Simulation Modelling Practice and Theory 34 (2013) 20–36

Contents lists available at SciVerse ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

# Accelerated floating random walk algorithm for the electrostatic computation with 3-D rectilinear-shaped conductors

Wenjian Yu [a,*], Kuangya Zhai [a], Hao Zhuang [a], Junqing Chen [b]

[a] Tsinghua National Laboratory for Information Science and Technology, Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China
[b] Department of Mathematical Sciences, Tsinghua University, Beijing 100084, China

ARTICLE INFO

*Article history:*
Received 8 November 2012
Received in revised form 19 December 2012
Accepted 6 January 2013

*Keywords:*
Capacitance calculation
Electrostatic computation
Floating random walk
GPU-based parallel computing
Importance sampling
Stratified sampling

ABSTRACT

With the advancement of fabrication technology, the electrostatic coupling has increasing impact on the performance of very large-scale integrated (VLSI) circuits and micro-electromechanical systems (MEMS). For the structures in VLSI circuits which are mostly rectilinear geometries, the floating random walk (FRW) method using cubic transition domains has been successfully applied to calculate the electric capacitances among interconnect wires. In this work, the techniques of importance sampling and stratified sampling are presented to accelerate the FRW algorithm by improving the convergence rate of the Monte Carlo procedure. An efficient approach is then presented to parallelize the FRW algorithm with the graphic processing units (GPUs). GPU-friendly algorithmic flow and data structure are designed to reduce the divergence among random walks and the time of accessing the device memory. The presented techniques are also applicable to the calculation of electric field intensity, which is also an important problem for nowadays nanometer-technology circuits. Numerical results are presented with several simple structures and larger ones from real VLSI circuits or MEMS. The results validate the accuracy of the presented techniques and demonstrate up to 100× speedup due to the variance reduction and GPU-based parallel computing.



## 1. Introduction

Although the laws of electrostatics have been known for a long time, nowadays the electrostatic computation is still of much importance in the areas of electrical/electronic engineering, biophysics, very-large scale integrated (VLSI) circuit, micro-electromechanical systems (MEMS), particle accelerator, etc. [1–6]. The methods for solving electrostatic problems also bring inspiration to the research of similar problems in mechanics, materials and other disciplines. The electrostatic field is governed by the Poisson/Laplace equation where electric potential is the unknown. The numerical methods for this problem are classified into two categories: deterministic methods and the Monte Carlo methods. The deterministic methods include the finite difference method (FDM), the finite element method (FEM), and the boundary element method (BEM). The FDM and FEM employ volume discretization of the domain, while BEM discretizes the boundary of domain and generates linear equations with fewer unknowns. For some three-dimensional (3-D) problems, the BEM [2,7–9] exhibits advantages over the FDM and FEM. Besides, the closed-form or empirical formulae are also used for calculating the interconnect capacitance in VLSI design.

* Corresponding author. Tel.: +86 10 6277 3440.
*E-mail address:* yu-wj@tsinghua.edu.cn (W. Yu).

1569-190X/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.simpat.2013.01.003

The Monte Carlo (MC) method, firstly introduced by von Neumann and Ulam as an approach of solving deterministic problems using random numbers, is often referred to as the *random walk method* when it is applied to the boundary value problems of elliptical partial differential equation. Compared with the deterministic methods, it is commonly agreed that the random walk method is most efficient when point values or linear functionals of the solution are needed [1]. While for a global solution or a high-accuracy solution is desired, it may lose its validity because of the quadratic dependence of its computing time on the demand of accuracy. The simulation time is actually a significant problem in some research, e.g. the applications in the design process of ICs. The random walk method can be classified into three categories [10]: (1) the discrete random walk (DRW) method (also called "fixed" random walk method) [11], (2) the walk inside the domain methods [1,3,12–16], and (3) the walk on the boundary (WOB) method [17–20]. The DRW method needs a fine-mesh grid imposed on the problem domain, such that a large number of steps are required for a walk to end at the boundary. This largely harms its computational efficiency. The floating random walk (FRW) method in [12], and the equivalent random "walk on spheres" (WOS) method in later literatures [1,13,14], are the representatives of the walk inside the domain methods. In the FRW method, the spherical or cubic transition domains with variable size are employed to reduce the steps in a walk, and thus the amount of computation. The spherical transition domain is suitable for general boundary geometry, which makes the WOS method widely applied. However, for some problems where there are mainly rectilinear geometries, using cubic transition domains has distinct advantage [15,16]. As shown in Fig. 1, the probability of touching the boundary with the cube is much larger than that with the sphere. The WOB method was originally proposed in [17], but was not applied to the calculation of electrostatic capacitance until recent years. In [19,20], the WOB method was used to compute the capacitance of the isolated unit cube with the highest accuracy. The results exhibit its advantage over other MC methods. However, the WOB method is limited to the simulation of a single convex body, and cannot be employed for general cases [19].

With the increase of circuit density and the demand of higher performance for VLSI circuit and MEMS, calculating the capacitances of interconnect wires fast and accurately becomes increasingly important. On the other hand, calculating the electric field intensity around wires is also very important for the nanometer-technology integrated circuits and MEMS. The prominent advance of multi-/many core processors has brought large opportunities for accelerating the FRW algorithm for capacitance calculation. The general purpose graphics processing units (GPUs) integrate hundreds of cores into a single chip, and thus have much higher computing throughput than the multi-core CPU. If suitably developed, the parallel computing on GPUs would be more energy-efficient and promising to accelerate the computing-intensive workload [22]. Recently, an algorithm was proposed to accelerate the multipole-based BEM for capacitance calculation on GPUs [31]. For some bus crossing structures in homogeneous medium, the GPU-based algorithm demonstrated 22× to 30× speedups compared to the serial CPU computing. However, there are some difficulties for parallelizing the FRW algorithm for calculating electric field intensity and capacitances. Firstly, the randomness of FRW results in the workload divergence among different walk paths. Secondly, due to the single-instruction-multiple-data (SIMD) computing scheme on GPUs, the conditional branching statements in the FRW algorithm greatly harm the efficiency of parallelization. Finally, the large latency of accessing GPU's device memory forbids frequently reading the pre-calculated tables of surface Green's function, which is inevitable in the FRW method for capacitance or electrostatic computation.

In this paper, the FRW algorithm which calculates the electric field intensity and capacitance, for the problems involving a considerable quantity of 3-D rectilinear-shaped conductors is investigated. Based on the FRW algorithm using transition cubes [15], the techniques of importance sampling and stratified sampling are proposed to improve the convergence rate of the FRW procedure. For a given accuracy criterion, they reduces the computing time by several times. Then, an efficient approach is presented to parallelize the FRW algorithm with the GPUs. A GPU-friendly FRW algorithm flow is proposed to efficiently minimize the divergence of operation and alleviate the bottleneck of the GPU device memory. Numerical experiments are carried out with small structures, such as the isolated unit cube and the parallel plates, and larger structures from real VLSI circuit and MEMS. The results demonstrate that the GPU computing brings more than 20× speedup over the serial CPU version of the same algorithm. For the capacitance computation of larger structures, the proposed techniques achieve the speedup ranging from 70× to 124×.

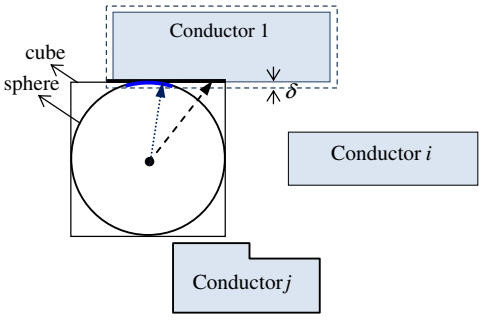


**Fig. 1.** With the cubic transition domain, the probability of termination for the walk becomes larger than that with the sphere. $\delta$ denotes the touching threshold for the FRW with spheres.

The paper is organized as follows. In Section 2, we introduce the FRW for electrostatic computation. In Section 3, the variance reduction techniques to quicken the convergence of MC procedure, i.e. the importance sampling and stratified sampling, are proposed for the FRW algorithm. In Section 4, we discuss how to make an efficient parallelization of the FRW algorithm on GPUs after a brief review of the GPU architecture. The numerical results and conclusions are presented in the last two sections.

## 2. The floating random walk algorithm using transition cubes

The fundamental formula of the FRW method is:

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

where $\phi(r)$ is the electric potential at point $r$, and $S$ is a closed surface surrounding $r$. The domain enclosed by $S$ is often called the transition domain, because Eq. (1) expresses the transition of a calculated quantity from the point inside the domain to its boundary. If the transition domain is a sphere or cube and the point $r$ is at the center of domain, the function $G_\phi(r, r^{(1)})$ (called the surface Green's function) may be derived analytically. Moreover, $G_\phi(r, r^{(1)})$ is of non-negative value for any point $r^{(1)}$ on $S$, and can be regarded as the probability density function (PDF) for selecting a random point on $S$. With the principle of MC simulation, $\phi(r)$ can be estimated as the statistical mean of $\phi(r^{(1)})$.

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

$$\phi(r) = \oint_{S^{(1)}} G_\phi^{(1)}(r, r^{(1)})dr^{(1)} \oint_{S^{(2)}} G_\phi^{(2)}(r^{(1)}, r^{(2)})dr^{(2)} \cdots \oint_{S^{(k+1)}} G_\phi^{(k+1)}(r^{(k)}, r^{(k+1)})\phi(r^{(k+1)})dr^{(k+1)}, \tag{2}$$

where $S^{(i)}$, $(i = 1, \ldots, k+1)$ is the boundary of the $i$th transition domain with center at $r^{(i-1)}$, and $r = r^{(0)}$. $G_\phi^{(i)}(r^{(i-1)}, r^{(i)})$, $(i = 1, \ldots, k+1)$, are the surface 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 $i$th hop of a walk, a transition domain centered at $r^{(i-1)}$ is constructed and then a point $r^{(i)}$ is randomly selected on its boundary according to the discrete probabilities obtained with $G_\phi^{(i)}(r^{(i-1)}, r^{(i)})$. The walk terminates after $k$ hops if the potential at point $r^{(k)}$ is known, e.g. it is on the surface of a conductor with known potential. The surface Green's function for a spherical transition domain has a simple analytical expression [12,13]. So, the FRW method with spheres is widely used and called "walk on spheres" (WOS) method accordingly. However, as we have mentioned previously, the spherical transition domain is not as suitable as the cubic domain, for the problem involving mostly rectilinear geometries (e.g. the Manhattan-shaped interconnects in VLSI circuit).

For a cubic transition domain including homogenous medium, the surface Green's function can be derived as the summation of a double-nested infinite series [15,16]. And because it only depends on the relative position of $r^{(1)}$, and is not related to the size of cube, we actually pre-calculate and tabulate the sampling probabilities for a unit-size cube. In the execution of the FRW method, with the pre-calculated tables the random walk can be performed quickly. For the problem with non-homogenous medium, the FRW method with cubes cannot be derived straightforwardly. However, with a pre-characterization procedure for the given piecewise homogenous dielectric configuration, the FRW method with cubes can still work very efficiently with some overhead of memory usage [23]. The strategy used is similar to those proposed in [1] for the diffusion problem, and in [3] with the spherical domain. To simplify the presentation, we only consider the homogenous dielectric configuration in this work. But, the presented techniques are also applicable to the situations with multiple dielectric regions.

For calculating the electric field intensity, we need some modification on (1). As an example, we consider to calculate the field component $E_x$ along the $x$-axis at point $r$ (see Fig. 2). We have

$$E_x(r) = -\nabla\phi(r) \bullet \hat{n}_x(r) = \oint_{S^{(1)}} -\nabla_r G_\phi^{(1)}(r, r^{(1)}) \bullet \hat{n}_x(r)\phi(r^{(1)})dr^{(1)}, \tag{3}$$

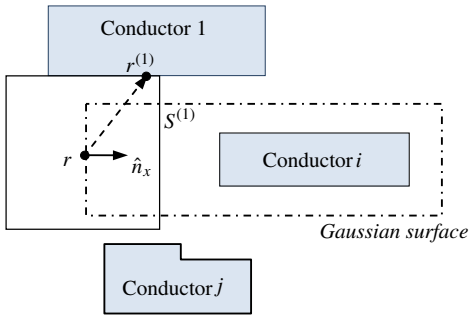


**Fig. 2.** The illustration of calculating electric field intensity and capacitance with the FRW method.

where $\nabla_r$ means the gradient operator is with respect to $r$. $E_x$ can still be evaluated with the FRW procedure, because

$$E_x(r) = \oint_{S^{(1)}} \omega_x(r, r^{(1)}) G_\phi^{(1)}(r, r^{(1)}) \phi(r^{(1)}) dr^{(1)}, \tag{4}$$

where

$$\omega_x(r, r^{(1)}) = -\frac{\nabla_r G_\phi^{(1)}(r, r^{(1)}) \bullet \hat{n}_x(r)}{G_\phi(r, r^{(1)})}. \tag{5}$$

The difference between (4) and (1) is the extra item $\omega_x(r, r^{(1)})$, which we call weight value. The weight value only depends on the first hop of walk, and its calculation can also be accelerated with a pre-characterization process as that for the transition probability. Multiplying the weight value and the potential of finally touched conductor produces an estimate of the electric field intensity. The above derivation also applies to the calculation of components $E_y$ and $E_z$.

The calculation of electric field can be extended to the computation of electric capacitance. The capacitances among conductors (electrodes) are important parameters for a system of conductors, and have a significant role in the modeling and analysis of VLSI circuit and MEMS. For example, the interconnect delay has dominated the circuit delay, and should be calculated with the equivalent circuit of interconnect with capacitance and resistance components. To compute the capacitance, a Gaussian surface $G_i$ is constructed to enclose conductor $i$ (called master conductor; see Fig. 2). According to the Gauss theorem,

$$Q_i = \oint_{G_i} D(r) \bullet \hat{n}(r) dr = \oint_{G_i} F(r)(-\nabla\phi(r)) \bullet \hat{n}(r) dr, \tag{6}$$

where $Q_i$ is the charge on conductor $i$, $F(r)$ is the dielectric permittivity at point $r$, and $\hat{n}(r)$ is the outward normal direction of $G_i$ at $r$. Similar to (4), we derive:

$$Q_i = \oint_{G_i} F(r) g dr \oint_{S^{(1)}} \omega(r, r^{(1)}) G_\phi^{(1)}(r, r^{(1)}) \phi(r^{(1)}) dr^{(1)}, \tag{7}$$

where the weight value

$$\omega(r, r^{(1)}) = -\frac{\nabla_r G_\phi^{(1)}(r, r^{(1)}) \bullet \hat{n}(r)}{g G_\phi^{(1)}(r, r^{(1)})}, \tag{8}$$

and the constant $g$ satisfies $\oint_{G_i} F(r) g dr = 1$. Now the first integral in (7) can be interpreted as a stochastic sampling procedure on $G_i$, and the second integral can be calculated with the above FRW procedure based on (2). The only difference is the extra weight value (8), which only depends on the relative position of $r^{(1)}$ and the direction $\hat{n}(r)$. By a pre-characterization procedure with the unit-size cube domain for the $\hat{n}_x$, $\hat{n}_y$ and $\hat{n}_z$ directions, the calculation of weight value (8) can be largely accelerated. The above deduction relates the conductor charge to the conductor voltages (potential) through the FRW procedure, which reveals that the statistical mean of the weight values for the walks terminating at conductor $j$ approximates the coupling capacitance $C_{ij}$ ($j \neq i$) between conductors $i$ and $j$. If $j = i$, the obtained $C_{ii}$ is called the self-capacitance of master conductor $i$.

The FRW algorithm for capacitance calculation can be described as Algorithm 1.

**Algorithm1**: The FRW algorithm for capacitance calculation

1: Load the pre-computed transition probabilities and weight values for the unit-size cubic domain;
2: Construct the Gaussian surface enclosing master conductor $i$;
3: $C_{ij} := 0, \forall j$; $npath := 0$;
4: **Repeat**
5: $npath := npath + 1$;
6: Pick a point $r$ on the Gaussian surface, and generate a cubic transition domain $T$ centered at $r$; pick a point $r^{(1)}$ on the surface of $T$ with the transition probabilities and then calculate the weight value $\omega$ (8) with the help of the pre-computed weight values;
7: **While** the current point is not on a conductor **do**
8: Construct the largest cubic domain with known transition probabilities;
9: Pick a point on the domain surface, according to the transition probabilities;
10: **End**
11: $C_{ij} := C_{ij} + \omega$; //the current point is on conductor $j$
12: **Until** the convergence criterion is met
13: $C_{ij} := C_{ij}/npath, \forall j$.

The FRW algorithm has been successfully proposed for calculating the capacitances of VLSI interconnects, where a large number of rectilinear conductors are involved [15,23]. Comprehensive comparison of the FRW method and the deterministic methods of FEM and BEM has been carried out in [16,24]. The results show that the FRW method possesses several favorable features: (1) lower memory usage which is also independent to the accuracy demand, (2) tunable accuracy which is easy to control, (3) running time independent to problem complexity, (4) easier to be parallelized on multi-core CPU platform. Although tabulating the surface Green's function and weight value for the cubic transition domain incurs some error, experimental results have shown this bias error is about 0.01% or less [16]. A recent progress of the FRW algorithm for the capacitance calculation of VLSI structures is the hierarchical FRW (HFRW) algorithm in [32]. The HFRW algorithm considers the problem of fabric-aware capacitance extraction, which includes the topological variations of the composition of several "motif" structures. So, it is not suitable for the general problem of capacitance calculation.

## 3. The variance reduction techniques

For a non-bias MC or FRW procedure, the statistical mean of $n$ samples obeys the normal probability distribution. Due to the central limit theorem [21], its standard deviation (Std) can be estimated with:

$$err_n \approx \frac{\sqrt{Var_n}}{\sqrt{n}}, \tag{9}$$

where $Var_n$ denotes the variance of the sample values. $err_n$ in (9) is also called the $1-\sigma$ error. With (9), it can be observed that the error is inversely proportional to the square root of the number of samples because $Var_n$ approaches to a constant. Note that, the $1-\sigma$ error $err_n$ means the error bound at the confidence level of 68%; it has the confidence level of 99.7% that the result is within the error of $3 \cdot err_n$.

In the electrostatic computation with FRW, the $1-\sigma$ error of result can be predicted with (9). So, the accuracy criterion can be set as the termination condition of the FRW method. In practice, we check the error after every certain number of walks. Once the $1-\sigma$ error is below the specified accuracy goal, the algorithm will terminate. In this section, we present two techniques for the variance reduction of FRW method, which largely reduces the number of walks without scarifying the accuracy.

### 3.1. The importance sampling with weight values averaged

To calculate a multi-dimensional integral

$$I = \int_{\Omega} f(\boldsymbol{x}) d\boldsymbol{x}, \tag{10}$$

with the MC method, we pick $n$ random samples $\boldsymbol{x}_i$ uniformly distributed in $\Omega$. Then,

$$I_n = A \cdot \frac{\sum_{l=1}^{n} f(\boldsymbol{x}_l)}{n}, \tag{11}$$

approximates $I$, where $A = \int_{\Omega} d\boldsymbol{x}$. Note that $I_n$ is the mean value of $A \cdot f(\boldsymbol{x}_l)$, and is also a random quantity due to the randomness of $\boldsymbol{x}_l$. The variance of function $A \cdot f(\boldsymbol{x})$ can be approximated by the variance of the discrete estimates $A \cdot f(\boldsymbol{x}_l)$,

$$Var_n = \frac{A^2 \sum_{l=1}^{n} (f(\boldsymbol{x}_l))^2 - n \cdot I_n^2}{n}. \tag{12}$$

The Std of $I_n$ can be estimated with (9) and (12). It can be found out that reducing the variance of the integrand $f(\boldsymbol{x})$ brings smaller $err_n$ and therefore more accurate $I_n$, for a given number of sample points. The important sampling (IS) is a technique to reduce the variance, which uses a positive valued function $q(\boldsymbol{x})$ to convert the integral formula. If $q(\boldsymbol{x})$ is nearly proportional to the function $f(\boldsymbol{x})$ and $\int_{\Omega} q(\boldsymbol{x}) d\boldsymbol{x} = 1$, (10) can be transformed to

$$I = \int_{\Omega} \frac{f(\boldsymbol{x})}{q(\boldsymbol{x})} q(\boldsymbol{x}) d\boldsymbol{x}. \tag{13}$$

Now, we can use $q(\boldsymbol{x})$ as a probability density function for the sampling of $\Omega$, and evaluate $f(\boldsymbol{x}_l)/q(\boldsymbol{x}_l)$ as the estimate in the MC procedure. Since $q(\boldsymbol{x})$ is nearly proportional to $f(\boldsymbol{x})$, the variance of the new integrand $f(\boldsymbol{x})/q(\boldsymbol{x})$ is much less than that of $f(\boldsymbol{x})$. Therefore, the MC procedure corresponding to (13) will converge faster.

An IS technique was proposed in [30] for the FRW based capacitance extraction, where the function $q(\boldsymbol{x})$ was defined to be proportional to $1/R(r)$ and $R(r)$ was the distance of point $r$ to the conductor inside the Gaussian surface. This is based on the assumption that the electric field diminishes with the inverse of distance. Below we propose another IS technique which is based on the mathematical properties of the integrand, and therefore is more rigorous and general.

In the FRW method, the weight values in (5) and (8) are the estimates of electric field and capacitance, respectively. Inspired by the idea of IS, we can reduce the variance of weight values by converting the FRW formula, which would benefit the convergence rate of the MC procedure. With the tabulated weight value, we actually know the distribution of weight value

for different samples on cube surface. Suppose the direction vector in (5) or (8) is along the $z$-axis. As an example, we draw the distributions of weight value for the $r^{(1)}$ on top and side faces of the cube in Fig. 3.

From Fig. 3 we see that $\omega$ is non-positive for $r^{(1)}$ on the top face, while it may approach to zero for $r^{(1)}$ on the side face (outlined with bold lines). The figure demonstrates that the weight value has certain degree of variance. We now seek a function $q(\boldsymbol{x})$ to modify the integral formula (5) and (8), so as to reduce the variance of resulting weight value. Without loss of generality, we only derive the formula for the problem of capacitance calculation expressed by (7) and (8).

We firstly convert (8) to:

$$\omega(r, r^{(1)}) = -\frac{1}{g \cdot L} \cdot \frac{\frac{\partial G_\phi(r,r^{(1)})}{\partial x} n_x + \frac{\partial G_\phi(r,r^{(1)})}{\partial y} n_y + \frac{\partial G_\phi(r,r^{(1)})}{\partial z} n_z}{G_\phi(r, r^{(1)})}, \tag{14}$$

where $L$ is the edge length of the first cube, and we assume $\hat{n}(r) = [n_x, n_y, n_z]^T$. With the superscript dropped, here $G_\phi(r, r^{(1)})$ denotes the surface Green's function for the unit-size cube, and $r$ and $r^{(1)}$ is the corresponding points in the unit-size cube. Then, we define

$$K = \oint_S |\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)| dr^{(1)}, \tag{15}$$

where $S$ is the boundary of the unit-size cube. Since $r$ is the center of cube, $K$ is a positive constant for a known $\hat{n}(r)$ along $x$-, $y$-, or $z$-axis direction. We transform (7) to:

$$\begin{aligned} Q_i &= \oint_{G_i} F(r) g dr \oint_S -\frac{\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)}{L \cdot g} \phi(r^{(1)}) dr^{(1)} \\ &= \oint_{G_i} F(r) g dr \oint_S \frac{-K \cdot \nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)}{L \cdot g |\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)|} \cdot \frac{|\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)|}{K} \phi(r^{(1)}) dr^{(1)}. \end{aligned} \tag{16}$$

We can define the function $q(\boldsymbol{x})$ to be:

$$q(r, r^{(1)}) = \frac{|\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)|}{K}. \tag{17}$$

which can serve as the PDF for sampling $S$, according to (15). Then,

$$Q_i = \oint_{G_i} F(r) g dr \oint_S \frac{-K \cdot \nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)}{L \cdot g |\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)|} \cdot q(r, r^{(1)}) \phi(r^{(1)}) dr^{(1)}. \tag{18}$$

This suggests a new FRW scheme, where the weight value is

$$\tilde{\omega}(r, r^{(1)}) = \begin{cases} -K/(L \cdot g), & \text{if } \nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r) > 0 \\ K/(L \cdot g), & \text{if } \nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r) < 0 \end{cases}. \tag{19}$$

Note it is impossible that $r^{(1)}$ has a value such that $\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r) = 0$, according to the sampling probability function (17). In Fig. 4, we draw the distribution of the new weight value (19) under the same assumptions in Fig. 3. For cubes with same size and same $\hat{n}(r)$, the new weight value has only two distinct values. Thus, the new FRW scheme may exhibit much less variance than the original one.

Now, the point sampling on the first cube obeys the new probabilities obtained from $q(r, r^{(1)})$. Considering the difference of (17) and (8), we see that we can characterize the new transition probability by multiplying the original discrete transition

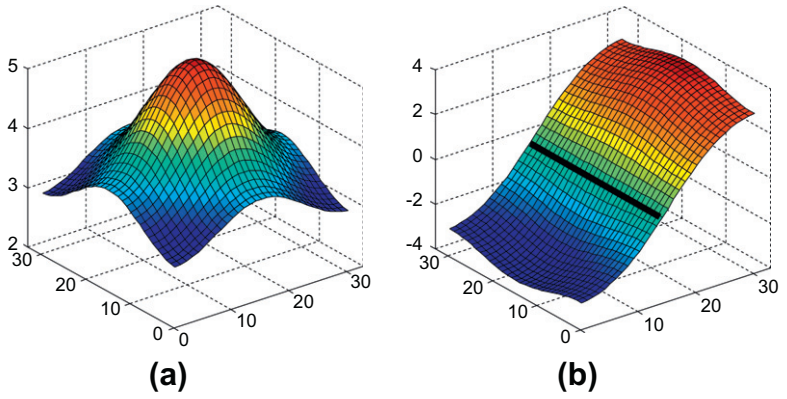


**Fig. 3.** The distributions of weight value on (a) the top face, and (b) a side face of the cube.

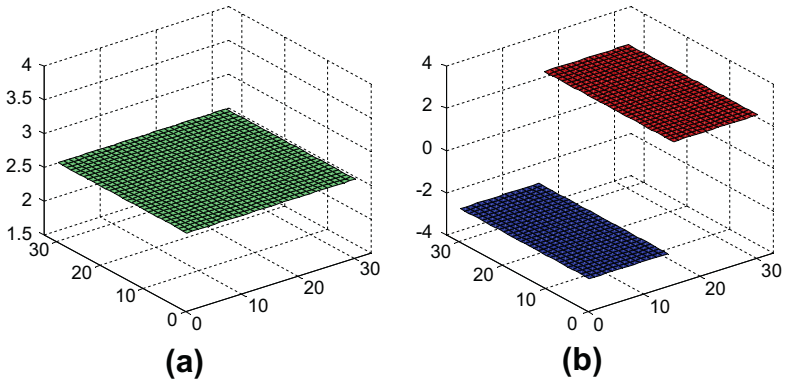


**Fig. 4.** The distributions of weight value on (a) the top face, and (b) a side face of the cube, after using the importance sampling technique.

probability and weight value for a given $\hat{n}(r)$ direction, element by element. It needs the same amount of storage as that for the original weight value in the tabulation step. The $K$ in (15) has only three distinct values, because we only consider three coordinate directions in the applications with rectilinear geometries, and it is also pre-computed.

### 3.2. The stratified sampling technique

Another variance reduction technique is the stratified sampling (SS), whose idea is to divide the original integral to several integrals on its subdomains (also called "strata"), and then calculate them separately. For example, dividing the $\Omega$ in (10) into two equal-sized subdomains $\Omega_a$ and $\Omega_b$, we have

$$I = I_a + I_b = \int_{\Omega a} f(\boldsymbol{x})d\boldsymbol{x} + \int_{\Omega b} f(\boldsymbol{x})d\boldsymbol{x}. \tag{20}$$

If the MC method is used to calculate $I_a$ and $I_b$ separately, the formula (11) can be substituted with:

$$I'_n = I^{(a)}_{na} + I^{(b)}_{nb} = \frac{A}{2}\cdot\frac{\sum_{l=1}^{n_a} f(\boldsymbol{x}_{a,l})}{n_a} + \frac{A}{2}\cdot\frac{\sum_{l=1}^{n_b} f(\boldsymbol{x}_{b,l})}{n_b}, \tag{21}$$

where $n_a$ and $n_b$ are the sampling numbers in $\Omega_a$ and $\Omega_b$ respectively ($n_a + n_b = n$). Due to the stochastic independence of $I^{(a)}_{na}$ and $I^{(b)}_{nb}$, the variance of $I'_n$ is the sum of the variances of $I^{(a)}_{na}$ and $I^{(b)}_{nb}$. It can be proved that $Var(I'_n) < Var(I_n)$, if $f(\boldsymbol{x})$ has different mean value in subdomains $\Omega_a$ and $\Omega_b$ [21]. Furthermore, $Var(I'_n)$ achieves its minimum value if $(n_a/n_b)^2$ equals to the ratio of variances of function $f(\boldsymbol{x})$ in $\Omega_a$ and $\Omega_b$. Because the $Var(I_n)$ is the square of the Std in (9), calculating (21) with the idea of SS produces the estimation to the original integral with less $1-\sigma$ error. Thus, we need fewer MC samples to achieve the same accuracy.

The SS technique was also considered in [30], where the first transition cube's surface is decomposed into 16 stratas according to the distribution of homogenous-medium weight value. However, based on the IS technique proposed in last subsection, the weight value has only two nonzero values for a given first transition cube. Decomposing the cube's surface into 16 stratas as in [30] is therefore unnecessary. We just divide it into two stratas, one with the positive weight value and the other with negative weight value in (19). Although the weight value is averaged with the IS technique, it still fluctuates because the first cube varies with different positions of $r$. This kind of variance can be further reduced with the SS technique

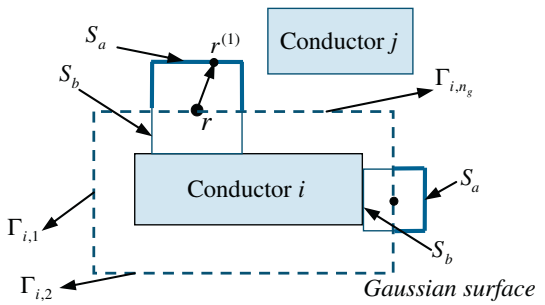


**Fig. 5.** A 2-D view illustrating the $n_g$ faces of a Gaussian surface, and the two half surfaces of the first transition cube.

which decomposes the Gaussian surface. Suppose the Gaussian surface has $n_g$ faces (see Fig. 5). The original Eq. (7) is converted to:

$$\begin{aligned} Q_i &= \sum_{k=1}^{n_g} \int_{\Gamma_{i,k}} F(r)gdr \oint_S -\frac{\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)}{L \cdot g} \phi(r^{(1)}) dr^{(1)} \\ &= \sum_{k=1}^{n_g} \int_{\Gamma_{i,k}} F(r)gdr \int_{S_a} -\frac{\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)}{L \cdot g} \phi(r^{(1)}) dr^{(1)} + \sum_{k=1}^{n_g} \int_{\Gamma_{i,k}} F(r)gdr \int_{S_b} -\frac{\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)}{L \cdot g} \phi(r^{(1)}) dr^{(1)}, \end{aligned} \tag{22}$$

where $\Gamma_{i,k}$ stands for the $k$th face of the Gaussian surface $G_i$. $S_a$ stands for the surface of the scaled half cube outside $G_i$, and $S_b$ stands for the surface of the scaled half cube inside $G_i$. Note that $\nabla_r G_\phi(r, r^{(1)}) \bullet \vec{n}(r)$ always has positive value for $r^{(1)}$ on $S_a$, and negative value for $r^{(1)}$ on $S_b$.[1] Now, $Q_j$ becomes the summation of $2n_g$ integrals. Each integral can be calculated separately. For example,

$$I_k = \int_{\Gamma_{i,k}} F(r)gdr \int_{S_a} -\frac{\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)}{L \cdot g} \phi(r^{(1)}) dr^{(1)}, \quad (1 \leqslant k \leqslant n_g). \tag{23}$$

Because $\nabla_r G_\phi(r, r^{(1)}) \bullet \vec{n}(r)$ is positive for $r^{(1)}$ on $S_a$, we define

$$K_a = \int_{S_a} \nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r) dr^{(1)}, \tag{24}$$

Then,

$$q_a(r, r^{(1)}) = \frac{\nabla_r G_\phi(r, r^{(1)}) \bullet \vec{n}(r)}{K_a}, \tag{25}$$

becomes a PDF for the second integral in (23). We derive

$$I_k = A_k \int_{\Gamma_{i,k}} \frac{F(r)g}{A_k} dr \int_{S_a} -\frac{K_a}{L \cdot g} q_a(r, r^{(1)}) \phi(r^{(1)}) dr^{(1)}, \quad (1 \leqslant k \leqslant n_g), \tag{26}$$

where $A_k = \int_{\Gamma_{i,k}} F(r)gdr$. This can be interpreted by a FRW procedure, where the sampling on $S_a$ obeys the PDF of $q_a(r, r^{(1)})$ and the corresponding weight value is

$$\tilde{\omega}_k(r, r^{(1)}) = \frac{-K_a}{L \cdot g}, \quad (1 \leqslant k \leqslant n_g). \tag{27}$$

Similar derivation can be applied to

$$I_k = \int_{\Gamma_{i,k-n_g}} F(r)gdr \int_{S_b} -\frac{\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)}{L \cdot g} \phi(r^{(1)}) dr^{(1)}, \quad (k > n_g), \tag{28}$$

provided that $K_b$ and $q_b(r, r^{(1)})$ are defined like $K_a$ and $q_a(r, r^{(1)})$, respectively. And for $k > n_g$, we let $A_k$ equal to $A_{k\text{-}ng}$. Suppose $n_k$ walks are used to calculate $I_k$, ($k = 1, \ldots, 2n_g$), and the weight values returned are $\tilde{\omega}_{k,1}, \ldots, \tilde{\omega}_{k,n_k}$. The capacitance derived from (22) with the SS technique is computed as

$$\overline{C}_{ij} = \sum_{k=1}^{2n_g} \theta_k = \sum_{k=1}^{2n_g} \left( \frac{A_k}{n_k} \sum_{l=1}^{n_k} \tilde{\omega}_{k,l} \right), \tag{29}$$

where $\theta_k$ is the mean value for the $k$th strata. Also note that if the $l$th walk for strata $k$ does not terminate at conductor $j$, the returned $\tilde{\omega}_{k,l}$ is zero. And, the corresponding $1 - \sigma$ error is computed piecewise as

$$\mathrm{std}(\overline{C}_{ij}) = \sqrt{\sum_{k=1}^{2n_g} Var(\theta_k)} = \sqrt{\sum_{k=1}^{2n_g} \left( \frac{A_k^2}{n_k^2} \sum_{l=1}^{n_k} \tilde{\omega}_{k,l}^2 - \frac{\theta_k^2}{n_k} \right)}. \tag{30}$$

The last equality in (30) is due to (9) and (12).

It should be pointed out that the proposed IS and SS techniques are more solid in theory than those in [30]. They are applicable without loss of generality to non-homogeneous media and do not make any assumption on the expected physical behavior of the electric field.

Comparing (24) and (25) with (15) and (17), we conclude that the new sampling scheme is compatible with the scheme in last subsection. We do not need to change the pre-characterization scheme, which tabulates the surface Green's function

[1] Supposing $\vec{n}(r)$ is along the positive $z$-axis direction, $\nabla_r G_\phi(r, r^{(1)}) \bullet \vec{n}(r) = \partial G_\phi(r, r^{(1)})/\partial z = \lim_{\Delta z \to 0} [G_\phi(r + \Delta z, r^{(1)}) - G_\phi(r, r^{(1)})]/\Delta z$. Because $G_\phi(r, r^{(1)})$ means the correlation (or contribution) coefficient of the potential at point $r^{(1)}$ to the potential at point $r$, a movement of $r$ towards $r^{(1)}$ makes this correlation stronger. So, if $r^{(1)}$ is on $S_a$ (see Fig. 5), $G_\phi(r + \Delta z, r^{(1)}) - G_\phi(r, r^{(1)}) > 0$, and thus $\nabla_r G_\phi(r, r^{(1)}) \bullet \vec{n}(r)$ has a positive value. For $r^{(1)}$ on $S_b$, we can similarly show that $\nabla_r G_\phi(r, r^{(1)}) \bullet \vec{n}(r) < 0$.

$G_\phi(r, r^{(1)})$, the new PDF $q(r, r^{(1)})$ in (17) and the $K$ in (15). Actually, $K$ is double of $K_a$ in (24),[2] and the complete formula of (27) is:

$$\tilde{\omega}_k(r, r^{(1)}) = \begin{cases} \frac{-K}{2L \cdot g}, & 1 \leqslant k \leqslant n_g \\ \frac{K}{2L \cdot g}, & n_g < k \leqslant 2n_g \end{cases}. \tag{31}$$

In our implementation, the values of $n_k$, ($k = 1, \ldots, 2n_g$) are not pre-defined. Instead, the FRW procedure corresponding to the IS technique in last subsection is performed. The actual walks are registered to the $2n_g$ stratas, followed by calculating the capacitance value and error estimation with formulas similar to (29) and (30). The resulting FRW algorithm for capacitance calculation is described as Algorithm 2.

**Algorithm 2**: The FRW algorithm with variance reduction for capacitance calculation

1: Load the pre-computed surface Green's function, the new PDF and the values of $K$.
2: Construct the Gaussian surface enclosing master conductor $i$;
3: $C_{ij} := 0, \forall j$; $n_k := 0, \forall k$;
4: **Repeat**
5: Pick a point $r$ on Gaussian surface, and then generate a cubic transition domain T centered at it; pick a point $r^{(1)}$ on the surface of T according to the new PDF; obtain the value of $k$ according to the position of $r^{(1)}$ on the cube, and calculate $\tilde{\omega}_k$ with (31).
6: **While** the current point is not on a conductor **do**
7: Construct the largest cubic domain with known transition probabilities;
8: Pick a point on the domain surface, according to the surface Green's function of the cubic domain;
9: **End**
10: Register $\tilde{\omega}_k$ to strata $k$ for $C_{ij}$, where $j$ is the number of conductor hit, and register 0 to strata $k$ for $C_{il}$ ($\forall l, l \neq j$);
11: $n_k := n_k + 1$;
12: Calculate the $1 - \sigma$ errors with (30);
13: **Until** the stopping criterion is met
14: Calculate $C_{ij}, \forall j$ with formula (29).

## 4. Parallelizing the FRW algorithm on GPUs

In this section, we present the techniques to parallelize the FRW method on GPU. The derivation is based on the capacitance calculation, while it can be easily applied to the computation of electric field intensity.

### 4.1. GPU architecture and CUDA

The GPU is a highly parallel, multi-threaded, many-core processor with tremendous computational horsepower and very high memory bandwidth. In the recent Nvidia Fermi GPU, each streaming multi-processor (SM) includes 32 streaming processors (SPs), resulting in totally 512 SPs into a GPU chip and up to 1.5 TFlops peak computing performance [25]. The peak performance of GPU is almost 10 times that of the CPU in the same year. Because in the GPU more transistors are devoted to data processing rather than data caching and flow control, 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 device memory. The shared memory resides in each SM and is much faster than the device memory. However, the size of shared memory is limited (e.g. 48 kB). The device memory has larger than 1 GB size, but has long access latencies (several hundreds of clock cycles) and limited access bandwidth. Hence, the excessive access to device memory should be avoided.

The common unified device architecture (CUDA) released by NVIDIA Inc. 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, each of which is assigned to a certain SM, and the threads inside a same block exchange data through the shared memory of the SM. Fig. 6 shows the hierarchical thread organization and the corresponding hardware architecture, where the thread blocks are organized into a grid for the use of a CUDA kernel function. 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.

[2] Because $\oint_S G_\phi(r, r^{(1)}) dr^{(1)} = 1$, $\oint_S \nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r) dr^{(1)} = 0$. According to the definition of $K_a$ and $K_b$, $\oint_S \nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r) dr^{(1)} = K_a + K_b = 0$. We have proved that $K_a > 0$, which means $K_b$ is its opposite number. So, $K = \oint_S |\nabla_r G_\phi(r, r^{(1)}) \bullet \hat{n}(r)| dr^{(1)} = K_a - K_b = 2K_a$.

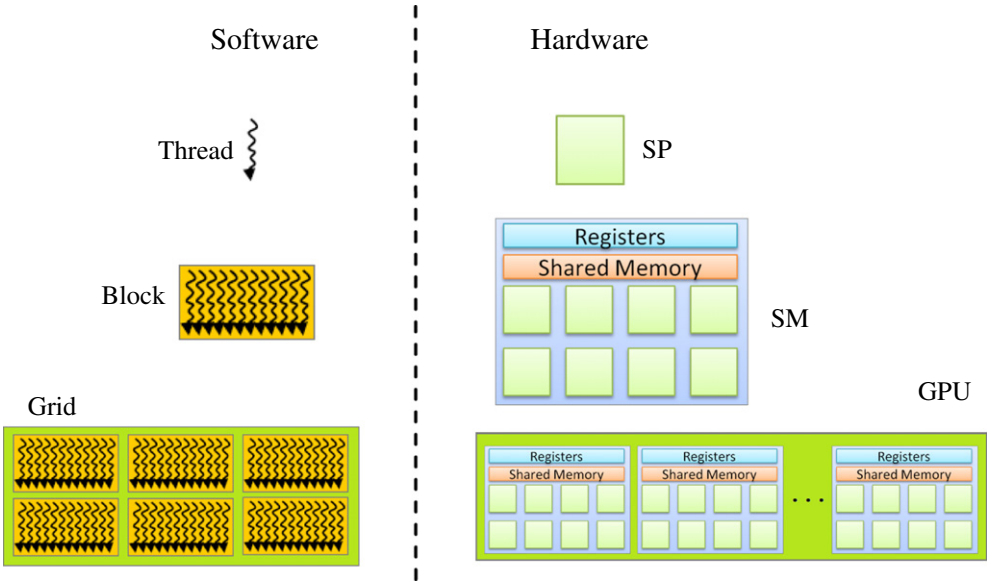


**Fig. 6.** The hierarchical thread organization in CUDA.

*4.2. An iterative FRW flow with divided tasks*

Simply parallelizing the CPU-based FRW on-the-fly will not yield high efficiency due to the divergence of number of hops needed by different walks. A thread block cannot terminate until the thread with the longest walk finishes. This leads to the great waste of computing resource. A better way to parallelize FRW on GPU would be the walk-regeneration strategy [26]. It generates a new walk once a thread finishes a walk. This strategy is able to improve the workload balance to some extent. However, the operations of threads in the same block become divergent. The operation divergence causes threads executing in a serial way because of the SIMD scheme of CUDA. On the other side, in the original FRW algorithm, every thread needs to register their contribution to a global variable (e.g. the capacitance) after it finishes a walk. To make sure only one thread modify the global variable at a time, a lock to it is needed. The thread competition for the lock will harm the performance.

With the foregoing concerns, we propose to divide the FRW algorithm into three kernels for different specific tasks:

- ***Walk-starting kernel***. It includes choosing a point $r$ from the Gaussian surface (if needed), generating the transition cube $S^{(1)}$ centered at $r$, randomly selecting a point $r^{(1)}$ on $S^{(1)}$, and calculating the weight value $\tilde{\omega}_k$.
- ***Hop kernel***. It loads $r^{(1)}$ and the weight values generated by the walk-starting kernel, repeatedly constructs the transition cube and chooses a point on it. Once the point hits a conductor, the number of the hit conductor is stored in the device memory.
- ***Reduction kernel***. It calculates the mean values and the variances for error estimation with the walk results generated by the hop kernel. The techniques in last Section can be employed.

With this task dividing, the instruction divergences and resource competition are largely reduced, as shown in Fig. 7. In the first kernel there is no divergence, because every thread follows the same instructions. For the third kernel, there are efficient parallel reduction algorithms based on the GPU architecture [27,28]. For the hop kernel, the divergent part is just several instructions of memory accessing. Hence, the efficiency of parallelization is largely improved (see Fig. 7).

The FRW algorithm usually sets the accuracy as the termination criterion. However, the task-dividing strategy is not compatible to the accuracy control in the CPU-based FRW algorithm (Line 13 in Algorithm 2), because the hop kernel does not update the variance timely. To enable the FRW on GPU to terminate based on the accuracy criterion, we propose an iterative FRW flow (see Fig. 8). In each iteration step, a number of walks are assigned to the sequential three kernels, and finally the reduction kernel produces the variance. After the first iteration, a mechanism of automatic walk assignment is applied, based on (9). 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

$$n_{rem} = n_{cur} \times \left( \frac{err_{cur}^2}{err_{goal}^2} - 1 \right). \tag{32}$$

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

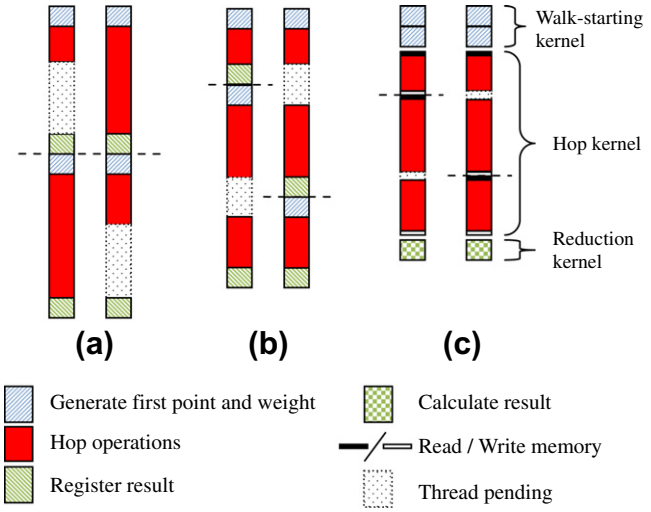


**Fig. 7.** The execution of FRW algorithm on two threads with three strategies: (a) simple one, (b) walk-regeneration, (c) the proposed task-dividing. The horizontal dot lines indicate the border of two walks. Note that the task-dividing strategy consumes the least time.

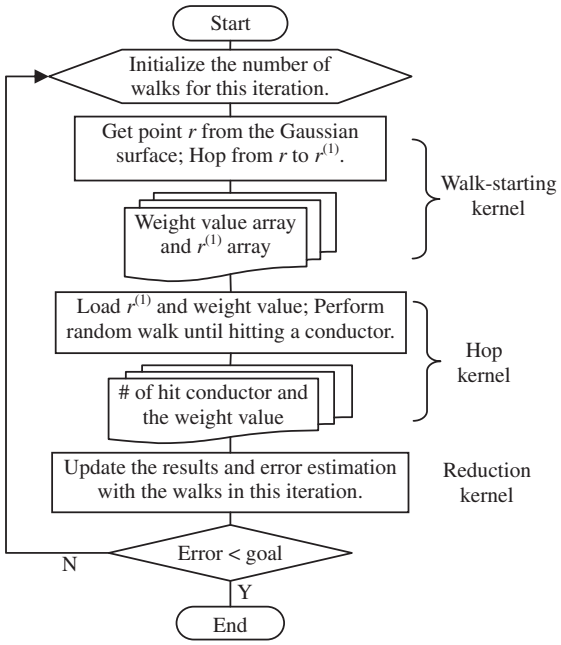


**Fig. 8.** The iterative FRW flow on GPUs with three kernels.

The three kernels exchange data via the storage on the device memory. To avoid accessing the same memory location for different threads, we assign every thread with exclusive memory space to store the starting points and weight values generated by the first kernel. This memory assignment is performed at the beginning of the 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. We shall assign the tasks evenly, i.e. let

$$W_t = \frac{W_b}{BlockDim}. \tag{33}$$

During the hop kernel, there are some threads executing faster. If we generate exactly the same number of start points for each thread with the first kernel, i.e., $N_{sp} = W_t$, where $N_{sp}$ is the number of start points for each thread, when the faster threads finishes $W_t$ walk, they have to wait for the slower threads. We have observed that the hop kernel consumes most

of the computing time. Thus, to improve the efficiency, we propose to generate redundant start points for each thread to enable fast threads to do more walks. This means

$$N_{sp} = r_{redun} \times \frac{W_b}{BlockDim}, \tag{34}$$

where $r_{redun} > 1$ is the redundant factor. With the new scheme, in the hop kernel, a thread terminates when either it performs $N_{sp}$ walks or its block finishes $W_b$ walks. The larger is $r_{redun}$, the fewer is the waiting time for a block, with the price of generating more unused start points. Some results on three test cases are shown in Fig. 9, which show at most 40% speedup can be obtained if setting $r_{redun}$ properly. In our experiments, we set $r_{redun} = 2$, which is nearly optimal for most cases.

With exclusive memory space and redundant start point generation, the flowchart of hop kernel is shown as Fig. 10. Vertical dot lines indicate the borders of the memory space for a thread. It should be pointed out that the actual number of walks performed by a block is always larger than the assigned value of $W_b$. This also guarantees that the flow converges with two or three iteration steps in most situations, and therefore minimizes the overhead of iteration.

### 4.3. Hop with the inverse cumulative probability array

There are usually millions of hops in the FRW algorithm. Therefore, reducing the computing time for each hop is crucial. A main operation in each hop is to pick a point on the cube surface according the transition probabilities. With the pre-calculated transition probabilities stored in the device memory, the inverse transform sampling is used to translate a uniform random number to the index of a surface panel. Suppose there are $N$ panels, and the transition probabilities for the panels are $p_i$, $i = 1, \ldots, N$. To facilitate the computation, the cumulative summation of $p_i$ is calculated in advance:

$$s_i = \sum_{j=1}^{i} p_j, \quad i = 1, \ldots, N. \tag{35}$$

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 device memory of GPU, due to the space limit of on-chip memory, the approach of binary search will be suffered from the long memory access latency.

To reduce the time of picking a point, 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,

$$u_{res} = \min_{1 \leqslant i \leqslant N} \{p_i\}. \tag{36}$$

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

$$t_i = \lfloor p_i/u_{res} + 0.5 \rfloor, \quad i = 1, \ldots, N, \tag{37}$$

$$\mathrm{ICPA}\left[\sum_{i=0}^{k-1} t_i + 1 : \sum_{i=0}^{k-1} t_i + t_k\right] = k, \quad (k = 1, 2, \ldots, N). \tag{38}$$

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 the panel of cube surface quickly. The time complexity is $O(1)$, instead of $O(N \log N)$ of the binary search. In the implementation on GPU, this reduces the time of accessing the device memory and the operation divergence between different hops. As for the overhead, the ICPA corresponding to the tabulated surface Green's function is within several tens megabytes, which is obviously affordable.

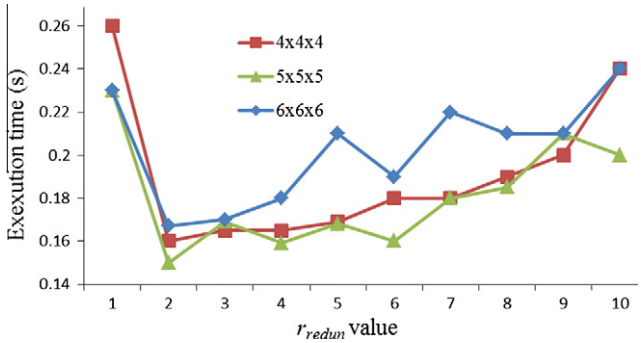


**Fig. 9.** The relationship between $r_{redun}$ and performance. The three test cases are the $4 \times 4 \times 4$, $5 \times 5 \times 5$, $6 \times 6 \times 6$ cross-over interconnect structures respectively.

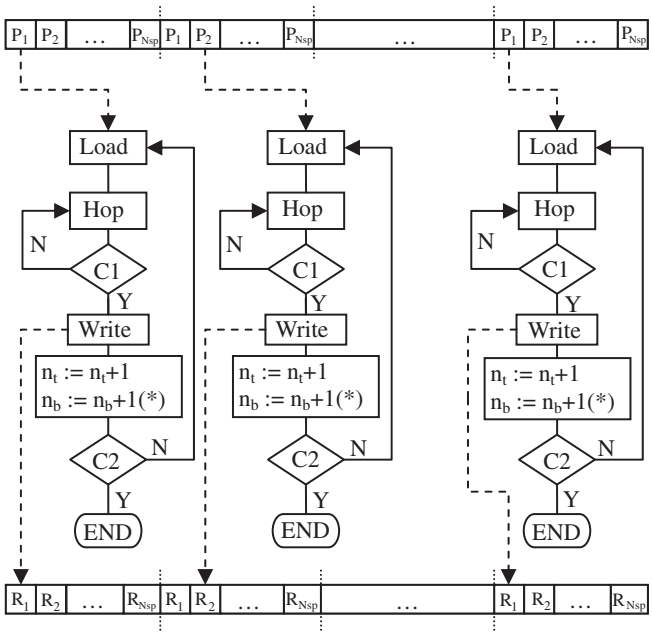


**Fig. 10.** The flowchart of the hop kernel. C1 means "Point on conductors?" and C2 means "$n_{t>=}$ $N_{sp}$ or $n_b > W_b$?" The operation with (∗) means it is an atomic operation on shared memory.

## 5. Numerical results

We have implemented the 3-D FRW algorithms in C++ and the GPU programming language CUDA, for the calculation of capacitance and electric field. The "Mersenne Twister" algorithm [29] is used to generate the sequence of random numbers, which has a very long period of $2^{19937} - 1$. Numerical results are presented in this section to validate the accuracy and efficiency of the proposed techniques. We firstly present the experiments on two small simple structures, and then those on larger cases with considerable number of conductors. All experiments are carried out on a PC with a 2.70 GHz Pentium dual-core CPU, 6 GB memory, and one Nvidia GTX580 GPU card with 1.5 GB device memory. In all experiments, the CPU version of FRW algorithm is executed in the serial computing mode. It has been demonstrated that the parallel FRW algorithm can achieve about 7× speedup on a machine with 8 CPU cores [23]. So, the computing time of parallel CPU version of FRW algorithm can be estimated for the comparison with the proposed GPU-based algorithm.

### 5.1. Smaller cases

The first test case is an isolated unit cube. There is no analytical solution for the capacitance of this cube structure. Several methods have been applied to numerically calculate this quantity (see [8,9,19,20] and the references therein). While setting the potential of faraway points to be 0 V, the FRW algorithm can be used to solve this structure. This approximation on boundary makes the capacitance result overestimated. In our experiments, we set the faraway boundary 1000 times larger than the size of cube, which makes the capacitance result differ little from that for the structure with infinite boundary. The termination criterion of FRW algorithm is set to be $1 - \sigma$ error of 0.1%, and the results of the FRW algorithms are listed in Table 1, with $3\sigma$ error (corresponding to 99.7% confidence interval). The results obtained with other methods [8,9,19,20] are also given. Although the result from the FRW algorithm is a little larger than the results with higher accuracy [9,19,20], they are in good agreement. Note that the result in [8] has larger error, and the results from [9] and [20] are not consistent with each other.

To validate the efficiency of proposed variance reduction techniques and the GPU-based FRW algorithm, the computational time and result of different versions of FRW method are compared. The number of walks and times are listed in Table 2. From it we can see the variance reduction brings 2.5× speedup, which is produced by the reduction of the number of walks for same accuracy criterion. The speedup brought by the GPU-based parallel computing is over 60×, though more walks than the CPU counterpart are performed to take advantage of the massive parallelism of GPU. Statistically, there is no loss of accuracy with the proposed techniques. These FRW algorithms all consume 1.38 MB main memory in computer, which is mainly used to store the pre-calculated data for the transition probability and weight value. The variance reduction techniques hardly increase the memory usage of the FRW algorithm. For the GPU-based FRW algorithm, additional 26 MB is consumed in the device memory of GPUs for storing the data structures in Section 4.

**Table 1**
Capacitance of an isolated unit cube (in units of $4\pi\varepsilon$).

| Method | Result |
|---|---|
| Mascagni–Simonov, Random walk on the boundary [19] | $0.6606780 \pm 2.7 \times 10^{-7}$ |
| Hwang–Mascagni–Won, Random walk on the boundary [20] | $0.66067813 \pm 1.01 \times 10^{-7}$ |
| Bontzios–Dimopoulos–Hatzopoulos, Evolutionary method [8] | [0.6738, 0.6820] |
| Lazic–Stefancic–Abraham, The Robin Hood method [9] | $0.66067786 \pm 8 \times 10^{-8}$ |
| Proposed floating random walk algorithm | $0.6613 \pm 0.0020$ |

**Table 2**
Running time comparison of the original FRW algorithm, the FRW with variance reduction and the FRW accelerated with GPU computing (the accuracy criterion is set to be $1-\sigma$ error of 0.1%).

| Method | #Walk | Result ($4\pi\varepsilon$) | Time (s) | Speedup |
|---|---|---|---|---|
| Original FRW(CPU) | $3.56 \times 10^7$ | $0.6613 \pm 0.0020$ | 195.2 | – |
| FRW + Variance reduction (CPU) | $1.44 \times 10^7$ | $0.6619 \pm 0.0020$ | 78.1 | 2.5 |
| FRW + Variance reduction (GPU) | $2.47 \times 10^7$ | $0.6627 \pm 0.0020$ | 1.24 | **157** |

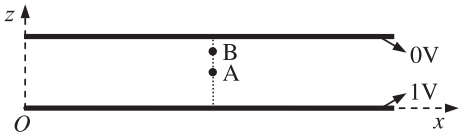


**Fig. 11.** The cross-section view of a two-parallel-plate structure. Its electrical capacitance and electric field intensity at points A and B are calculated.

The second case is a structure with two parallel plates. The size of the plates is $1000 \times 1000$, and their separation is 10 (see Fig. 11). The unit is μm. The coupling capacitance of the two plates and the electric field intensity at points A and B are calculated. The points have the coordinates of (500, 500, 5) and (500, 500, 8), respectively. In Table 3, we list the capacitance results and the field component $E_z$ along the $z$-axis. We see the result of electric field is very close to the accurate $E_z$ of $10^5$ V/m.[3] The result of capacitance is close to the accurate value of $8.854 \times 10^{-13}$ F with an error of 0.6%.[4]

The number of walks and computing time for the calculation of capacitance and electric field intensity are listed in Table 4. Because the accelerating techniques do not affect the computational result, the values of capacitance and electric field intensity are not presented in the table. We see that the variance reduction brings over 15× speedup to the capacitance calculation. While for the electric field intensity, the speedup is from 1.9 to 6.2. The reason for the less speedup is that the SS technique is not fully applicable for calculating the electric field. And, we see that the computing time for $E_z$ at point B is larger than that at point A, which is due to the smaller cube centered at B that only touches the top plate and thus has smaller probability for walk to terminate. With the GPU-based parallel computing, the total speedup ratio increases to 199× for the capacitance calculation and over 70× for the electric field calculation.

### 5.2. Larger cases from VLSI circuit and MEMS

The structures of the interconnects in VLSI circuit and the MEMS device involve a large number of conductors with rectilinear geometry. The proposed FRW algorithms are used to calculate the capacitances of these structures.

We firstly consider a set of cross-over structures. They are two-layer (denoted by "$m \times m$") or three-layer (denoted by "$m \times m \times m$") structures, where $m$ is the number of parallel wires lying in each metal layer. The cross-section of each wire is 70 μm × 140 μm, complying with the 45 nm process technology [23]. One of these cases is shown in Fig. 12. We set the convergence criterion of FRW to be 0.5% $1-\sigma$ error of self-capacitance. We calculate the capacitances related to the master conductor for each case, and list the computational results of the original FRW method, the FRW with variance reduction and the FRW accelerated with GPU computing in Table 5. Here "Cap." denotes the self-capacitance of the master conductor. From the table, we observe that with the proposed variance reduction and GPU-based FRW algorithm, more than 70 × speedup can be achieved. The memory usage of the FRW algorithms is still 1.38 MB, the same as for the cases in last subsection. Because the accuracy criterion in FRW method is lower than that used in the experiments in last subsection, we find out that the speedup ratio decreases as compared with Table 2 or Table 4. This verifies that the GPU computing is more effective for the task with more walks.

[3] For two infinite large parallel plates, $E_z = \Delta V/d$, where $\Delta V$ is the difference of the voltages on the two plates, and $d$ is their separating distance. And, the coupling capacitance between the plates is $C_c = \varepsilon l^2/d$, where $l$ is the edge size of plate.

[4] To calculate the parallel-plate capacitance we only count the charges on the bottom surface of the upper plate.

**Table 3**
The computational results of capacitance and electric field intensity for the two-parallel-plate case.

| | Capacitance ($10^{-13}$ F) | $E_z$ at point A ($10^5$ V/m) | $E_z$ at point B ($10^5$ V/m) |
|---|---|---|---|
| Analytical value | 8.854 | 1 | 1 |
| Result by FRW algorithm | $8.91 \pm 2.7 \times 10^{-2}$ | $1.000 \pm 0.003$ | $1.000 \pm 0.003$ |

**Table 4**
Running time comparison of the original FRW algorithm, the FRW with variance reduction and the FRW accelerated with GPU computing for the two-parallel-plate case.

| Method | #Walk | Time (s) | Speedup |
|---|---|---|---|
| *Capacitance* | | | |
| Original FRW(CPU) | $2.50 \times 10^7$ | 37.61 | – |
| FRW + Variance reduction (CPU) | $1.82 \times 10^6$ | 2.38 | 15.8 |
| FRW + Variance reduction (GPU) | $7.65 \times 10^6$ | 0.189 | **199** |
| *$E_z$ at point A* | | | |
| Original FRW(CPU) | $3.54 \times 10^6$ | 4.13 | – |
| FRW + Variance reduction (CPU) | $2.41 \times 10^6$ | 2.20 | 1.9 |
| FRW + Variance reduction (GPU) | $3.72 \times 10^6$ | 0.070 | **59** |
| *$E_z$ at point B* | | | |
| Original FRW(CPU) | $1.04 \times 10^7$ | 14.20 | – |
| FRW + Variance reduction (CPU) | $7.54 \times 10^6$ | 9.52 | 1.5 |
| FRW + Variance reduction (GPU) | $1.04 \times 10^7$ | 0.207 | **69** |

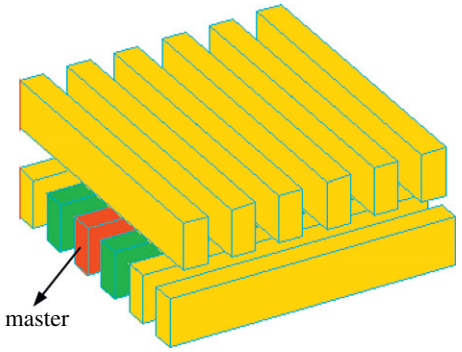


**Fig. 12.** The 3-D view of the "6 × 6" cross-over case.

Then, we consider two more complex structures. One is a part of interconnects cut from actual VLSI layout. As shown in Fig. 13, the structure includes metal wires distributed on five layers. In the figure we also draw a Gaussian surface enclosing the master conductor net. The capacitances between the master net and other nets are calculated. The other case is a transverse comb drive (shown in Fig. 14), which is a typical MEMS device widely applied in micro-accelerometers, hard disk actuators, etc. This structure has two electrodes with complex rectilinear geometry. The capacitance between these two electrodes is calculated. In Table 6, we present the computational results of our FRW methods, along with the results obtained from a fast BEM capacitance solver QBEM [7].

**Table 5**
The computational results of the original FRW algorithm, the FRW with variance reduction and the FRW accelerated with GPU computing for the cross-over cases (unit of capacitance is $10^{-17}$ F).

| Case | Original FRW (CPU) | | | FRW + VR (CPU) | | | FRW + VR (GPU) | | | Speedup |
|---|---|---|---|---|---|---|---|---|---|---|
| | #Walk | Cap. | Time (s) | #Walk | Cap. | Time (s) | #Walk | Cap. | Time (s) | |
| 4 × 4 | 414K | 3.97 | 1.39 | 132K | 3.97 | 0.42 | 243K | 4.02 | 0.0184 | **75** |
| 5 × 5 | 395K | 4.86 | 1.37 | 123K | 4.87 | 0.40 | 234K | 4.92 | 0.0185 | **74** |
| 6 × 6 | 391K | 5.70 | 1.48 | 121K | 5.70 | 0.43 | 235K | 5.75 | 0.0193 | **76** |
| 4 × 4 × 4 | 367K | 4.12 | 1.26 | 121K | 4.11 | 0.39 | 218K | 4.14 | 0.0179 | **70** |
| 5 × 5 × 5 | 361K | 4.96 | 1.29 | 113K | 5.04 | 0.37 | 218K | 5.01 | 0.0179 | **72** |
| 6 × 6 × 6 | 346K | 5.91 | 1.35 | 109K | 5.96 | 0.40 | 210K | 6.00 | 0.0190 | **71** |

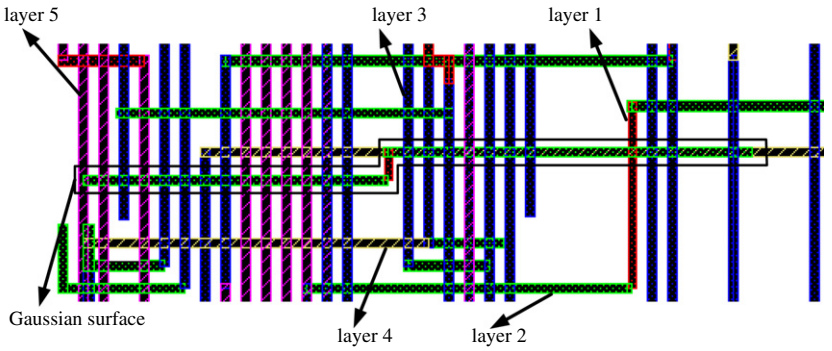


**Fig. 13.** A part of VLSI interconnect layout with five layers. The wires on different layers are distinguished by different color patterns.

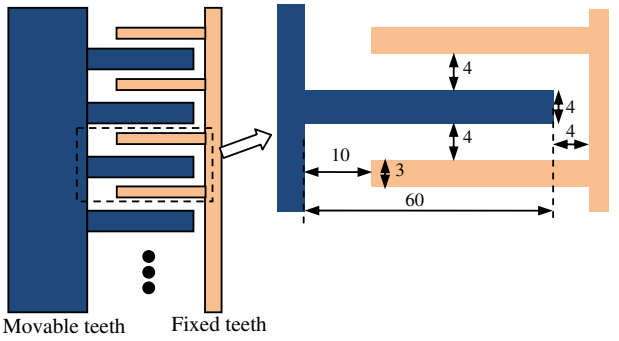


**Fig. 14.** A transverse comb drive with interdigitated teeth (the unit of dimension is μm).

**Table 6**
The computational results for two complex cases (unit of capacitance is $10^{-16}$ F).

| Case | FRW | | Original FRW (CPU) | | FRW + VR (CPU) | | FRW + VR (GPU) | | | | QBEM | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Cap. | Mem. (MB) | #Walk | Time (s) | #Walk | Time (s) | #Walk | Time (s) | Sp.[a] | Sp.[b] | #Panel | Cap. | Time (s) | Mem. (MB) |
| VLSI | 8.01 | 2.18 | 909K | 4.96 | 289K | 1.20 | 545K | 0.054 | **92** | **694** | 35,335 | 7.93 | 37.5 | 646 |
| MEMS | 371 | 1.39 | 1070K | 2.86 | 248K | 0.74 | 504K | 0.023 | **124** | **287** | 14,382 | 366 | 6.6 | 114 |

[a] The speedup ratio to the original FRW on CPU.
[b] The speedup ratio to the QBEM.

QBEM employs a quasi-multiple medium (QMM) accelerated BEM approach, which can be 10× faster than the multipole accelerated BEM for the capacitance calculation of VLSI interconnects [7]. From Table 6, we see that the discrepancy of capacitance obtained by the FRW method and QBEM is less than 2%. It is also found out that the proposed techniques bring about 100× speedup for the two cases. While comparing with QBEM, the GPU accelerated FRW is more than 200× faster. To compare the algorithms performed on CPU, with Table 6 we see that the FRW algorithm is about one order of magnitude faster than QBEM, but consumes much less memory. This exhibits the advantage of the FRW method over the deterministic methods like BEM while solving large structures.

## 6. Conclusions

In this work, two techniques are developed to accelerate the floating random walk (FRW) method for the electrostatic computation involving rectilinear shapes. The first technique explores the importance sampling and stratified sampling ideas, and significantly improves the convergence rate of the FRW algorithm. The second technique takes the benefit of the massively parallel computing based on GPUs; an iterative GPU-based FRW algorithm with GPU-friendly data structure is proposed. This is the first GPU-based parallel FRW algorithm reported in the literature, which employs the transition cubes. Numerical experiments are carried out on test cases of general interest, and the structures from real VLSI circuit and MEMS. With the former, we verify the accuracy and efficiency of the proposed techniques for calculating electric field

intensity and capacitance, while large efficiency is demonstrated with the latter cases in application areas. The results reveal that the variance reduction technique usually brings from 2× to 4× speedup without memory overhead and loss of accuracy, and can bring over 10× speedup for some special cases. The GPU-based FRW algorithm achieves over 20× speedup, as compared with the CPU-based serial computing. With the proposed techniques together, over 70× and up to 100× speedups are observed for the electrostatic computation in experiments. The accelerated FRW method is also compared with a fast BEM solver, which shows the former is several hundreds of times faster for the capacitance calculation of large structures.

Because of the limited device memory of GPUs (as compared with the main memory in computer), the GPU-based algorithm may not able to handle very large VLSI structures including thousands of conductor blocks directly. Extending the proposed techniques to chip-scale VLSI structures will be explored in the future.

## Acknowledgments

The authors acknowledge the financial support from National Natural Science Foundation of China under Contract No. 61076034, Tsinghua University Initiative Scientific Research Program, and Tsinghua National Laboratory for Information Science and Technology (TNList) Basic Research Project.

## References

[1] C.-O. Hwang, J.A. Given, M. Mascagni, The simulation–tabulation method for classical diffusion Monte Carlo, Journal of Computational Physics 174 (2001) 925–946.
[2] P. Lazic, H. Stefancic, H. Abraham, The Robin Hood method – a novel numerical method for electrostatic problems based on a non-local charge transfer, Journal of Computational Physics 213 (2006) 117–140.
[3] J.N. Jere, Y. Le Coz, An improved floating-random-walk algorithm for solving the multi-dielectric Dirichlet problem, IEEE Transactions on Microwave Theory and Techniques 41 (1993) 325–329.
[4] R.C. Garcia, M.N.O. Sadiku, Neuro-Monte Carlo solution of electrostatic problems, Journal of Franklin Institution 335B (1998) 53–69.
[5] N. Agarwal, N.R. Aluru, A stochastic Lagrangian approach for geometrical uncertainties in electrostatics, Journal of Computational Physics 226 (2007) 156–179.
[6] W. Yu, Q. Zhang, Z. Ye, Z. Luo, Efficient statistical capacitance extraction of nanometer interconnects considering the on-chip line edge roughness, Microelectronics Reliability 52 (2012) 704–710.
[7] W. Yu, Z. Wang, Enhanced QMM-BEM solver for three-dimensional multiple-dielectric capacitance extraction within the finite domain, IEEE Transactions on Microwave Theory and Techniques 52 (2004) 560–566.
[8] Y.I. Bontzios, M.G. Dimopoulos, A.A. Hatzopoulos, An evolutionary method for efficient computation of mutual capacitance for VLSI circuits based on the method of images, Simulation Modelling Practice and Theory 19 (2011) 638–648.
[9] P. Lazic, H. Stefancic, H. Abraham, The Robin Hood method – a new view on differential equations, Engineering Analysis with Boundary Elements 32 (2008) 76–89.
[10] K.K. Sabelfeld, Monte Carlo Methods in Boundary Value Problems, Springer-Verlag, New York, 1991.
[11] J.M. Hammersley, D.C. Handscomb, Monte Carlo Methods, Methuen, London, 1964.
[12] A. Haji-Sheikh, E.M. Sparrow, The floating random walk and its application to Monte Carlo solution of heat equation, Journal of SIAM on Applied Mathematics 14 (1966) 370–389.
[13] G.M. Royer, A Monte Carlo procedure for potential theory problems, IEEE Transactions on Microwave Theory and Techniques 19 (1971) 813–818.
[14] J.M. DeLaurentis, L.A. Romero, A Monte Carlo method for Poisson's equation, Journal of Computational Physics 90 (1990) 123–140.
[15] Y. Le Coz, R.B. Iverson, A stochastic algorithm for high speed capacitance extraction in integrated circuits, Solid-State Electronics 35 (1992) 1005–1012.
[16] R.B. Iverson, Y. Le Coz, A floating random-walk algorithm for extracting electrical capacitance, Mathematics and Computers in Simulation 55 (2001) 59–66.
[17] K.K. Sabelfeld, N.A. Simonov, Random Walks on Boundary for Solving PDEs, VSP BV, Utrecht, The Netherlands, 1994.
[18] D. Shia, C.Y. Hui, A Monte Carlo solution method for linear elasticity, International Journal of Solids and Structures 37 (2000) 6085–6105.
[19] M. Mascagni, N.A. Simonov, The random walk on the boundary method for calculating capacitance, Journal of Computational Physics 195 (2004) 465–473.
[20] C.-O. Hwang, M. Mascagni, T. Won, Monte Carlo methods for computing the capacitance of the unit cube, Mathematics and Computers in Simulation 80 (2010) 1089–1095.
[21] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, Numerical Recipes in C, second ed., Cambridge University Press, 1992.
[22] H. Qian, Y. Deng, B. Wang, S. Mu, Towards accelerating irregular EDA applications with GPUs, Integration, the VLSI Journal 45 (2012) 46–60.
[23] H. Zhuang, W. Yu, G. Hu, Z. Liu, Z. Ye, Fast floating random walk algorithm for multi-dielectric capacitance extraction with numerical characterization of Green's functions, in: Proc. ASP-DAC, January 2012, pp. 377–382.
[24] Y. Le Coz, H.J. Greub, R.B. Iverson, Performance of random walk capacitance extractors for IC interconnects: a numerical study, Solid-State Electronics 42 (1998) 581–588.
[25] NVIDIA Inc., NVIDIA's Next Generation CUDATM Computing Architecture: FermiTM, 2010.
[26] J. Novak, V. Havran, C. Dachsbacher, Path regeneration for random walks, in: W.-M.W. Hwu (Ed.), GPU Computing GEMS (Emerald Edition), Elsevier Inc., 2011, pp. 401–420.
[27] NVIDIA Inc., NVIDIA GPU Computing SDK, <http://developer.nvidia.com/gpu-computing-sdk>.
[28] NVIDIA Inc., NVIDA Toolkit, <http://developer.nvidia.com/cuda-toolkit-41>.
[29] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Transactions on Modeling and Computer Simulation 8 (1998) 3–30.
[30] S.H. Batterywala, M.P. Desai, Variance reduction in Monte Carlo capacitance extraction, in: Proc. the 18th International Conference on, VLSI Design, January 2005, pp. 85–90.
[31] X. Zhao, 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.
[32] T.A. El-Moselhy, I.M. Elfadel, L. Daniel, A hierarchical floating random walk algorithm for fabric-aware 3D capacitance extraction, in: Proc. ICCAD, November 2009, pp. 752–758.