Efficient Techniques for the Capacitance Extraction of Chip-Scale VLSI Interconnects Using Floating Random Walk Algorithm

Chao Zhang Wenjian Yu

Tsinghua National Laboratory for Information Science and Technology
Department of Computer Science & Technology
Tsinghua University, Beijing 100084, China
Tel : (8610)-62773440
e-mail : eric.3zc@gmail.com, yu-wj@tsinghua.edu.cn

Abstract – To enable the capacitance extraction of chip-scale large VLSI layout using the floating random walk (FRW) algorithm, two techniques are proposed. The first one is a virtual Gaussian surface sampling technique. It makes efficient random sampling on the Gaussian surface for complex nets with vias, and optimizes the sampling scheme to reduce the time of random walk. The other one is a parallelized, improved construction approach for Octree based space management structure. It can be over 5000X faster than the existing approach and provides same convenience to the FRW procedure. Numerical experiments on large cases with up to half million conductors validate the proposed techniques, and demonstrate a fast FRW solver for chip-scale extraction task.

I. Introduction

With the increase of the density of integrated circuit (IC), capacitances among the interconnects in IC have the growing impact on circuit performance. So, accurately and quickly extracting the interconnect capacitances is more and more important. The three-dimensional (3-D) field solver which directly solves the electrostatic field possesses the highest degree of accuracy. Despite the boundary element method (BEM) based field solvers [1-3] have been proposed, they are only able to handle small structures due to the bottleneck of runtime and memory. Differing from these deterministic methods, the floating random walk (FRW) algorithm was also proposed for capacitance extraction [4-9]. It is based on the Monte Carlo (MC) method, and on the field solver level of accuracy. Compared with the deterministic solvers, the FRW algorithm has the advantages of using less memory, better parallelism and tunable accuracy. More importantly, it is suitable for the chip-scale large interconnect structures, since its cost of computing time barely depends on the number of conductors involved.

The FRW algorithm has been developed into commercial software (such as *QuickCap*), despite with approximating approaches to adapt to complex VLSI structures and obtain high efficiency. An efficient FRW algorithm (called *RWCap*) was proposed to accurately extract the multi-dielectric VLSI structures, by pre-characterizing the transition domain including multiple dielectric layers and quickening the convergence of MC procedure with a comprehensive variance reduction scheme [6]. Efficient techniques were also proposed to parallelize the FRW based capacitance extraction on GPUs [9]. However, only medium-scale interconnect structures was tested in [6], and the technical details of the FRW algorithm for chip-scale large structures have not yet been reported in literature. In actual scenario, the distributed capacitances for a net (a number of connected interconnect wires) are needed. The environment of the nets, which constitutes chip-scale large structures, should also be handled to preserve high accuracy. This brings challenges to existing FRW algorithms, for the difficulties of handling the Gaussian surface of a whole net and millions of conductor blocks while executing the FRW procedure.

In this paper, two techniques are proposed to make the FRW algorithm effective for the chip-scale capacitance extraction task. An efficient virtual Gaussian surface sampling (VGSS) technique for a whole net with vertical vias is firstly proposed. It is versatile, and incorporates with the importance sampling on the Gaussian surface. The approach to optimize the placement of Gaussian surface for the FRW efficiency is also discussed. Then, an improved Octree construction algorithm is proposed to largely reduce the time for handling a large quantity of conductor blocks.

Experiments are carried out on several large scale VLSI layouts, and distributed full-net capacitances are extracted with 1% 1-σ error of total capacitance. Experimental results validate the effectiveness of the VGSS technique, and reveal that the optimized placement and sampling scheme of Gaussian surface could bring **1.4X** speedup to the FRW procedure. Compared with the existing approach, the improved Octree construction algorithm is over **5000X** faster. For a layout with 3037 nets including vias, the enhanced FRW solver is able to accomplish the extraction of all nets in about half an hour, on an 8-core CPU machine.

II. Background

The electric potential of a point $\boldsymbol{r}$ can be expressed as an integral of the potential on the surface S surrounding it:

$$\phi(\boldsymbol{r}) = \oint_S P(\boldsymbol{r}, \boldsymbol{r}^{(1)}) \phi(\boldsymbol{r}^{(1)}) \mathrm{d}\boldsymbol{r}^{(1)}, \quad (1)$$

where $\phi(\boldsymbol{r})$ is the potential of point $\boldsymbol{r}$ and $P(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ is called the surface Green's function. The domain enclosed by S is often called the transition domain. $P(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ is non-negative for any point $\boldsymbol{r}^{(1)}$ on S, and can be regarded as the probability density function (PDF) for selecting a random point on S. With the principle of Monte Carlo (MC) simulation, $\phi(\boldsymbol{r})$ can be estimated as the statistical mean of $\phi(\boldsymbol{r}^{(1)})$.

To compute the capacitances related with conductor i (called master conductor), a Gaussian surface G is firstly constructed to enclose it (see Fig. 1). According to the

This work was supported in part by NSFC under Grant 61076034, the Beijing Natural Science Foundation under Grant 4132047, the Opening Foundation of ASIC and System State Key Laboratory (Fudan University, No. 12KF009), and the Tsinghua University Initiative Scientific Research Program.
*W. Yu is the corresponding author.

978-1-4799-2816-3/14/$31.00 ©2014 IEEE

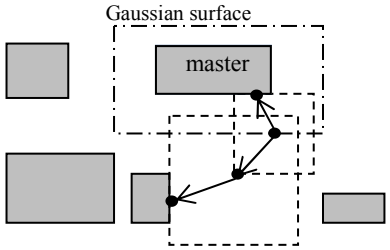


Fig. 1. Two examples of the random walk in the FRW algorithm for capacitance extraction (a 2-D cross-section view).

Gaussian theorem, charge Q of conductor i becomes

$$Q = \oint_G F(\boldsymbol{r}) \int_{S^{(1)}} \omega(\boldsymbol{r}, \boldsymbol{r}^{(1)}) q(\boldsymbol{r}, \boldsymbol{r}^{(1)}) \phi(\boldsymbol{r}^{(1)}) \mathrm{d}\boldsymbol{r}^{(1)} \mathrm{d}\boldsymbol{r} \quad . \qquad (2)$$

where $F(\boldsymbol{r})$ is the dielectric permittivity at point $\boldsymbol{r}$, $q(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ is the PDF for sampling on $S^{(1)}$ which may be different from $P(\boldsymbol{r}, \boldsymbol{r}^{(1)})$, and $\omega(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ is the weight value. Thus, Q can be estimated as the statistical mean of sampled values on G, which is further the mean of sampled potentials on $S^{(1)}$ multiplying the weight value. When $\phi(\boldsymbol{r}^{(1)})$ is unknown, (1) can be applied recursively to (2), which means the sampling procedure repeats until the potential of a sample point is known. This can be interpreted as a floating random walk (FRW) procedure: for the ith hop of a walk, a transition domain centered at $\boldsymbol{r}^{(i-1)}$ is constructed and then a point $\boldsymbol{r}^{(i)}$ is randomly selected on its boundary according to the discrete probabilities obtained with $P(\boldsymbol{r}^{(i-1)}, \boldsymbol{r}^{(i)})$. The walk terminates after k hops if $\phi(\boldsymbol{r}^{(k)})$ is known, e.g. it is on the surface of a conductor with known potential (see Fig. 1). With the techniques in [6], the weight value varies slightly, which reduces the random walks for the convergence of MC.

Although the surface Green's function for a spherical transition domain has simple analytical expression, we only consider the cubic transition domain that is well suited to the Manhattan-shaped interconnects in VLSI circuit. The surface Green's function only depends on the relative position of $\boldsymbol{r}^{(1)}$. So, we can pre-calculate and tabulate the sampling probability and weigh value for a unit-size cube. In multi-dielectric environment, the cubic transition domain filled by multiple dielectrics can also be pre-characterized [6].

The above deduction reveals that the statistical mean of the weight values for the walks terminating at conductor j approximates the capacitance C_{ij} between conductors i and j (if $j \neq i$), or the self-capacitance C_{ii} of master conductor i.

III. Techniques for Full-Net Capacitance Extraction

Calculating the delay of a full signal path, for which the interconnect delay caused by parasitic resistance and capacitance (RC) is a dominant factor, is a critical task in digital circuit design. To achieve accuracy, the interconnect wires (including vertical vias) in a net should be segmented and converted to a distributed RC network with parasitic extraction techniques. Due to the simplicity of resistance calculation, the capacitance extraction is the major challenge. The existing FRW algorithm should be enhanced to consider the existence of vias and produce the distributed capacitances, accurately and efficiently. This is also required for the accurate crosstalk analysis and circuit simulation.

Below, we will propose a virtual Gaussian surface sampling technique for the FRW-based full-net capacitance extraction. Considering the influence on the runtime of FRW algorithm, we will also discuss how to optimize the placement of Gaussian surface.

A. The basic idea

To obtain the capacitances for a net, which is made of a number of metal blocks, a simple approach is to construct one Gaussian surface for the whole net as if we only extract the capacitances related with the whole net. The key to get the distributed capacitances is to split the Gaussian surface into pieces and assign each block a piece. Thus, counting the walks starting from the Gaussian piece for a block, we can get the capacitances related to the block. In this way, the statistical error of a block capacitance could be larger than that of the full-net total capacitance, since the former is obtained with a smaller number of walks. However, due to the statistical cancellation, the error of derived net delay would be comparable to the error associated with the full-net total capacitance [7]. An example has shown that the statistical error of Elmore delay is ±2.3%, while the error of the full-net total capacitance is ±2% [7]. Therefore, setting a moderate accuracy criterion for the full-net total capacitance can ensure the accuracy we need for the sequent analysis.

It is a problem to generate the Gaussian surface for a net with complicated geometries of blocks and vias. A straightforward idea is to generate the Gaussian surface for each block and then calculate the envelope of them with geometric operations. However, this could cost a lot of computing time and bring complication to the representation and sampling of the Gaussian surface, which is not necessary. The FRW algorithm actually needs not to know what the Gaussian surface exactly is. All it needs is to make random sampling on it. So, we propose a virtual Gaussian surface sampling (VGSS) technique, which avoids calculating the envelope of the block's Gaussian surface (BGS). By suitably discarding invalid samples, the VGSS technique ensures the correctness and high efficiency.

B. The virtual Gaussian surface sampling technique

For each block in a net, we can construct its BGS as if its adjacent blocks in the net do not exist. If getting the BGSs for all blocks in a net, they may intersect with each other. This makes some parts of BGSs illegal for sampling. The idea of VGSS is selecting points on the BGSs and discarding illegal or invalid samples, so that the selected points can be treated as they are on the net's Gaussian surface which is not really generated. Take Fig. 2 as an example. Points like P_1 and P_2 are legal points, while P_3 within the BGSs of block 1 and the via is an illegal point. P_4 is also on legal area, which is however a place where two BGSs coincide. If we want to make uniform sampling on the Gaussian surface by sampling the BGSs, P_4 should be accept with probability of 1/2. With these considerations, the uniform sampling on the Gaussian surface is easily obtained.

It is not required that the Gaussian surface is sampled uniformly. Actually, some different sampling PDF may be

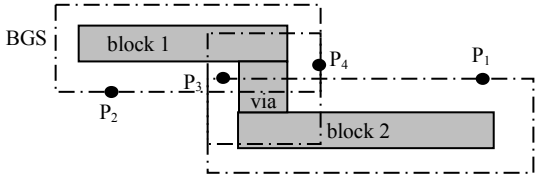


Fig. 2. Side view of two blocks and a via' BGSs.

used to take advantage of the importance sampling technique. With a non-negative function $p(\boldsymbol{r})$ that satisfies

$$\oint_G p(\boldsymbol{r})\mathrm{d}r = H \neq 0\,, \tag{3}$$

Eq. (2) can be transformed to

$$Q = H\oint_G \frac{p(\boldsymbol{r})}{H}\oint_{S^{(1)}} \frac{F(\boldsymbol{r})}{p(\boldsymbol{r})}\omega(\boldsymbol{r},\boldsymbol{r}^{(1)})q(\boldsymbol{r},\boldsymbol{r}^{(1)})\phi(\boldsymbol{r}^{(1)})\mathrm{d}\boldsymbol{r}^{(1)}\mathrm{d}\boldsymbol{r}\quad . \tag{4}$$

This means that we can use $p(\boldsymbol{r})/H$ as the PDF to sample the Gaussian surface. To achieve this, we can firstly make a uniform sampling with the aforementioned approach, and then accept the sample with probability of $p(\boldsymbol{r})/U$, where U is a constant value no less than the maximum of $p(\boldsymbol{r})$.

The efficiency of the FRW algorithm, i.e. the number of walks needed to reach a specified accuracy, is affected by the choice of $p(\boldsymbol{r})$. The basic option is $p(\boldsymbol{r})$=1, which means uniformly sampling. The second choice is $p(\boldsymbol{r}) = F(\boldsymbol{r})$, which could remove the variance on the weight value brought by (4). In [8], an IS technique based on electric field is proposed, which samples the Gaussian surface with a PDF proportional to $1/D(\boldsymbol{r})$. $D(\boldsymbol{r})$ denotes the distance of point $\boldsymbol{r}$ to the master conductor along Gaussian surface's normal direction. Following this strategy, the third choice of $p(\boldsymbol{r})$ could be $1/D(\boldsymbol{r})$. Finally, we can also try the choice $p(\boldsymbol{r}) = F(\boldsymbol{r})/D(\boldsymbol{r})$. For the four choices of the importance sampling on Gaussian surface, experiments are presented in Section V.A to see which one produces the best performance.

In (4), the weight value for each random walk should be multiplied by $HF(\boldsymbol{r})/p(\boldsymbol{r})$. So, the left problem is calculating H. If we apply the MC method to (3), H can be estimated with $S_G \cdot \overline{p}(\boldsymbol{r})$, where S_G is the area of Gaussian surface and $\overline{p}(\boldsymbol{r})$ is the statistical mean of $p(\boldsymbol{r})$. Suppose N_G points $\boldsymbol{r}$ on the Gaussian surface are generated during the VGSS sampling procedure. With them we get $\overline{p}(\boldsymbol{r})$. Suppose we totally generate N_T points on the BGSs, in which N_G points forms the uniform sampling of the Gaussian surface. Because the sampling on the BGSs is also a uniform one, N_G/N_T approximates the area ratio of the Gaussian surface to the sum of areas of BGSs. Therefore, $S_G \approx S_{all}\cdot N_G/N_T$, where S_{all} is the sum of areas of BGSs. Note that the samples on the Gaussian surface, equal to the number of walks, is usually at least several thousands. So, we can calculate H, and thus the weight value for (4) very accurately.

The following Algorithm describes how to get a valid sample on the Gaussian surface with the VGSS technique. $n_c(\boldsymbol{r})$ denotes the number of coinciding BGSs on point $\boldsymbol{r}$.

Algorithm 1 SampleOnVGSS (BGS list)

1. Randomly select BGS_i from the BGS list according to their areas;
2. Randomly select a point $\boldsymbol{r}$ on BGS_i uniformly;
3. **If** $\boldsymbol{r}$ is in the domain enclosed by other BGS **then goto** 1;
4. **Elseif** $\boldsymbol{r}$ is also on some other BGS whose outer normal direction is opposite to that of BGS_i **then goto** 1; **Endif**
5. Get two uniform random numbers in (0, 1): x_1, x_2;
6. **If** $x_1 > 1/ n_c(\boldsymbol{r})$ **then goto** 1 **Endif**
7. **If** $x_2 > p(\boldsymbol{r})/U$ **then goto** 1 **Endif**
8. **Return** $\boldsymbol{r}$.

The VGSS is a rejection sampling procedure. When we sample each BGS separately and uniformly, it actually give samples on G_0 with PDF $f_1(\boldsymbol{r}) = n_c(\boldsymbol{r})/N_c$, where $G_0 = \{\boldsymbol{r} \mid \boldsymbol{r}$ is on a BGS's surface$\}$ and $N_c = \oint_{G_0} n_c(\boldsymbol{r})\mathrm{d}\boldsymbol{r}$. In the VGSS, each of the samples on BGSs is only accepted with a probability $P_{ac}(\boldsymbol{r})$. When $\boldsymbol{r} \in G$, $P_{ac}(\boldsymbol{r}) = 1/n_c(\boldsymbol{r})\cdot p(\boldsymbol{r})/U$. Otherwise, $P_{ac}(\boldsymbol{r}) = 0$. According to the principle of rejection sampling [11], the accepted samples obey the PDF proportional to $f_0(\boldsymbol{r}) = f_1(\boldsymbol{r})\cdot P_{ac}(\boldsymbol{r}) = p(\boldsymbol{r})/(UN_c)$, for $\boldsymbol{r} \in G$. It is otherwise 0. This is the function what we want for sampling the Gaussian surface as in (4), since U and N_c are both constants.

Because points may be discarded, the selection will repeat several times until get a valid point. The time cost of selecting one point is about 1% of the cost of a random walk. So, as long as the repetition time of sampling is not very large, it will not harm the performance. The experiments reveal that the average repetition time is always less than 5.

In line 3 and 4 of Algorithm 1, we need to judge if a point is within or on some BGS. This can be done by comparing the point with all the BGSs. But it could be slow since a net may contain more than 50 blocks. To do this quickly, we pre-calculate each BGS's neighbor BGS list. Two BGSs are neighbors if they intersect. Only the BGSs in the neighbor list of the source BGS, which the point is selected from, need to be tested. This neighbor list is normally very short, so the efficiency of the judging can be high enough.

C. Optimize the placement of Gaussian surface

Before discussing how to determine the optimal placement of a BGS, we first give some definitions about distances. Symbols A and B denote geometric objects which may be cuboids, points or faces. The last two are the special cases of a cuboid. D denotes one of directions X, Y and Z. S denotes one of signs P (positive) and N (negative).

Definition 1: PD-distance from A to B is B's minimum D-coordinate minus A's maximum D-coordinate which is also ND-distance from B to A. d_{PD}(A, B) and d_{ND}(A, B) denote the PD- and ND-distance from A to B respectively.

Definition 2: The distance between A and B, denoted by d(A, B) is the maximum of d_{PX}(A, B), d_{NX}(A, B), d_{PY}(A, B), d_{NY}(A, B), d_{PZ}(A, B), d_{NZ}(A, B).

The placement of a BGS is determined by six distances between it and the enclosed conductor block along six directions (PX, PY, PZ, NX, NY and NZ). The placement of Gaussian surface has an impact on the performance of FRW algorithm. In [8], each face of the Gaussian surface is set to be exactly at the middle of the block and its nearest block. Therefore, points on the Gaussian surface are equidistant from the master and from an adjacent block, and the first cubic transition domain of random walk can touch two blocks. This gives the random walk more probability to stop after the first hop. However, this strategy is not suitable for the master conductor with rare blocks around, where the positions of Gaussian surface in different directions may have very large disparity. And, the larger distance from the Gaussian surface to the conductor also worsens the convergence behavior of the FRW procedure. The authors of [6] firstly find the six equidistant positions from the master, and then use the minimum of the six distances to construct the Gaussian surface. However, this may cause a very small BGS, which is also not good to the efficiency of FRW.

Below, we present a general strategy to determine the Gaussian surface, which compromises those used in [6] and [8]. The effect of multiple dielectric layers is also considered. The position of Gaussian surface is adjusted to guarantee that there is probability of terminating the random walk just after

one hop in a two-dielectric transition cube. Algorithm 2 describes how to determine the placement of a BGS. A *scale_factor* not less than 1 is used, whose best value will be discussed in Section V.A. If *scale_factor*=1, the approach is that in [6]; if *scale_factor* is infinite, it is that in [8].

Algorithm 2 GenerateBGS (master block A, other blocks)

```
1.  For signed direction K in {PX, PY, PZ, NX, NY, NZ} do
2.      dm := inifinty;
3.      For each other block B do
4.          If d_K(A, B) = d(A, B) then
5.              dm := min(dm, d(A, B))
6.          Endif
7.      Endfor
8.      D[K] := dm / 2;
9.  Endfor
10. For K in {PZ, NZ} do
11.     Start from A's K-face and find the second dielectric
        interface I on A's K-direction.
12.     If d(A, I) < D[K] then D[K] := d(A, I) / 2 Endif
13. Endfor
14. d := min(D[PX], D[PY], D[PZ], D[NX], D[NY], D[NZ]);
15. d := d * scale_factor
16. For K in { PX, PY, PZ, NX, NY, NZ} do
17.     D[K] := min(d, D[K]);
18. Endfor
19. Use six distances D[PX], D[PY], D[PZ], D[NX], D[NY] and
    D[NZ] to construct the BGS.
```

IV. Parallel Space Management Techniques

For each hop of a walk, a conductor-free cubic transition domain is needed. And, we hope it could be as large as possible. Thus, we need to find the distance from current point to its nearest block as the cubic transition domain's radius. A space management structure which indexes all the blocks will helpfully accelerate this. With such approach, only the distances to a few of blocks, which are in the so-called "candidate list" are calculated for each hop.

Octree is a widely used space management structure. The basic idea of Octree structure is dividing the problem domain into small subdomains and organizing them as a tree [10]. Each non-leaf node has 8 children and each leaf node has a candidate list which contains every possible nearest block from any point in the node (subdomain). To find the nearest block from a specified point, we can firstly find the leaf node that contains the point and then compare every block in that node's candidate list to find the nearest one. The candidate list is usually not long, so the Octree has very high performance in finding the nearest block.

Construction of Octree can be done by inserting all blocks into an empty Octree's root one by one. There are two thresholds in Octree. One is called $\boldsymbol{n_t}$. When a node's candidate list contains more than n_t blocks, the node should be divided into 8 subnodes. To avoid the tree growing too high, we set a minimal size limit $\boldsymbol{l_t}$ for node. The primary operation for constructing the Octree is checking if a block should be added into a node's candidate list. This is judged with the domination relationship defined as follows [6].

Definition 3: T is a node, and B_1, B_2 are two blocks. If for any point $P \in T$, and $P \notin B_1 \cup B_2$, $d(P, B_1) \le d(P, B_2)$, we say B_1 *dominates* B_2 regarding T.

A block will not be inserted into a node's candidate list unless it's not dominated by any other blocks in that candidate list. Thus, the domination relationship should be judged between the block and every block in the node's candidate list. This could cost much runtime while handling a structure with a lot of blocks. For an example structure with 37 thousand blocks, the original Octree construction approach can take half an hour. Note that extracting the capacitances of a net in this structure only costs several seconds. In this section, we will propose the techniques to accelerate the Octree construction.

A. The pruning skills for judging domination relationships

We firstly give two definitions.

Definition 4: The *size* of a cuboid node is defined as the maximum of the node's length, width and height.

Definition 5: The *distance limit* $L(T)$ of an Octree node T is the minimum distance between it and a block in its candidate list, plus the node's size.

Actually, the distance limit $L(T)$ is an upper bound of the distance to nearest block from any point in node T. For a new block B, the candidate list of a leaf node T and block B_0 in the list, if $d(B, T) \ge L(T) = d(B_0, T)+Size(T)$, then B is dominated by B_0. Because for any point P in the conductor-free space within T, we have $d(P, B_0) \le d(T, B_0)+Size(T) = L(T) \le d(T, B) \le d(P, B)$. When a new block is added to a node's candidate list, the value of distance limit dynamically changes. Algorithm 3 describes the candidate checking procedure. For most blocks, it returns at beginning instead of after testing all blocks in the candidate list.

Algorithm 3 CandidateCheck(block B, node T)

```
1.  d := d(B, T); l is the size of T;
2.  If d ≥ L(T) then return false;
3.  For each b in the candidate list of T do
4.      If b dominate B then return false;
5.      Elseif B dominate b then
6.          Remove b from the candidate list of T;
7.      Endif
8.  Endfor
9.  Add B to the candidate list of T;
10. If (d + l) < L(T) then L(T) := d + l; Endif
11. Return true.
```

The distance limit of a node is initialized with infinity since the first tried block should be added to the candidate list. However, we can set it to be a finite value d_{nb}, to prune more block testing. In this way, the candidate list no longer contains all of possible nearest block but only the blocks in the node's *neighbor region* (see Fig. 3). When the candidate list of a node is incomplete, more operation is needed after finding the minimum distance from current point to the candidates. Because maybe a block out of the neighbor region is the nearest block to current point, the minimum distance should be compared with the distance between the point and the boundary of neighbor region, and the smaller one (such as d_b rather than d_3, in Fig. 3) should be used to guarantee a conductor-free transition domain.

Because the transition domain obtained by inquiring the

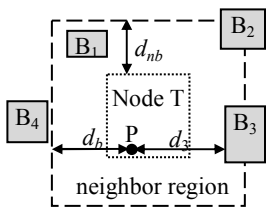


Fig. 3. The illustration of a point in node, node's neighbor region and calculating the nearest distance.

incomplete candidate list is not the largest, it would increase the number of hops in a walk. But with an appropriate value of d_{nb}, this shortcoming is negligible. Our experiments tell that the distance limit and setting the neighbor region can accelerate the construction by thousands times.

B. *Parallel construction of the Octree*

The Octree structure has great potential in parallelization. Finding the nearest block is embarrassingly parallel, since all the operations on Octree are read-only. The construction of Octree can also be parallelized with a little modification. Here, we come up with two different ways to parallelize it with multi-thread model.

In the parallel construction of Octree, we need to avoid different threads accessing same tree node simultaneously. One way for parallelization is adding a lock to each node and all the accesses need to acquire the lock. Because the insertion of candidate to Octree is performed recursively, the lock of parent node should be released before trying its children nodes. Otherwise, the entire tree would be locked. The cost of this method has two parts: the cost of lock operation itself and the cost of waiting same lock. The first part depends on the implementation of lock. Since only basic functions are needed for the lock, an atomic variable is enough and able to keep the cost of lock at the lowest. The second part can be reduced by two ways: traversing the children nodes in random order or temporarily skipping the occupied node. They both reduce the probability that two threads accessing same node. However, they are not very helpful, because with the distance limit the operation on a node is usually simple and fast, and they won't help to avoid frequently waiting for the root.

Another way of parallelization is letting different threads inserting all blocks into different parts of the Octree. Thus, the lock will be needed any more. We split the root node into 8 subnodes. Each of them represents a subtree and can be further split to get more subtrees if necessary. It's better not to split the tree too many times since the splitting times decide the minimal height of the Octree which affects the efficiency to locate a leaf node.

Each subtree will be assigned to a thread and all the blocks will be tried for insertion. With above pruning skills, most blocks will only be tested with the subtree's root and affirmed not in any node's candidate list of this subtree. This guarantees each thread only has few work to do. Because different region contains different number of blocks, the workload of different subtree may be very different. To balance all the threads, the number of subtrees should be much larger than the number of threads.

Our experiments tell that the second method with several subtrees works much better than the first one with lock. 4.5X speedup can be achieved on a machine with 8-core CPU.

V. Numerical results

We have implemented the FRW algorithm [6], and the proposed techniques in C++. The obtained enhanced FRW algorithm is used to extract the full-net distributed capacitances for several large-scale VLSI layouts.

The first test case is an artificially generated complex structure, which includes a net formed by 11 blocks and 6 vias in 3 metal layers, along with other 12 nets. The second case is an actual design called "FreeCPU" based on the *180-nm technology* with the minimum wire width of 200nm. It includes 37,062 conductor blocks in 5 metal layers, which forms 3037 nets with vias. There are 12 dielectric layers. The lateral and vertical dimensions of the case are about 700μm and 9.4μm, respectively. The third case is an artificially created layout based on the *45-nm technology* with the minimum wire width of 70nm. It includes 101,595 conductor blocks in 3 metal layers, with random widths, spacing, lengths and positions. And, it contains 9 dielectric layers. The lateral size of Case 3 is about 1000μm. The fourth case is an even larger case with 484,441 conductor blocks, based on the 180-nm technology. Its parameters are similar to the "FreeCPU" case.

We firstly verify the correctness of the VGSS technique, and reveal the best sampling PDF and placement of the Gaussian surface. Then, the efficiency of the pruning skills and parallelization techniques for constructing the Octree is demonstrated. Experiments are carried out on a Linux server with Intel Xeon E5-2650 8-core CPU of 2.0 GHz, while the accuracy criterion of FRW algorithm is 1% 1-σ error.

A. *Validate the VGSS technique and optimized parameters*

For Case 1, we have manually generated the real Gaussian surface, which is represented by several rectangular planes. Random walks are executed with the VGSS technique and on the real Gaussian surface, respectively. The results show the both programs run same number of walks, and the resulting capacitances have less than 1% discrepancy, consistent to the set accuracy criterion.

The *scale_factor* in Algorithm 2 decides the placement of Gaussian surface and affects both the number of walks and the number of hops per walk. We set it to different values to find out the best one. We also compare the different Gaussian surface sampling PDFs. 33 randomly selected nets in each of Case 2 and 3 are extracted. The total time for performing FRW procedure is plotted in Fig. 4.

From Fig. 4, we see that setting *scale_factor*=1.25 and $p(\boldsymbol{r})=F(\boldsymbol{r})$ brings the best efficiency. A larger *scale_factor* increases the number of walks for convergence. And, the caused larger transition cube causes smaller probability to terminate after the first hop. Meanwhile, a very small factor also causes more hops in a walk. Comparing with the situation with *scale_factor*=1 (used by [6]), the optimized placement of Gaussian surface could bring **1.4X** speedup to the FRW procedure, as shown in Fig. 4(a). Compared with the strategy in [8], the speedup could be even larger. As for the sampling scheme on the Gaussian surface, the advantage of $p(\boldsymbol{r})=F(\boldsymbol{r})$ over other PDFs are obvious. This again validates

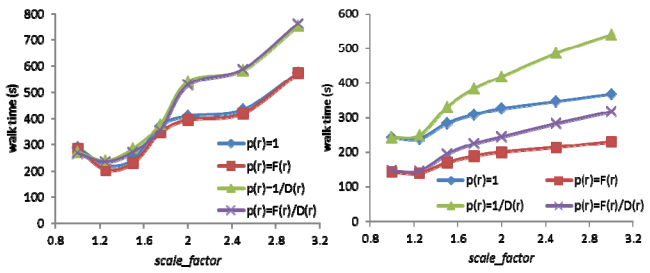


Fig. 4. The time of FRW procedure for extracting 33 nets in Case 2 (a) and Case 3 (b), with varied Gaussian surface placement and $p(\boldsymbol{r})$.

the importance of the variance of the weight values to the convergence rate. For Case 3, with $p(\boldsymbol{r})=F(\boldsymbol{r})$ the average number of walks is 245182, whereas it is 422909 (72% larger) if we make uniform sampling ($p(\boldsymbol{r})=1$). Finally, we have counted the repetition time of sampling for each PDF choices. The results reveal that the repetition time is always less than 5, which means the overhead of the VGSS technique is negligible, and hardly affects the total runtime.

B. *Validate the improved Octree construction algorithm*

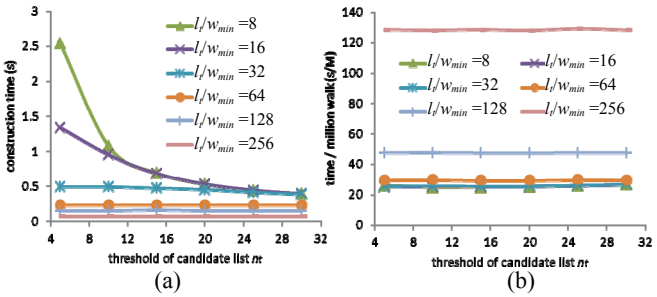
We firstly use Case 2 to find out suitable values for thresholds n_t and l_t considering both the Octree construction time and the performance of FRW procedure. Because cases with different process technology may need different value of l_t, we use l_t/w_{min} as the variable, where w_{min} represents the minimum wire width. The Octree construction time and the time for performing a million FRW walks under different settings of n_t and l_t are shown in Fig. 5. From it, we see the construction time decreases when l_t/w_{min} or n_t increases. It is because the Octree includes fewer nodes with larger l_t/w_{min} or n_t. The time for performing walks changes little when n_t increases. This is because larger n_t increases the time for traversing the candidate list, while reducing the height of Octree. As for the l_t/w_{min}, it affects only when it is very large ($\geq$ 128 in this experiment) causing a very long candidate list. Therefore, to balance the efficiency of Octree construction and the FRW procedure, we set l_t/w_{min} to 32, and n_t to 25.

In Table I, the Octree construction times for the three large cases are listed. It reveals that, with the proposed pruning techniques, our approach can be over **5000X** faster than the approach in [6]. The improved approach costs only 12 seconds for a large case with half a million blocks.

We use the largest Case 4 to test the parallel Octree construction technique. The approach using lock achieves 2.2X speedup of parallelization, while running 8 threads on the machine with 8-core CPU. On the contrary, the approach with several subtrees performs better. It achieves 4.4X and **4.5X** speedups for 8 and 12 threads, respectively. The experiments demonstrate that the improved parallel Octree construction costs very short time, and greatly facilitates the capacitance extraction of even larger layout structures.

TABLE I
The Construction Time of the Octree (in unit of second)

Case	#block	Approach of [5]	Proposed approach	Speedup
2	37062	1757.9	0.53	3316
3	101595	16595.6	3.12	5319
4	484441	> 2 days	12.27	--



(a) (b)

Fig. 5. The trends of the Octree construction time (a) and the time for performing a walk (b), with different n_t and l_t/w_{min}.

C. *Results of full-chip capacitance extraction*

Full-chip capacitance extraction producing all distributed capacitances for all nets is performed. With the proposed techniques, optimized parameter settings and 8-thread multi-core parallelization, the enhanced FRW solver completes the full-chip extraction for Case 2 in 35.2 minutes. This means an average extraction speed of about **0.7 second per net**, under the accurate 1% error criterion. Similar high speed of extraction is also observed on other test cases.

VI. Conclusions

The proposed techniques provide efficient solutions to the problems faced in extracting full-net distributed capacitances and handling chip-scale large interconnect structures using the FRW algorithm. With them, the major advantage of the FRW solver over the deterministic solvers can therefore become reality, which makes executing full-chip capacitance extraction directly with field solver possible. The techniques are crucial to satisfy the increasing accuracy demand in the capacitance extraction of VLSI designs.

References

[1] K. Nabors and J. White, "FastCap: A multipole accelerated 3-D capacitance extraction program," *IEEE Trans. Computer-Aided Design*, 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. Microwave Theory Tech.*, Vol. 52, No. 2, pp.560-566, 2004.

[3] W. Yu, X. Wang, Z. Ye, and Z. Wang, "Efficient extraction of frequencydependent substrate parasitics using direct boundary element method," *IEEE Trans. Computer-Aided Design*, Vol. 27, No. 8, pp. 1508–1513, Aug. 2008.

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

[5] T. A. El-Moselhy, I. M. Elfadel, and L. Daniel, "A hierarchical floating random walk algorithm for fabric-aware 3D capacitance extraction," in *Proc. ICCAD*, 2009, pp. 752-758.

[6] W. Yu, H. Zhuang, C. Zhang, G. Hu and Z. Liu, "RWCap: A floating random walk solver for 3-D capacitance extraction of VLSI interconnects," *IEEE Trans. Computer-Aided Design*, Vol. 32, no. 3, pp. 353–366, Mar. 2013.

[7] M. Kamon and R. Iverson, "High-accuracy parasitic extraction," in *EDA for IC Implementation, Circuit Design, and Process Technology*, CRC Press, Boca Raton, FL, 2006.

[8] S. H. Batterywala and M. P. Desai, "Variance reduction in Monte Carlo capacitance extraction," in *Proc. 18th Int. Conf. VLSI Design*, Jan. 2005, pp. 85–90.

[9] W. Yu, K. Zhai, H. Zhuang, and J. Chen, "Accelerated floating random walk algorithm for the electrostatic computation with 3-D rectilinear-shaped conductors," *Simulation Modelling Practice and Theory*, Vol. 34, No. 5, pp. 20-36, 2013.

[10] H. Samet, *Applications of Spatial Data Structure*, Addison-Wesley, Reading, MA. 1990.

[11] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes in C*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 1992.