GPU-Friendly Floating Random Walk Algorithm for Capacitance Extraction of VLSI Interconnects

Kuangya Zhai, Wenjian Yu and Hao Zhuang
Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
Email: gxiverson@gmail.com, yu-wj@tsinghua.edu.cn, zhuangh@ucsd.edu

***Abstract*—The floating random walk (FRW) algorithm is an important field-solver algorithm for capacitance extraction, which has several merits compared with other boundary element method (BEM) based algorithms. In this paper, the FRW algorithm is accelerated with the modern graphics processing units (GPUs). We propose an iterative GPU-based FRW algorithm flow and the technique using an inverse cumulative probability array (ICPA), to reduce the divergence among walks and the global-memory accessing. A variant FRW scheme is proposed to utilize the benefit of ICPA, so that it accelerates the extraction of multi-dielectric structures. The technique for extracting multiple nets concurrently is also discussed. Numerical results show that our GPU-based FRW brings over 20X speedup for various test cases with 0.5% convergence criterion over the CPU counterpart. For the extraction of multiple nets, our GPU-based FRW outperforms the CPU counterpart by up to 59X.**

I. Introduction

Under the nanometer process technology, accurate extraction of interconnect capacitance with 3-D field-solver algorithm becomes increasingly important for high-performance integrated circuit (IC) design. The field-solver algorithm for capacitance extraction can be classified as two categories: the conventional deterministic methods such as boundary element method (BEM) [1-3], finite element method (FEM), etc., and the floating random walk (FRW) algorithm with stochastic nature [4-6].

A 2-D FRW algorithm for capacitance extraction was firstly published in 1992 [4], whose basic idea is to convert the calculation of conductor charge to the Monte Carlo (MC) integration described with floating random walks. The FRW algorithm has the advantages of lower memory usage, more scalability for large structures and tunable accuracy, compared with the deterministic methods. To date, the FRW algorithm has evolved to several commercial capacitance solvers (e.g. the QuickCap™). Recently, the FRW algorithm regained the interest from the research community due to the increasing importance of capacitance field solver for large-scale structures. In 2008, a general FRW algorithm was proposed for arbitrary dielectric configuration, which numerically calculates the transition probabilities (Green's function) online rather than offline [5]. This technique largely reduces the number of transition hops in FRW, and therefore the total computational time, with the overhead of calculating and storing the transition probabilities for a lot of transition domains. This strategy will become inefficient for general large-scale structures. In 2012, the efficient finite-difference techniques were presented to numerically characterize the Green's function and weight value for 3-D cube transition domain [6]. With the pre-calculated Green's function and weight value for the cubes with two-layer dielectrics, the FRW is able to achieve the good trade-off of efficiency and memory usage for general 3-D problems.

The prominent advance of multi-/many core processors has brought large opportunities for accelerating the computational intensive tasks of capacitance extraction. The parallel algorithms on multi-core CPU have been proposed to speed up the multipole-based BEM [7] or the FRW for 3-D capacitance extraction [6]. The recently developed general purpose graphics processing units (GPUs) integrate hundreds of cores into a single chip, and thus has much higher computing throughput than the multi-core CPU. If suitably developed, the parallel computing on GPU would be more energy-efficient and promising to accelerate the computing-intensive workload. However, there is rare achievement for leveraging the GPU-based parallel computing to tackle the grand challenges of larger-scale 3-D capacitance extraction, though the GPU-friendly algorithm has become a popular research topic in the EDA community [8, 9]. As the first attempt, the algorithm "FMMGpu" was proposed in 2011 to accelerate the multipole-based BEM for capacitance extraction on GPU [9]. For some single-dielectric bus crossing structures, the FMMGpu has demonstrated 22X to 30X speedups compared to the serial CPU computing.

Although the FRW algorithm is embarrassingly parallel due to the data independency among walks, it is not easy to be parallelized on GPU. Firstly, the randomness of FRW results in workload divergence among different walk paths. Secondly, due to the GPU's single-instruction-multiple-data (SIMD) computing scheme [10], the conditional branches in the FRW algorithm greatly harm the computational efficiency. Lastly, the long latency of accessing GPU's global memory forbids frequently reading the pre-calculated tables of Green's function, which is inevitable in the FRW, especially for the multi-dielectric problem. In this paper, for the first time we propose a GPU-friendly FRW algorithm flow which efficiently minimize the divergence of operations and alleviate the bottleneck of global memory. A variant FRW scheme is proposed for multi-dielectric extraction problem. And, the task of extracting multiple nets is accelerated by utilizing the GPU's concurrent kernel execution mode. Numerical experiments validate the efficiency of proposed techniques. For various extraction modes and test cases, the GPU-based FRW outperforms the CPU-based FRW with over 20X speedups.

978-3-9815370-0-0/DATE13/©2013 EDAA

II. PRELIMINARIES

The fundamental formula of the FRW algorithm is:

$$\phi(r) = \oint_S P(r, r^{(1)})\phi(r^{(1)})dr^{(1)} \quad , \tag{1}$$

where $\phi(r)$ is the electric potential at point r, and S is a closed surface surrounding r. $P(r, r^{(1)})$ is called the Green's function. For a fixed r, $P(r, r^{(1)})$ can be regarded as the probability density function (PDF) for selecting a random point $r^{(1)}$ on S. In this sense, $\phi(r)$ can be estimated by the mean value of $\phi(r^{(1)})$, providing sufficient large number of random samples on S are evaluated. If S is the surface of a homogeneous cube centered at r, $P(r, r^{(1)})$ only depends on the relative position of $r^{(1)}$, and is not related to the size of cube. This Green's function can be derived analytically [4], and pre-calculated and stored as the discrete probabilities for jumping to the discretized cells of the cube surface.

In the case that $\phi(r^{(1)})$ is unknown, we apply (1) recursively to obtain the following nested integral formula:

$$\begin{aligned}\phi(r) = &\oint_{S^{(1)}} P^{(1)}(r, r^{(1)})\oint_{S^{(2)}} P^{(2)}(r^{(1)}, r^{(2)})\cdots \\ &\oint_{S^{(k+1)}} P^{(k+1)}(r^{(k)}, r^{(k+1)})\phi(r^{(k+1)})dr^{(k+1)}\cdots dr^{(2)}dr^{(1)} \quad ,\end{aligned} \tag{2}$$

where $S^{(i)}$, (i=1,⋯,k+1) is the ith cubic surface with center at $r^{(i-1)}$. $P^{(i)}$, (i=1,⋯,k+1), are the Green's functions relating the potentials at $r^{(i-1)}$ to $r^{(i)}$. This can be interpreted as a floating random walk procedure: for the ith hop of a walk, the maximum homogeneous cube centered at $r^{(i-1)}$ is constructed and then a point $r^{(i)}$ is randomly selected on the cube surface according to the discrete probabilities obtained with $P^{(i)}$. Note that in a single-medium problem, the expression of $P^{(i)}$ does not change with i, if the corresponding ith cube is normalized to a unit-size cube. The walk terminates after k hops if the potential at point $r^{(k)}$ is known, e.g. it is on a conductor surface in the problem of capacitance extraction. With the Green's function and sampling probabilities for a unit-size cube calculated in advance, the major computational cost of a walk is for geometric operations.

For extracting capacitances among multiple conductors, the relationship between conductor charge and potential is needed. So, a Gaussian surface G_j is constructed to enclose conductor j, and according to the Gauss theorem,

$$Q_j = \oint_{G_j} D(r)\bullet\hat{n}(r)dr = \oint_{G_j} F(r)(-\nabla\phi(r))\bullet\hat{n}(r)dr \quad , \tag{3}$$

where Q_j is the charge on conductor j, $F(r)$ is the dielectric permittivity at point r, and $\hat{n}(r)$ is the normal direction of G_j at r. Substituting (1) into (3), we derive:

$$Q_j = \oint_{G_j} F(r)g\oint_{S^{(1)}} \omega(r, r^{(1)})P^{(1)}(r, r^{(1)})\phi(r^{(1)})dr^{(1)}dr \quad , \tag{4}$$

where the weight value

$$\omega(r, r^{(1)}) = -\frac{\nabla_r P^{(1)}(r, r^{(1)})\bullet\hat{n}(r)}{gP^{(1)}(r, r^{(1)})} \quad . \tag{5}$$

Here ∇_r is the gradient operator with respect to r, and the constant g satisfies $\oint_{G_j} F(r)gdr = 1$. Now the first integral in (4) can be interpreted as a stochastic sampling procedure on G_j, and the second integral can be calculated with the above FRW procedure based on (2). The only difference is the extra weight value (5), which contributes to the estimating value of Q_j and can also be pre-calculated for the unit-size cube.

The above deduction relates the conductor charge with the conductor voltages (potential) through the FRW procedure. This derives the FRW algorithm for capacitance extraction (Algorithm 1). For the problem with multiple dielectrics, the Green's function and weight value (referred to as GFT and WVT respectively) for multi-dielectric cubic transition domain should also be computed in advance [4, 6].

Algorithm1: FRW algorithm for capacitance extraction (on CPU)

1: Load the pre-computed transition probabilities and weight values for single-dielectric or multi-dielectric cubic domain;
2: Construct the Gaussian surface enclosing master conductor j;
3: $C_{ji} := 0$; $npath := 0$;
4: **Repeat**
5: $npath := npath + 1$;
6: Pick a point $r^{(0)}$ on Gaussian surface, and then generate a cubic transition domain (may contain multiple dielectrics) T centered at it; pick a point $r^{(1)}$ on the surface of T according to the transition probabilities and then calculate the weight value ω with the help of the pre-computed weight values;
7: **Repeat**
8: Construct the largest cubic domain with known transition probabilities;
9: Pick a point on the domain surface, according to the transition probabilities;
10: **Until** the current point touches a conductor i
11: $C_{ji} := (C_{ji} \cdot (npath - 1) + \omega) / npath$
12: **Until** the stopping criterion is met

GPU is a highly parallel, many-core processor with tremendous computational horsepower and very high memory bandwidth. In a GPU, more transistors are devoted to data processing rather than data caching and flow control, hence it suits to address problems that can be expressed as data-parallel computations with high arithmetic intensity. The GPU memory system consists of registers, on-chip shared memory, and off-chip global memory. The shared memory resides in each streaming multi-processor (SM) and is much faster than the global memory. However, the size of shared memory is limited (e.g. 48 KB). The global memory has larger than 1GB size, but has long access latencies (several hundreds of clock cycles) and limited bandwidth, which often become the bottlenecks of many applications.

The common unified device architecture (CUDA) released by NVIDIA makes it possible to perform a general purpose computing on GPU. With CUDA specific extensions, the C language program can be modified to realize the GPU computing. The GPU code is organized as one or more kernel functions. A kernel launches a massive number of threads running concurrently on CUDA cores. The threads are grouped into thread blocks. During execution, a SM groups every 32 threads into a *warp*. The warp is the scheduling unit of coalescing memory access. Following the SIMD execution model, all the threads in a warp share the same instruction. Therefore, if divergent tasks are assigned to the threads within a warp, they will be executed serially.

III. Accelerating the FRW Algorithm on GPU

In this section, the techniques accelerating the FRW algorithm on GPU are proposed. They can be used for capacitance extraction, and other applications of FRW.

A. Task Dividing and the Iterative FRW Flow

A straightforward approach to parallelize FRW on GPU is to continuously feed each thread with a walk at a time, until the required accuracy is achieved. Due to the SIMD execution mode of GPU, the thread finishing current walk cannot start a new walk until all threads in a warp finish their walks. This leads to great waste of computing resource, because the number of hops in each walk can be very different.

In [12], a path regeneration approach is proposed for the random walk algorithm (i.e., path tracing) in photo-realistic image rendering problem. In this approach, when a thread finishes a walk, instead of waiting for other threads, it starts a new walk immediately. For the FRW for capacitance extraction, this strategy is not efficient, because before starting a new walk it needs to update capacitance results and get a new point on the Gaussian surface. When a thread is performing these operations, other threads need to stop and wait. And, the time for these operations is not trivial.

To improve the usage of computational resource, we propose the following task-dividing strategy. The FRW algorithm is divided into three kernels for specific tasks:

- ***Walk-starting kernel***. It includes choosing a point $r^{(0)}$ from the Gaussian surface, generating a maximum conductor-free cube $S^{(1)}$ centered at $r^{(0)}$, selecting a point $r^{(1)}$ on $S^{(1)}$, and calculating the weight value ω.
- ***Hop kernel***. It loads $r^{(1)}$ and ω generated by walk-starting kernel, repeatedly constructs the maximum conductor-free cube and chooses a point on it. Once the point hits a conductor, the number of the hit conductor is stored in the global memory.
- ***Reduction kernel***. It calculates the capacitance value with the walk results generated by hop kernel. The variance of the MC procedure is also updated for the estimation of capacitance error.

With this task dividing, the operation divergences and resource competition are largely reduced, as shown in Fig. 1. In the walk-starting kernel there is no divergence, because every thread follows the same execution path. For the third kernel, there are efficient parallel reduction algorithms based on the GPU architecture [11]. We can modify them for our reduction kernel without much effort. For the hop kernel, the divergent part is just several instructions of memory accessing. Hence, the efficiency of parallelization is improved.

A major advantage of FRW algorithm is its tunable accuracy. However, the task dividing strategy is not compatible to the accuracy control in the CPU-based FRW algorithm (Line 12 in Algorithm 1), because the hop kernel doesn't update the accuracy achieved on the fly. To enable the FRW on GPU to terminate based on the accuracy criterion, we propose an iterative FRW flow. Every time after the execution of the reduction kernel, we estimate the number of remaining walks needed to achieve the accuracy criterion based on the square root inverse relation between the error and the number of walks

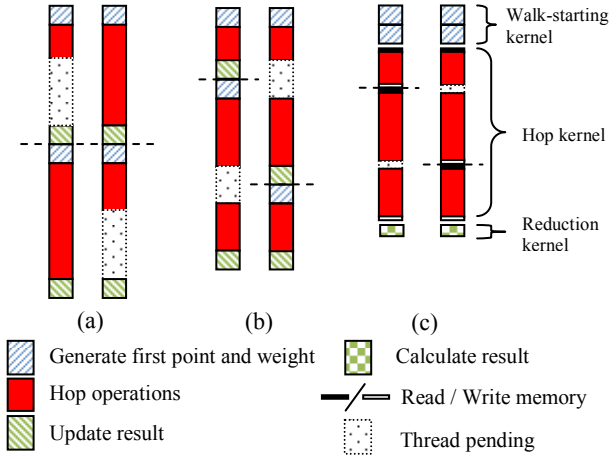


Fig. 1. The execution of FRW algorithm on two threads with three schemes: (a) straightforward scheme, (b) path regeneration, (c) the proposed task-dividing. The dot lines indicate the border of two walks. Note that the task-dividing strategy consumes the least time.

[4-6]. Suppose n_{cur} is the number of walks having been performed, we can estimate the number of walks n_{rem} for the next iteration to achieve the accuracy goal:

$$n_{rem} = n_{cur} \times \left(\frac{err_{cur}^2}{err_{goal}^2} - 1\right) \quad . \tag{6}$$

Here err_{cur} and err_{goal} are the current error with n_{cur} walks and the target error respectively. Note that most data for the three kernels resides on GPU, and only the capacitance values need to be copied to CPU for convergence judgment. The size of the transferred data is fairly small, thus the data communication time between CPU and GPU is negligible.

B. Exclusive Memory Space and Extra Start Points

The three kernels exchange data via the storage on the global memory. To prevent different threads from accessing the same memory location, every thread need to maintain a private space of memory for storing the exchanging data. Otherwise, either thread synchronization or memory lock is needed. Both them will harm the efficiency. We assign every thread an exclusive memory space on the global memory of GPU, at the beginning of the FRW algorithm.

Assume W_b is the number of walks assigned to a thread block. *BlockDim* is the number of threads in that thread block. W_t is the number of walks assigned to each thread. To achieve load balance, we assign the tasks to each thread evenly, i.e.,

$$W_t = \frac{W_b}{BlockDim} \quad . \tag{7}$$

During the hop kernel, there are some threads executing faster. If we generate exactly the same number of start points for each thread in the first kernel, i.e., $N_{sp}=W_t$, where N_{sp} is the number of start points for each thread, when faster threads finish W_t walks, they have to wait for slower threads. Because the hop kernel consumes most of the computing time, we propose to generate margin start points for each thread. This enables the fast threads performing more walks. So,

$$N_{sp} = r_{extra} \times W_t \quad , \tag{8}$$

where $r_{extra} > 1$ is the extra factor. Now, in the hop kernel, a thread terminates when either it uses up N_{sp} start points or its

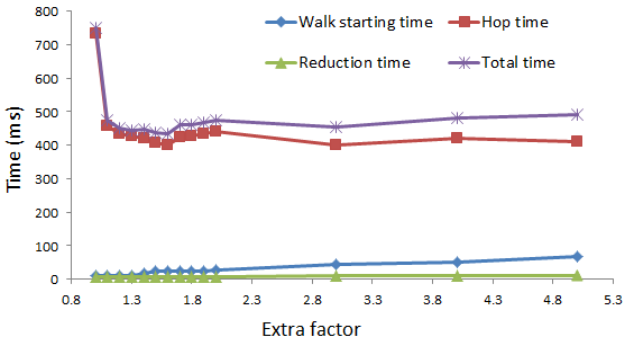


Fig. 2. The relation between the extra factor and run time of each kernel. The test case is Case 4 in Section V, with 1 million walks.

block finishes W_b walks. The larger is r_{extra}, the fewer is the waiting time for a block, with the price of generating more extra start points. Fig. 2 shows that at most 40% speedup can be obtained if setting r_{extra} properly. In our experiments, we set $r_{extra} = 2$, which is nearly optimal for most cases.

With exclusive memory space for each thread and the generation of extra start points, Fig. 3 shows the flowchart of hop kernel. Array1 is used to store start points generated by walk-starting kernel. Array2 is used to store results of each walk, which are to be loaded by the reduction kernel. Vertical dot lines indicate the borders of the exclusive memory space for each thread.

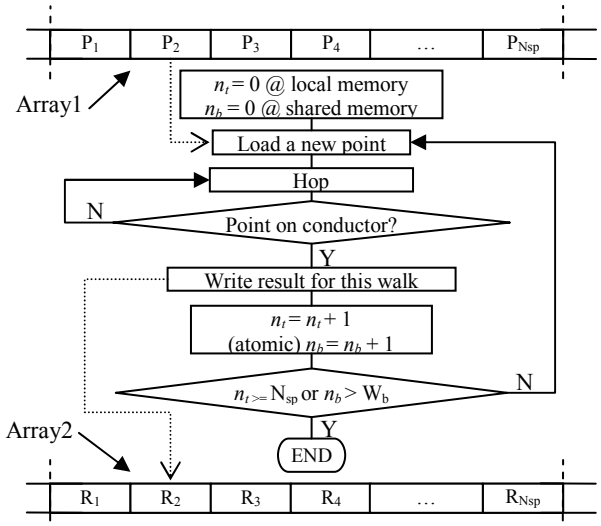


Fig. 3. The flowchart of the hop kernel. n_t and n_b are the number of walks performed by a thread and a block respectively.

C. Hop with the Inverse Cumulative Probability Array

There are usually thousands of hops in the FRW algorithm. Each hop requires loading the pre-stored GFT from global memory, which causes long latency for GPU. A main operation in each hop is to pick a point on the cube surface according the transition probabilities. With the GFT storing the jumping probabilities to the surface panels, the inverse transform sampling is used to translate a uniform random number to a panel index. Suppose there are N surface panels, and the transition probabilities for the panels are p_i, i=1, …, N. To facilitate the computation, the cumulative summation of p_i's is calculated in advance:

$$s_i = \sum_{j=1}^{i} p_j \ , \ i = 1, \ \ldots, \ N \ , \tag{9}$$

The left problem is to determine the index of panel (i) for a given uniform random number $s_i \in [0, 1]$. This is usually accomplished by a binary search on the array of $\{s_i\}$. Because the transition probabilities are stored in global memory of GPU, due to the space limit of on-chip memory, the approach of binary search suffers from the long memory access latency. On the other hand, binary search is inherently divergent, whose performance is very poor for SIMD platforms.

With the foregoing concerns of binary search, we propose to design an inverse cumulative probability array (ICPA) to map the cumulative probability s_i to the index i. We choose the smallest probability as the resolution,

$$u_{res} = \min_{1 \le i \le N} \{p_i\} \ . \tag{10}$$

With it we allocate an integer array ICPA with length of $1/u_{res}$. We then build the array with the following two formulas:

$$t_i = \lfloor p_i / u_{res} + 0.5 \rfloor \ , \ i = 1, \ \ldots, N \ , \tag{11}$$

$$\mathrm{ICPA}[\sum_{i=0}^{k-1} t_i + 1 : \sum_{i=0}^{k-1} t_i + t_k] = k, \ \ (k = 1, 2, ..., N) \ . \tag{12}$$

While picking a point, a uniform random number s in [0,1] is firstly generated. Then, ICPA[$\lfloor s/u_{res}+0.5 \rfloor$] is just the index for the panel. With this technique, FRW can target directly to the panel for the next hop. In the implementation on GPU, ICPA largely reduces the time of accessing the global memory as well as the operation divergence of binary search. As for the overhead, the ICPA corresponding to the single-dielectric Green's function needs about 10MB (for the usual discretization of transition cube with about 5000 panels), which is obviously affordable.

IV. Challenges and Techniques for Actual Multi-Dielectric Capacitance Extraction

In this section, we discuss the challenges and techniques for applying FRW for actual capacitance extraction.

A. The Challenge for the Multi-Dielectric Problem

In modern VLSI process technology, interconnects are embedded in multiple layered dielectrics. Numerical techniques are used to characterize the Green's function and weight value for transition cube involving multi-layer dielectrics (referred to as multi-dielectric GFT and WVT), so that the walk can cross the interface and therefore reduces the computation.

When migrating the multi-dielectric FRW to GPU, the main challenge is the excessive global memory accessing due to the following reasons: (1) The ICPA technique proposed in Section III.C cannot be applied to the multi-dielectric GFT directly because it needs excessive memory usage for various dielectric configurations. Hence, when the transition cube contains multi-dielectrics, it resorts to binary searching to get a point on the surface of the cube. (2) The number of hops in the extraction of multi-dielectric structure is larger than that of single-dielectric structure, which leads to more times of global memory accessing.

B. The Usage of a Variant FRW Scheme

To reduce the global memory accessing in picking a point on the multi-dielectric cube surface, we modify (4) to be:

$$Q_j = \oint_{G_j} F(r) g \oint_{S^{(1)}} \omega(r, r^{(1)}) \frac{P^{(1)}(r, r^{(1)})}{P_0(r, r^{(1)})} P_0(r, r^{(1)}) \phi(r^{(1)}) dr^{(1)} dr, \quad (13)$$

where $P^{(1)}(r, r^{(1)})$ is the multi-dielectric Green's function, i.e., the probability of choosing a point from the surface of $S^{(1)}$. $P_0(r, r^{(1)})$ stands for the Green's function of single-dielectric cube. Defining

$$\omega'(r, r^{(1)}) = \frac{P^{(1)}(r, r^{(1)})}{P_0(r, r^{(1)})} \quad (14)$$

to be an extra weight, we have:

$$Q_j = \oint_{G_j} F(r) g \oint_{S^{(1)}} \omega(r, r^{(1)}) \omega'(r, r^{(1)}) P_0(r, r^{(1)}) \phi(r^{(1)}) dr^{(1)} dr. \quad (15)$$

With (15), the sampling procedure on the first cube can be performed with the single-dielectric Green's function. This can also be applied to the subsequent hops according to the nested integral formula similar to (2). We call this *the variant FRW scheme*, whose difference to the standard FRW is that for each hop an extra weight ω' should be multiplied. Note that the variant scheme itself doesn't accelerate FRW on GPU, but it enables the use of ICPA for multi-dielectric problem, which reduces the operation divergence and global memory accessing. The extra weight can be pre-calculated and stored in advance, for a given multi-dielectric configuration, which just replaces the memory usage for the multi-dielectric GFTs, and does not induce extra memory usage.

The variant FRW scheme increases the variance of MC procedure, and thus slows down its convergence. However, the efficiency benefit of the variant FRW overwhelms its drawback, as demonstrated in Section V.

C. Concurrent Extraction of Multiple Nets

For the actual application of FRW, instead of the single-net capacitance extraction (with a single master conductor), we often need to extract the capacitances of multiple nets in a same design. Since the multi-dielectric GFTs and WVTs are characterized only once for a given process technology, the FRW algorithm can be easily extended to handle the multi-net extraction problem, without extra table-loading procedure.

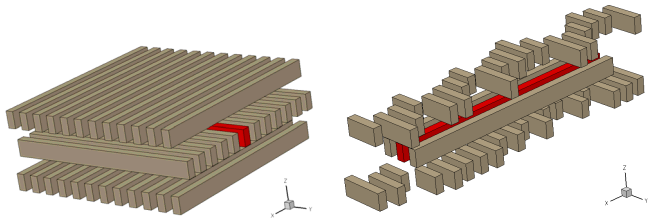
With the proposed three-kernel scheme which largely reduces the divergence, the bottleneck of our parallel FRW algorithm on GPU becomes the latency of global memory accessing. CUDA has the warp switching technique to hide the global memory latency with computations. Compared with the strategy of extracting the nets one by one, higher GPU computing efficiency can be achieved for the multi-net extraction by utilizing the concurrent kernel execution mode of the Nvidia GPU [10]. This technique allows performing several single-net extraction tasks on GPU concurrently, which makes more computing threads assigned to every SM.

For the concurrent extraction of multiple nets, each single-net extraction task follows the iterative FRW flow proposed in Section III. Once the numbers of remaining walks are calculated for all tasks, we assign the computing resource of GPU to each task accordingly to balance the workload among the concurrent executed kernels. The numerical results in Section V.C validate the efficiency of our GPU-based FRW for the multi-net extraction.

V. Numerical Results

We have implemented the 3-D FRW algorithm for capacitance extraction in C++. The techniques with multi-dielectric GFT and WVT [6] are adopted to handle the structure with multiple layers of dielectrics. Numerical results are presented in this section to compare the proposed GPU-based FRW algorithm with the corresponding CPU version. All experiments are carried out on a PC with 2.70GHz dual-core Pentium CPU, 6GB memory, and one Nvidia GTX580 GPU with 1.5GB global (device) memory.

We consider the 45nm process technology (see Fig. 8 in [6]), and several representative structures of VLSI interconnects. The first case is a two layer cross-over structure, with 4 wires on M1 and M2 each. The second and third cases are three layer cross-over structures, with 6 and 12 wires on each layer respectively. The fourth case is a structure with 41 wires, where three parallel wires are on M2 layer, and 19 wires are randomly distributed on M1 and M3 layer each. Fig. 4 shows the bird-eye view of the third and fourth cases.



(a) (b)

Fig. 4. The bird-eye view of two test cases: (a) Case 3, (b) Case 4. The red conductor is the critical net to be extracted.

A. Comparison of Different Parallel Schemes

We have implemented three different parallel schemes described in Section III.A. To compare their efficiency, we run the FRW algorithms for one million walks with different cases. We only consider the single-dielectric situation of our test cases, by simply removing the dielectrics. The results of FRW on CPU, and three versions of FRW on GPU are given in Table I. From the results we see that the path generation scheme in [12] is slower than the straightforward approach, and is not suitable for the FRW for capacitance extraction. This is because the operations for starting a new walk bring too much control flow divergence. The proposed three-kernel scheme is about 6X faster than the straight forward parallelization on GPU. While comparing the proposed FRW on GPU and FRW on CPU, the former shows more than 74X speedup for the test cases.

TABLE I
The Comparison of Different Parallel Schemes on GPU (with 10^6 walks each; unit of capacitance is 10^{-18} F)

Case	FRW-CPU		Straight-forward	Path re-generation	Three-kernel			
	Cap.	Time(ms)	Time(ms)	Time(ms)	Cap.	Time(ms)	Sp.[1]	Sp.[2]
Case 1	39.9	3520	286	607	40.1	47	**74**	**6.1**
Case 2	59.3	4285	299	710	59.3	49	**87**	**6.1**
Case 3	113	5347	322	895	113	59	**91**	**5.5**
Case 4	133	7910	369	1288	129	63	**126**	**5.9**

[1]: The speedup ratio to FRW-CPU.
[2]: The speedup ratio to the straightforward scheme on GPU.

B. Results for Multi-Dielectric Cases

In this section, we test the performance of the proposed variant FRW scheme, which enables the use of ICPA for multi-dielectrics problems. The convergence criterion of FRW is set to 0.5% 1-σ error, which is the default setting of commercial FRW-based capacitance field solver. It means that the error is less than 1.5% at the confidence level of 99.7%. The computational results are listed in Table II. "FRW-GPU standard" stands for the FRW algorithm on GPU which does not use the variant FRW scheme in Section IV.B.

TABLE II
The Computational Results on CPU and GPU for Multi-Dielectric Cases (0.5% accuracy criterion; unit of capacitance is 10^{-18} F)

Case	FRW-CPU			FRW-GPU standard			FRW-GPU Variant FRW + ICPA				
	# walk	Cap.	Time(s)	#walk	Cap.	Time(s)	# walk	Cap.	Time(s)	Sp.[1]	Sp.[2]
Case 1	606K	117	25.9	1088K	117	1.21	1800K	119	0.49	**53**	**2.5**
Case 2	507K	176	12.5	1015K	174	0.69	1613K	177	0.31	**40**	**2.2**
Case 3	483K	334	10.5	909K	336	1.16	1614K	334	0.39	**35**	**3.0**
Case 4	405K	353	7.84	590K	352	0.89	609K	355	0.36	**22**	**2.5**

[1]: The speedup ratio to FRW-CPU.
[2]: The speedup ratio to "FRW-GPU standard" without the variant FRW scheme.

From Table II we see that, with the variant FRW scheme and ICPA techniques, the proposed FRW on GPU is more than 22X faster than its CPU counterpart. Although the variant FRW scheme incurs more walks, it still outperforms the "FRW-GPU standard" with up to 3X speedup. This is because that the global memory accessing and control flow divergence of binary search are largely reduced. The data in Table I and II also validate that the proposed FRW algorithm on GPU keeps the same accuracy as the CPU-based FRW algorithm.

C. Concurrent Extraction of Multiple Nets

In this section, we consider the task of multi-net extraction in IC design flows. We generate a large case similar to Case 3 in Fig. 4(a), but with 140 wires in each layer. We assume the conductors in the middle layer are the critical nets need to be considered. We select different number of nets from the middle layer, and extract their capacitance concurrently with the technique in Section IV.C. The relation between number of concurrent nets and run time is listed as TABLE III. The serial extraction time of FRW on CPU is also listed.

TABLE III
The Computational Results for concurrent extraction of multiple nets for a large cross-over structure (0.5% accuracy criterion)

Number of nets to be extracted	CPU serial time (s)	GPU concurrent time (s)	Speedup
32	304	10.1	**30.1**
64	610	11.6	**52.6**
128	1246	21.0	**59.3**

As shown in TABLE III, the serial extraction time of CPU grows linearly with the number of nets to be extracted, while the time of GPU concurrent extraction grows slower. This property is quite favorable for the actual extraction task in IC design flow, where usually a lot of critical nets in a same design need to be considered.

VI. Conclusions

The advance of massively parallel computing brings a large opportunity to perform the highly accurate field-solver capacitance extraction for VLSI interconnects. The challenges of developing FRW on GPU platform is the control flow divergence and excessive global memory accessing. We propose a 3-kernel scheme, as well as the ICPA data structure to eliminate the control flow divergence. The extra start point generation scheme can reduce the waste of computational resource. The variant FRW scheme enables the use of ICPA for multiple dielectrics problems. The concurrent extracting of multiple nets can provide GPU with sufficient concurrent threads to hide the global memory accessing latency. With the techniques in this paper, it is promising to accelerate the extraction in IC design flows to improve design circle.

VI. Acknowledgement

The authors would like to thank Prof. Y. Deng, Tsinghua University, Beijing, China for many helpful discussions. This work is supported by National Natural Science Foundation of China under Grant No. 61076034, Tsinghua University Initiative Scientific Research Program, and Tsinghua National Laboratory for Information Science and Technology (TNList) Basic Research Project. W. Yu is the corresponding author.

References

[1] K. Nabors and J. White, "FastCap: A multipole accelerated 3-D capacitance extraction program," *IEEE Trans. CAD*, Vol. 10, pp. 1447–1459, Nov. 1991.

[2] W. Yu and Z. Wang, "Enhanced QMM-BEM solver for three-dimensional multiple-dielectric capacitance extraction within the finite domain," *IEEE Trans. MTT*, Vol. 52, No. 2, pp. 560-566, 2004.

[3] W. Chai, D. Jiao, and C.-K. Koh, "A direct integral-equation solver of linear complexity for large-scale 3D capacitance and impedance extraction," in *Proc. DAC,* July 2009, pp 752-757.

[4] Y. L. Le Coz and R. B. Iverson, "A stochastic algorithm for high speed capacitance extraction in integrated circuits," *Solid-State Electronics*, Vol. 35, No. 7, pp. 1005-1012, 1992.

[5] T. A. El-Moselhy, I. M. Elfadel, and L. Daniel, "A capacitance solver for incremental variation-aware extraction," in *Proc. ICCAD*, Nov. 2008, pp. 662-669.

[6] H. Zhuang, W. Yu, G. Hu, Z. Liu, and Z. Ye, "Fast floating random walk algorithm for multi-dielectric capacitance extraction with numerical characterization of Green's functions," in *Proc. ASP-DAC*, Jan. 2012, pp. 377-382.

[7] F. Gong, H. Yu, and L. He, "Picap: A parallel and incremental capacitance extraction considering stochastic process variation," in *Proc. DAC*, July 2009, pp. 764-769.

[8] H. Qian and Y. Deng, "Accelerating RTL simulation with GPUs," in *Proc. ICCAD*, Nov. 2011, pp. 687-693.

[9] X. Zhao and Z. Feng, "Fast multipole method on GPU: Tackling 3-D capacitance extraction on massively parallel SIMD platforms," in *Proc. DAC*, June 2011, pp 558-563.

[10] NVIDIA, *NVIDIA's Next Generation CUDA™ Computing Architecture: Fermi™, 2010.*

[11] NVIDIA, *NVIDIA GPU Computing SDK*, http://developer.nvidia.com/gpu-computing-sdk.

[12] J. Novak, V. Havran, C. Dachsbacher, "Path regeneration for random walks," in *GPU Computing GEMS(Emerald edition)*, W.-M. W. Hwu(eds.), Elsevier Inc. 2011. pp. 401-420.