworks for model reduction of weakly nonlinear systems. In *DAC*, pages 184–189. ACM Press, 2000.

[8] Peng Li and L.T.Pileggi. NORM: compact model order reduction of weakly nonlinear systems. In *DAC*, pages 472–477, 2003.

[9] D.Vasilyev, M.Rewienski, and J.White. A TBR-based trajectory piecewise-linear algorithm for generating accurate low-order models for non-linear analog circuits and mems. *DAC*, pages 490–495, 2003.

[10] Michal Rewienski and Jacob White. A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices. In *TCAD*, pages 155–170, 2003.

[11] Ning Dong and J.Roychowdhury. Automated extraction of broadly applicable nonlinear analog macromodels from SPICE-level descriptions. In *CICC*, 2004.

[12] Ning Dong and J.Roychowdhury. Piecewise polynomial nonlinear model reduction. In *DAC*, pages 484–489, 2003.

[13] M.Rewienski and J.White. A trajectory piecewise linear approach to model order reduction and fast simulation of non-linear circuits and micromachined devices. *TCAD*, pages 155–170, 2003.

[14] T.Quarles. The SPICE3 implementation guide. In *UCB/ERL M89/44*, April 1989.

[15] R.W.Freund. Krylov-subspace methods for reduced order modeling in circuit simulation. *Journal of Computational and Applied Mathematics*, 2000.

[16] S.K.Tiwary. Scalable trajectory methods for on-demand analog macromodel extraction. In *Phd Thesis (in preparation), CMU*, 2005.

[17] Pillage, Rohrer, and Visweswariah. *Electronic circuit and system simulation methods*. McGraw-Hill, 1995.

[18] A.P.Dempster, N.Laird, and D.Rubin. Maximum-likelihood from incomplete data via the EM algorithm. In *J. of Royal Statistics Society, B39*, 1977.

[19] J.H.Friedman, J.L.Bentley, and R.A.Finkel. An algorithm for finding best matches in logarithmic expected time. In *ACM Trans. on Mathematical Software 3(3)*, pages 209–226, 1977.

[20] S.Arya, D.M.Mount, N.S.Netanyahu, R.Silverman, and A.Wu. An optimal algorithm for approximate nearest neighbor searching. In *ACM-SIAM Symp. on Discrete algorithms*, pages 573–582, 1994.