A GPU-Accelerated Envelope-Following Method for Switching Power Converter Simulation

Xue-Xin Liu*, Sheldon X.-D. Tan*, Hai Wang*, and Hao Yu†
*Dept. Electrical Engineering, University of California, Riverside, CA 92521
†School of Electrical & Electronic Engineering, Nanyang Technological University

***Abstract*—In this paper, we propose a new envelope-following parallel transient analysis method for the general switching power converters. The new method first exploits the parallelisim in the envelope-following method and parallelize the Newton update solving part, which is the most computational expensive, in GPU platforms to boost the simulation performance. To further speed up the iterative GMRES solving for Newton update equation in the envelope-following method, we apply the matrix-free Krylov basis generation technique, which was previously used for RF simulation. Last, the new method also applies more robust Gear-2 integration to compute the sensitivity matrix instead of traditional integration methods. Experimental results from several integrated on-chip power converters show that the proposed GPU envelope-following algorithm leads to about 10× speedup compared to its CPU counterpart, and 100× faster than the traditional envelop-following methods while still keeps the similar accuracy.**

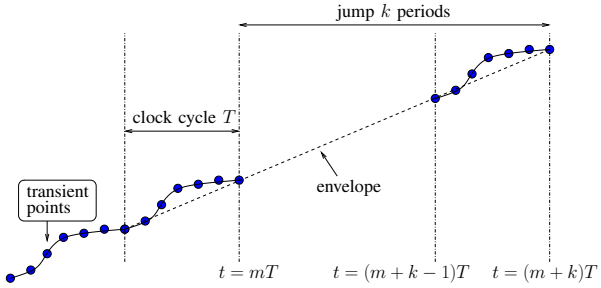
I. Introduction

Over the past decades, power electronics and especially switching power converters have seen a surge of new trends and novel applications — from the growing significance of PWM (pulse-width modulation) rectifiers and multilevel inverters to their widespread use in renewable (like solar and wind) energy systems, smart grids and emerging electric and hybrid vehicles [1].

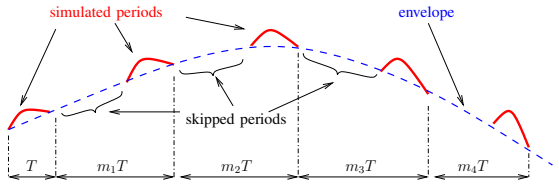
This requires more efficient simulation techniques for power electronics to meet the new application and demanding design constraints. To facilitate the design of typical power electronics circuits, many special-purpose simulation algorithms and tools were developed. Among them is the envelope-following method [2]–[4], which is able to calculate the slowly changing contour, or envelope, of a carrier waveform with a much higher switching frequency. This is the case for switching power converters, which have fast switching currents to convert powers from one level to another level. In those switching power converters, it is the envelope, which is the power voltage delivered, not the fast switching waves in every cycle, that is of interest to the designers. As shown in Fig. 1(a), the solid line is the waveform of the output node in a Buck converter [5], the dots are the simulation points of SPICE, and the appended dash line is the envelope.

Obviously, traditional SPICE will incur an extraordinarily high simulation time for this task, since it has to integrate the circuit's differential equation with many time points in every clock cycle to get the accurate details of the carrier. For switching power converters, the waveform of the carrier in consequent cycles does not change much, envelope-following method is an approximation analysis method, which skips over several cycles (the dash line in Fig. 1(b)), the so called envelope step, without simulating them, and then carries out a correction, which usually contains a sensitivity-based Newton iteration or shooting until convergence, in order to begin the next envelope step.

Also, iterative GMRES solver is typically used in the envelope-following method to compute the solution of Newton update due to its efficiency compared to direct LU method. However, as the Jacobian matrix or the sensitivity matrix in the equation to be solved is dense, explicit computing of the Jacobian is a very expense process. Recently, the matrix-free GMRES was proposed [6] for RF shooting based



(a) Illustration of one envelope skip.



(b) The envelope changes in a slow time scale.

Fig. 1. Transient envelope-following analysis. (Both two figures reflect backward-Euler style envelope-following.)

This research was supported in part by NSF grants under No. CCF-1017090 No. OISE-1051797, and No. OISE-0929699.

978-3-9810801-8-6/DATE12/©2012 EDAA

simulation. The new method leads to significant savings due to its implicit calculation of new basis vectors without the explicit formulation of the sensitivity matrix.

Modern computer architecture has shifted towards designs that employ so called multi-core processor or chip-multiprocessors (CMP) [7], [8]. The family of graphic processing units (GPU) are among the most powerful many-core computing systems in mass-market use [9]. For instance, the state-of-the-art NVIDIA Tesla T10 chip has a peak performance of over 1 TFLOPS versus about 80–100 GFLOPS of Intel i5 series Quad-core CPUs [10]. In addition to the primary use of GPUs in accelerating graphics rendering operations, there has been considerable interest in exploiting GPUs for general purpose computation (GPGPU) [11]. The introduction of new parallel programming interfaces for general purpose computation, such as Computer Unified Device Architecture (CUDA), Stream SDK, and OpenCL [12]–[14], has made GPUs an attractive choice for developing high-performance scientific computation tools and solving practical engineering problems. Hence, the applications with GPGPUs are rapidly growing in a broad variety of parallel numerical computation works [15].

In this paper, we propose a new parallel envelope-following method for the transient analysis of switching power converter circuits. The main contributions in our work include:

1) Parallelization of the Newton update, which is the most time consuming step, in the envelop-follow method, in GPU platforms.
2) Development of the new GMRES solver using matrix-free Jacobian-vector multiplication and the Gear-2 integration for sensitivity based Newton update equation.

The remainder of this paper is arranged as follows. After the basic algorithm of envelope-following is briefly reviewed and Newton update equation is derived in Section II. In Section III presents the new parallel envelope method where the CPU parallelization, matrix-free GMRES and Gear-2 integration will be discussed. Numerical examples are shown in Section IV, and finally, this paper is summarized in Section V.

II. Review of envelope following method

In transient analysis, a nonlinear circuit's behavior can be represented by a coupled set of nonlinear first-order differential algebraic equations (DAE) of the form

$$\frac{\mathrm{d}}{\mathrm{d}t}q(x(t)) + f(x(t)) = b(t), \tag{1}$$

where $x(t) \in R^N$ is the vector of circuit variables, usually comprising node voltages and possibly branch currents, $f(\cdot)$ is a nonlinear function that maps $x(t)$ to a vector of N entries most of which are sums of resistive currents at a node, $q(\cdot)$ maps $x(t)$ to a vector of N entries of capacitor charges or inductor fluxes, and $b(t)$ contains the input source values. For many power electronics circuits, the input switching signal is known and is periodic with clock period T.

Assume the time inside one period T has been discretised into M time steps, $0 = t_0 < t_1 < t_2 < \cdots < t_M = T$, and the i-th step length is $h_i = t_i - t_{i-1}$. In practice, the discretisation is nonuniform to control truncation error and convergence. Then, given an initial condition $x(0)$ at $t = 0$, numerical integration is applied to find the time domain solution of circuit state $x(t)$ at each time step till $t = T$.

For those circuits whose carrier waveforms vary slowly in a large number of periods, the envelope-following method only integrates the DAE in several selected periods and then jumps over to a new time point. By repeating this "simulate and skip" action, envelope-following method attains its efficiency compared to conventional transient analysis, but still can accurately obtain the slow varying envelope.

For example, if the clock period is T, and the suitable jump interval for the envelope is k periods, then the envelope step is kT. Suppose the state at time $t = mT$ is known as $x(mT)$, and the envelope-following wants to obtain the state at the next envelope step, $x((m+k)T)$. If the envelope step is much larger than the clock period (k is much bigger than one), envelope-following will lead to significant saving in simulation time.

To estimate $x((m+k)T)$, a forward-Euler style jumping relies only on $x(mT)$ and $x((m-1)T)$, i.e., $x((m+k)T) = x(mT) + k\left[x(mT) - x((m-1)T)\right]$. However, this approach is inefficient due to its restriction on envelope step k, since larger k usually causes instability. Instead, backward-Euler jumping, $x((m+k)T) - x(mT) = k\left[x((m+k)T) - x((m+k-1)T)\right]$, allows larger envelope steps. Here $x((m+k-1)T)$ is the unknown variable to be solved by Newton iteration, and $x((m+k)T)$ is dependent on $x((m+k-1)T)$ in each iteration. The Newton update equation in each iteration is thus expressed as

$$\begin{aligned}\Delta x((m+k-1)T) = A^{-1}\big[&(k-1)x^j((m+k)T) \\ &-kx^j((m+k-1)T) + x(mT)\big],\end{aligned} \tag{2}$$

where the Jacobian matrix A is computed as a combination of circuit sensitivity matrix and identity matrix, as

$$A = (k-1)\frac{\mathrm{d}x((m+k)T)}{\mathrm{d}x((m+k-1)T)} - kI = (k-1)J - kI. \tag{3}$$

In each Newton iteration, $x((m+k-1)T)$ is used as initial condition to calculate $x((m+k)T)$ by integrating the DAE in one clock period. Meanwhile, the conductance matrix $G_i = \mathrm{d}f(x(t_i))/\,\mathrm{d}x(t_i)$ and the capacitance matrix $C_i = \mathrm{d}q(x(t_i))/\,\mathrm{d}x(t_i)$ at each time step are used to derive the sensitivity matrix $J = \mathrm{d}x((m+k)T)/\,\mathrm{d}x((m+k-1)T)$.

Different integration rules can be applied to the computation of sensitivity matrix. It can be easily derived that, if the DAE is integrated with backward-Euler rule, the sensitivity is

$$J = \frac{\mathrm{d}x_M}{\mathrm{d}x_0} = \prod_{i=1}^{M}\left(\frac{1}{h_i}C_i + G_i\right)^{-1}\frac{1}{h_i}C_{i-1}$$

In summary, the envelope-following method is fundamentally a boosted version of traditional transient analysis, with certain skips over several periods and a Newton iteration to update or correct the errors brought by the skips, as is exhibited by Fig. 2.

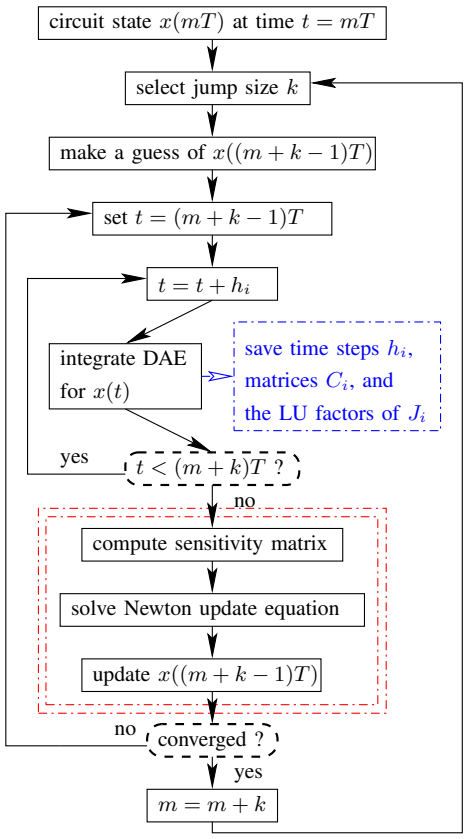


Fig. 2. The flow of envelope-following method.

III. New Parallel Envelop-Following Method

In this section, we explain how to efficiently use matrix-free GMRES to solve the Newton update problems with implicit sensitivity calculation, i.e., the steps enclosed by the double dashed block in Fig. 2. Then implementation issues of GPU acceleration will be discussed in detail. Finally, the Gear-2 integration is briefly introduced.

A. GMRES Solver for Newton update Equation

Generalized Minimum Residual (GMRES) method is an iterative method for solving systems of linear equations ($Ax = b$) with dense matrix A. The standard GMRES is given in Algorithm 1. It constructs a Krylov subspace with order m,

$$\mathcal{K}_m = \text{span}(b, Ab, A^2b, \ldots, A^{m-1}b),$$

where the approximate solution x_m resides. In practice, an orthonormal basis V_m that spans the subspace $\mathcal{K}_m$ can be generated by the Arnoldi iteration. The goal of GMRES is to search for an optimal coefficient y such that the linear combination $x_m = V_m y$ will minimize its residual $\|b - Ax_m\|_2$. The Arnoldi iteration also creates a by-product, an upper Hessenberg matrix $\tilde{H}_m \in R^{(m+1)\times m}$. Thus, the projection of A on the orthonormal basis V_m is described by the Arnoldi decomposition $AV_m = V_{m+1}\tilde{H}_m$, which is useful to check the residual at each iteration without forming x_m, and to solve for coefficient y when residual is smaller than a preset tolerance [16].

Algorithm 1 A standard GMRES

Input: $A \in \mathbb{R}^{N\times N}$, $b \in \mathbb{R}^N$, and initial guess $x_0 \in \mathbb{R}^N$
Output: $x \in \mathbb{R}^N$: $\|b - Ax\|_2 < tol$

1: $r = b - Ax_0$
2: $h_{1,0} = \|r\|_2$
3: $m = 0$
4: **while** $m < max\_iter$ **do**
5: $m = m + 1$
6: $v_m = r/h_{m,m-1}$
7: $r = Av_m$
8: **for** $i = 1 \ldots m$ **do**
9: $h_{i,m} = \langle v_i, r\rangle$
10: $r = r - h_{i,m}v_i$
11: **end for**
12: $h_{m+1,m} = \|r\|_2$
13: Compute the residual ϵ
14: **if** $\epsilon < tol$ **then**
15: Solve the problem: minimize $\|b - Ax_m\|_2$
16: Return $x_m = x_0 + V_m y_m$
17: **end if**
18: **end while**

At a first glance, the cost of using standard GMRES directly to solve the Newton update equation (2) seems to come mainly from two parts: the formulation of the sensitivity matrix $J = \mathrm{d}x_M/\mathrm{d}x_0$ in Eq. (9) in Section III-C, and the iteration inside the standard GMRES, especially the matrix-vector multiplication and the orthonormal basis construction (Line 7 through Line 12 in Algorithm 1). Based on the observation that only the matrix-vector product is required in GMRES, the work in [6] introduces an efficient matrix-free algorithm in the shooting-Newton method, where the equation solving part also involves with a sensitivity matrix. The matrix-free method does not take an explicit matrix as input, but directly passes the saved capacitance matrices C_i and the LU factorizations of J_i, $i = 0, \ldots, M$, into the Arnoldi iteration for Krylov subspace generation. Therefore, it avoids the cost of forming the dense sensitivity matrix and focuses on subspace construction. Briefly speaking, Line 7 will be replaced by a procedure without explict A, and we will talk about the flow of matrix-free generation of new basis vectors in later sections.

B. Parallelization on GPU platforms

There exist many GPU-friendly computing operations in GMRES, such as the vector addition (`axpy`), 2-norm of vector (`nrm2`), and sparse matrix-vector multiplication (`csrmv`), which have been parallelized in the CUDA Basic Linear Algebra Subroutine (CUBLAS) Library and the CUDA Sparse Linear Algebra Library (CUSPARSE) [17].

GPU programming is typically limited by the data transfer bandwidth as GPU favors computationally intensive algorithms [10]. So how to efficiently transfer the data and wisely partition the data between CPU memory and GPU memory is crucial for GPU programming. In the following, we discuss these two issues in our implementation.

As noted in Section II, the envelope-following method

requires the matrices gathered from all the time steps in a period in order to solve a Newton update. At each time step, SPICE has to linearize device models, stamp matrix elements, and solve circuit equations in its inner Newton iteration. When convergence is attained, circuit states are saved and then next time step begins. This is also the time when we store the needed matrices for the envelope-following computation. Since these data are involved in the calculation of Gear-2 sensitivity matrix in the generation of Krylov subspace vectors in Algorithm 2, it is desirable that all of these matrices are transferred to GPU for its data parallel capability.

To efficient transfer those large data, we explores asynchronous memory copy between host and device in the recent GPUs (Fermi architecture), so that the copying overlaps with the host's computing of the next time step's circuit state. The implementation of asynchronous matrices copy includes two parts: allocating page-locked memory, also known as pinned memory, where we save matrices for one time step, and using asynchronous memory transfer to copy these data to GPU memory. While it is known that page-locked host memory is a scarce resource and should not be overused, the demand of memory size of the data generated at one time step can be generously accommodated by today's mainstream server configurations.

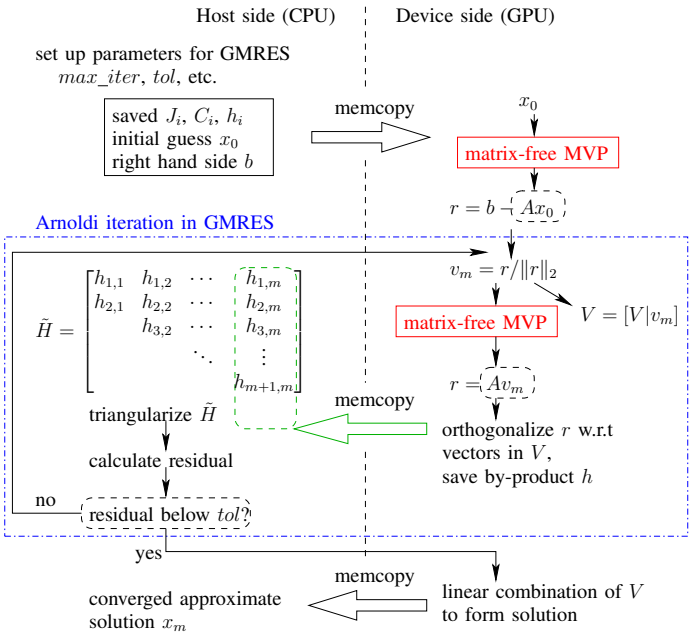


Fig. 3. GPU parallel solver for envelope-following update.

The second issue is to decide the location of data between CPU and GPU memories. Therefore let us first make a rough sketch of the quantities in the GMRES Algorithm 1. Although GMRES tends to converge quickly for most circuit examples, i.e., the iteration number $m \ll N$, the space for storing all the subspace basis V_m of N-by-m, i.e., m column vectors with N-length, is still big. In addition, every newly generated matrix-vector product needs to be orthogonalized with respect to all its previous basis vectors in the Arnoldi processes. Hence, keeping all the vectors of V_m in GPU global memory allows GPU to handle those operations, such as inner-product of basis vectors (`dot`) and vector subtraction (`axpy`), in parallel.

On the other hand, it is better to keep the Hessenberg matrix $\tilde{H}$, where intermediate results of the orthogonalization are stored, at the host side. This comes with the following reasons. First, its size is $(m+1)$-by-m at most, rather small if compared to circuit matrices and Krylov basis. Besides, it is also necessary to triangularize $\tilde{H}$ and check the residual regularly in each iteration so the GMRES can return the approximate solution as soon as the residual is below a preset tolerance. Hence, in consideration of the serial nature of the trianularization, the small size of Hessenberg matrix, and the frequent inspection of values by host, it is preferable to allocate $\tilde{H}$ in CPU (host) memory. As shown in Fig. 3, the memory copy from device to host is called each time when Arnoldi iteration generates a new vector and the orthogonalization produces the vector h.

C. Gear-2 based Sensitivity Calculation

The Gear-2 integration method is a backward differentiation formula (BDF), which approximates the derivative of a function using information from past few steps. Gear-2 method has been shown to be more suitable for many practical problems such as stiff problems where circuit behavior is affected by different time constants (fast ones with large poles and slow ones with small poles) [18]. Switching power converters and RF systems are typically stiff systems as waveforms changing in two different time scales are involved.

Given the DAE (1), at the i-th time step, Gear-2 approximates the derivative $\mathrm{d}q(x(t))/\mathrm{d}t$ by a two-step backward finite difference, $\mathrm{d}q(x(t_i))/\mathrm{d}t = \frac{1}{h_i}\left[q(x(t_i)) - \alpha_1^i q(x(t_{i-1})) - \alpha_2^i q(x(t_{i-2}))\right]$, where the coefficients α's are chosen to satisfy Gear's backward differentiation formula [19]. For uniformly discretised time steps, the index i in h_i, α_1^i and α_2^i can be omitted.

For the first step t_1, only the initial condition at t_0 is available. Therefore backward-Euler is used, i.e.,

$$\tfrac{1}{h_1}[q(x_1) - q(x_0)] + f(x_1) = b_1. \tag{4}$$

Since x_0 directly affects the solution of x_1, the sensitivity matrix up to now is

$$\frac{\mathrm{d}x_1}{\mathrm{d}x_0} = \left[\frac{1}{h_1}C_1 + G_1\right]^{-1}\frac{C_0}{h_1} = J_1^{-1}\frac{C_0}{h_1}. \tag{5}$$

Let J_i denote $(1/h_i)C_i + G_i$ in the remaining part of this paper. In addition, the LU factorizations of J_i are already calculated since they are used to solve the circuit state at each time step before we calculate the sensitivity.

Next, for the solution at t_2, with the previous two steps information available, Gear-2 integration can be applied,

$$\tfrac{1}{h_2}[q(x_2) - \alpha_1 q(x_1) - \alpha_2 q(x_0)] + f(x_2) = b_2. \tag{6}$$

In view of the sensitivity of x_2 with respect to the changes of x_0, (6) indicates that x_2's perturbation can be traced back to x_0 along two paths: x_2 is directly affected by x_0, and x_2 is also affected indirectly by x_0 via x_1. That is,

$$\frac{\mathrm{d}x_2}{\mathrm{d}x_0} = \frac{\partial x_2}{\partial x_1}\frac{\mathrm{d}x_1}{\mathrm{d}x_0} + \frac{\partial x_2}{\partial x_0},$$

where the two partial derivatives are

$$\frac{\partial x_2}{\partial x_1} = J_2^{-1}\frac{\alpha_1}{h_2}C_1, \qquad \frac{\partial x_2}{\partial x_0} = J_2^{-1}\frac{\alpha_2}{h_2}C_0,$$

and $\mathrm{d}x_1/\mathrm{d}x_0$ is calculated previously in (5). Thus, the sensitivity matrix deduced from (6) is

$$\frac{\mathrm{d}x_2}{\mathrm{d}x_0} = J_2^{-1}\frac{\alpha_1}{h_2}C_1 \cdot J_1^{-1}\frac{C_0}{h_1} + J_2^{-1}\frac{\alpha_2}{h_2}C_0. \tag{7}$$

Likewise, for the third time step t_3, apply the Gear-2 integration formula to DAE (1),

$$\tfrac{1}{h_3}[q(x_3) - \alpha_1 q(x_2) - \alpha_2 q(x_1)] + f(x_3) = b_3, \tag{8}$$

and the chain rule for sensitivity is

$$\frac{\mathrm{d}x_3}{\mathrm{d}x_0} = \frac{\partial x_3}{\partial x_2}\frac{\mathrm{d}x_2}{\mathrm{d}x_0} + \frac{\partial x_3}{\partial x_1}\frac{\mathrm{d}x_1}{\mathrm{d}x_0} = J_3^{-1}\left[\frac{\alpha_1}{h_3}C_2\frac{\mathrm{d}x_2}{\mathrm{d}x_0} + \frac{\alpha_2}{h_3}C_1\frac{\mathrm{d}x_1}{\mathrm{d}x_0}\right],$$

where both $\mathrm{d}x_2/\mathrm{d}x_0$ and $\mathrm{d}x_1/\mathrm{d}x_0$ are ready from the previous two time steps, i.e., Eqs. (7) and (5). Follow this chain rule in the aforementioned recursive style, the sensitivity matrix for Gear-2 integration is computed along the remaining time steps up to the M-th step,

$$J = \frac{\mathrm{d}x_M}{\mathrm{d}x_0} = J_M^{-1}\left[\frac{\alpha_1}{h_M}C_{M-1}\frac{\mathrm{d}x_{M-1}}{\mathrm{d}x_0} + \frac{\alpha_2}{h_M}C_{M-2}\frac{\mathrm{d}x_{M-2}}{\mathrm{d}x_0}\right]. \tag{9}$$

Note that as the matrix-free GMRES method is applied, which only requires matrix-vector multiplication and no explicit J is required, as explained in Algorithm 2.

With matrix-free method, the matrix-vector multiplication (Line 7 in Algorithm 1) is replaced by the iteration shown in Algorithm 2, which calculates the multiplication product of the Gear-2 sensitivity we encounter in envelope-following and a basis vector in the Krylov subspace. Note that the scaling and shift of J in $A = (k-1)J - kI$, as described in Eq. (3), is taken into consideration by Line 8.

Algorithm 2 Matrix-free method for Krylov subspace construction

Input: current Krylov subspace basis vector v, time step lengths h_i, saved C_i matrices and LU factors of J_i, $i = 0, \ldots, M$

Output: matrix-vector product r, such that $r = Av$

1: solve $J_1 p_2 = h_1^{-1} C_0 v$ for p_2
2: solve $J_2 p_1 = h_2^{-1}(\alpha_1 C_1 p_2 + \alpha_2 C_0 v)$ for p_1
3: **for** $i = 3 \ldots M$ **do**
4: solve $J_i p_0 = \alpha_1 h_i^{-1} C_{i-1} p_1 + \alpha_2 h_i^{-1} C_{i-2} p_2$ for p_0
5: $p_2 = p_1$
6: $p_1 = p_0$
7: **end for**
8: $r = (k-1)p_0 - kv$

For the matrix-free generation of new basis vectors, it is straightforward to apply CUBLAS and CUSPARSE routines, and some customized GPU kernel functions to implement Algorithm 2 with the stored LU matrices of J_i and C_i mentioned earlier. For example in Line 4, as the iteration index i traverses all the M time points' matrix information, sparse matrix-vector multiplication `csrmv` is first called to calculate $C_{i-1}p_1$ and $C_{i-2}p_2$. And after the two resulted vectors are combined by `axpy` of CUBLAS, p_0 is solved for by `trsv`, since as we noted before, J_i is already in LU factorization form when transient simulation at step i finished.

IV. Experimental results

All the experiments in this paper have been carried out on a server which has an Intel Xeon quad-core CPU with 2.0 GHz clock speed, and 24 GBytes memory. The GPU card mounted on this server is NVIDIA's Tesla C2070 (Fermi), which contains 448 cores (14 MPs $\times$ 32 cores per MP) running at a 1.30 GHz and has 4 GBytes on-chip memory.

The envelope-following method with the proposed Gear-2 sensitivity matrix computation is added to an open-source SPICE, implemented in C [20]. Our envelope-following program is implemented by following the algorithm mentioned in [21]. To solve the Newton update equation, different methods are used to compare the computation time, such as direct LU, GMRES with explicitly formed matrix, and GMRES with implicit matrix-vector multiplication (matrix-free). Moreover, the matrix-free method is also incorporated to the same SPICE simulator using CUDA C programming interface, as described in Section III-B.

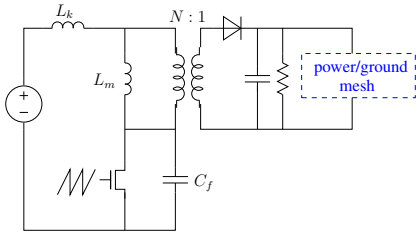


Fig. 4. Diagram of a zero-voltage quasi-resonant flyback converter.

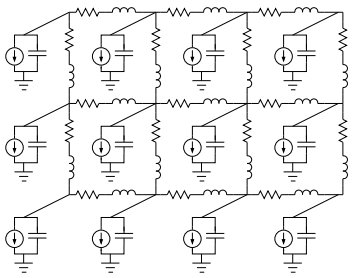
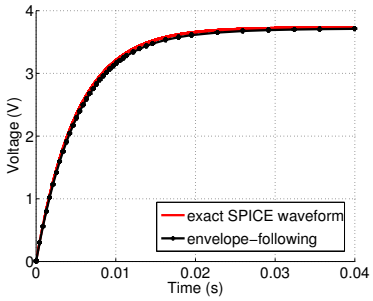


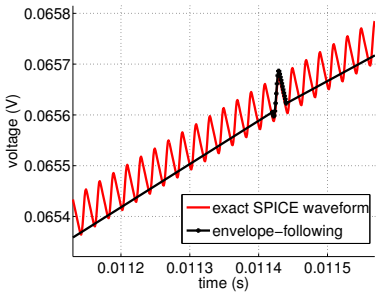
Fig. 5. Illustration of power/ground network model.

We use several integrated on-chip converters as simulation examples to measure running time and speedup. They include a Buck converter, a quasi-resonant flyback converter (shown in Fig. 4), and two boost converters. Each converter is directly integrated with on-chip power grid networks, since the performance of converters should be studied with their loads and we can easily observe the waveforms at different nodes in a power grid (see Fig. 5 for a simplified power grid structure).

Fig. 6 shows the waveform at output node of the resonant flyback converter. Note that on the envelope curve, the darker



(a) The whole plot



(b) Detail of one EF simulation period

Fig. 6. Flyback converter solution calculated by envelope-following. The red curve is traditional SPICE simulation result, and the back curve is the envelope-following output with simulation points marked.

TABLE I
CPU AND GPU TIME COMPARISONS (IN SECONDS) FOR SOLVING NEWTON UPDATE EQUATION WITH THE PROPOSED GEAR-2 SENSITIVITY.

Circuit	Nodes	Direct LU	Explicit GMRES	Implicit GMRES		
				CPU	GPU	X
Buck	910	423.8	420.3	36.8	3.9	9.4×
Flyback	941	462.4	459.6	64.5	7.4	8.7×
Boost-1	976	695.1	687.7	73.2	6.2	11.8×
Boost-2	1093	729.5	720.8	71.0	8.5	9.9×

dots in separated segments indicate the real simulation points were calculated in those cycles, and the segments without dots are the envelope jumps where no simulation were done. It can be verified that the proposed Gear-2 envelope-following method produces a envelope matching the original waveform well.

For the comparison of running time spent in solving Newton update equation, Table I lists the time costed by direct method, explicit GMRES, matrix-free GMRES, and GPU matrix-free GMRES. All methods carry out the Gear-2 based envelope-following method, but they handle the sensitivity and equation solving in different implementation steps. It is obvious that as long as the sensitivity matrix is explicitly formed, such as the cases in direct method and explicitly GMRES, the cost is much higher than the implicit methods. When matrix-free technique is applied to generate matrix-vector products implicitly, the computation cost is greatly reduced. Thus, for the same example, implicit GMRES would be one order of magnitude faster than explicit GMRES. Furthermore, our GPU parallel implementation of implicit GMRES makes this method even faster, with a further 10× speedup.

V. CONCLUSION

A new envelope-following method for transient analysis of switching power converters has been introduced. First, the computationally expensive step, the solving of Newton update equation, has been parallelized on CUDA-enabled GPU platforms with iterative GMRES solver to boost performance of the analysis method. To further speed up the GMRES solving for Newton update equation, we have employed the matrix-free Krylov basis generation technique. The proposed method also applies the more robust Gear-2 integration to compute the sensitivity matrix. Experimental results from several integrated on-chip power converters have shown that the proposed GPU envelope-following algorithm can lead to about 10× speedup compared to its CPU counterpart, and 100× faster than the traditional envelop-following methods while still keeps the similar accuracy.

REFERENCES

[1] Andrzej M. Trzynadlowski. *Introduction to Modern Power Electronics*. Wiley, second edition, 2010.
[2] K. Kundert et al. An envelope following method for the efficient transient simulation of switching power and filter circuits. In *Proc. ICCAD*, pages 446–449, Oct. 1988.
[3] J. White and S. Leeb. An envelope-following approach to switching power converter simulation. *IEEE Trans. Power Electron.*, 6(2):303–307, Apr. 1991.
[4] P. Feldmann and J. Roychowdhury. Computation of circuit waveform envelopes using an efficient, matrix-decomposed harmonic balance algorithm. In *Proc. ICCAD*, pages 295–300, Nov. 1996.
[5] P. Krein. *Elements of Power Electronics*. Oxford University Press, 1997.
[6] R. Telichevesky, K. Kundert, and J. White. Efficient steady-state analysis based on matrix-free Krylov-subspace methods. In *Proc. Design Automation Conf. (DAC)*, 1995.
[7] Intel Corporation. Intel multi-core processors, making the move to quad-core and beyond (White Paper), 2006. http://www.intel.com/multi-core.
[8] AMD Inc. Multi-core processors—the next evolution in computing (White Paper), 2006. http://multicore.amd.com.
[9] NVIDIA Corporation, 2011. http://www.nvidia.com.
[10] David B. Kirk and Wen-Mei Hwu. *Programming Massively Parallel Processors: A Hands-on Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, 2010.
[11] Dominik Göddeke. General-purpose computation using graphics hardware. http://www.gpgpu.org/, 2011.
[12] NVIDIA Corporation. CUDA (Compute Unified Device Architecture), 2011. http://www.nvidia.com/object/cuda_home.html.
[13] AMD Inc. AMD Steam SDK. http://developer.amd.com/gpu/ATIStreamSDK, 2011.
[14] Khronos Group. Open Computing Language (OpenCL). http://www.khronos.org/opencl, 2011.
[15] CUDA community showcase. http://www.nvidia.com/.
[16] G. H. Golub and C. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 3rd edition, 1996.
[17] NVIDIA, 2011. http://developer.nvidia.com/cuda-toolkit-40.
[18] J. Vlach and K. Singhal. *Computer Methods for Circuit Analysis and Design*. Van Nostrand Reinhold, New York, NY, 1995.
[19] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1971.
[20] NGSPICE. http://ngspice.sourceforge.net/.
[21] T. Kato et al. Envelope following analysis of an autonomous power electronic system. In *IEEE COMPEL'06*, pages 29–33, July 2006.