Realizing Reproducible and Reusable Parallel Floating Random Walk Solvers for Practical Usage

Mingye Song*, Zhezhao Xu*, Wenjian Yu*, Lei Yin†
*BNRist, Dept. Computer Science & Technology, Tsinghua University, Beijing 100084, China
†ANSYS, Inc., 2645 Zanker Road, San Jose, CA 95134, USA
Email: songmy16@mails.tsinghua.edu.cn, zhezhaoxu@gmail.com, yu-wj@tsinghua.edu.cn, lei.yin@ansys.com

***Abstract*—Capacitance extraction or simulation has become a challenging problem in the computer-aided design of integrated circuits (ICs), flat panel display, etc. Due to its scalability and reliability, the parallel floating random walk (FRW) based capacitance solver is widely used. In practice, the parallel FRW algorithms involve an issue of reproducibility and may consume a lot of time in the scenario requesting high accuracy. To relieve these issues, techniques are developed in this paper to enhance the reproducibility and reusability of the parallel FRW based simulation. With them we ensure that same result is reproduced while rerunning the parallel FRW solver with same setting. A "jump start" feature is also implemented to reduce the total runtime of simulating same structure with multiple accuracy criteria. Experiments on shared-memory and distributed-memory platforms have validated the effectiveness of the presented techniques. Compared with a synchronization based approach ensuring the reproducibility, the proposed technique with static workload allocation can brings 4.8X more parallel speedup while sacrificing nothing.**

***Index Terms*—Capacitance extraction/simulation, floating random walk (FRW) method, high-performance computing, reproducibility.**

I. Introduction

Accurate capacitance simulation with three-dimensional (3-D) field solver has been applied to the design of advanced integrated circuits (ICs) and flat panel displays (including touchscreens) [1]–[4]. For IC design, it enables accurate device/interconnect capacitance modeling which is necessary for the verification of circuit performance metrics, and provides a design validation tool for on-chip capacitor structures. For the flat panel display design, the accurate capacitance solver is required for verifying the timing constraints on signal wires and the functionality of touch sensor structures [3].

There are mainly three kinds of methods for 3-D capacitance field solver [1]: domain discretization method, boundary element method, and the floating random walk (FRW) method [5], [6]. Compared to the others, the FRW method is more stable in accuracy and scalable to large cases, due to its nature of a discretization-free method. Its another advantage is the parallelizability, as it's based on the Monte Carlo method. This makes the FRW based capacitance solver popular nowadays, also due to the easy availability of parallel computing [3], [7]–[9]. Recently, the FRW method was extended to efficiently handle cylindrical inter-tier-vias in 3-D ICs [10] and the non-Manhattan conductors in packaging and touchscreen [11]. A more reliable and accurate approach was proposed in [12] to enhance the FRW method for simulating the structure with general floating metals, while distributed parallel techniques were proposed in [9] for efficient capacitance simulation on a computer cluster. For the variation-aware capacitor modeling, an efficient technique based on the FRW method was also developed [13].

In this work, our concern is the issues of parallel FRW capacitance solvers in practical usage. The first one is the numerical reproducibility, which means obtaining the same result when the simulation is run several times either on the same machine or on different machines [14]. Parallel programs often result in the reproducibility issue because: 1) the parallel work can be allocated in ways that are not specifiable at compile time, and 2) the execution often proceeds in an opportunistic manner with the execution order changing between runs so that non-commutative floating-point computations produce varying results. The reproducibility is a cornerstone of the scientific method, and has drawn a lot of attention in the community of high-performance computing [15]–[17]. However, it is not addressed in the research of parallel FRW algorithm. Existing work in [6], [7], [9] cannot guarantee this reproducibility. Another issue is how to improve the reusability of parallel FRW based simulation. Its aim is to reduce the long runtime of FRW based solver for the situation requesting high accuracy. A practical scenario is that we first simulate a structure with a low accuracy criterion in order to complete it quickly, and then we may simulate the same case towards a higher accuracy with longer time budget. If the previous run (simulation) is reusable, the time spent for the subsequently runs can be reduced. This is not a hard problem, but not addressed in previous literature.

The aforementioned reproducibility issue is also motivated by the customers of the FRW capacitance solver. They don't want to obtain the same different results with the same simulation parameters when testing or debugging. In this work, two techniques are developed. Firstly, a technique including fixed random seeds and static work allocation is proposed, which enables the parallel FRW algorithms 100% reproducing same capacitance results when the simulated structure, the termination criterion, and the number of threads/processes do not vary. Actually, our technique is able to make same results produced from the multi-thread solver and the multi-process solver run on different platforms, as long as the number of threads is the same as the number of processes used. And, it does not degrade the accuracy and runtime of the original

978-3-9819263-2-3/DATE19/©2019 EDAA

FRW solvers. Secondly, a simple "jump start" feature is implemented for the parallel FRW solvers. It reduces the simulation time of the subsequent runs of same structure without loss of accuracy. Experiments carried out on multi-core machines and a computer cluster have validated the effectiveness of the developed techniques for enhancing the reproducibility and reusablity of parallel FRW solvers. Compared with a synchronization based approach for ensuring the reproducibility, the proposed technique with allocated accuracy criteria is able to bring up to **4.8X** more parallel speedup on a simulation with 120 processes.

II. Background

A. 3-D FRW Method for Capacitance Simulation

The FRW method for 3-D capacitance field solver is originated from the integral formula for electric potential:

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

where $\phi(\boldsymbol{r})$ is the potential of point $\boldsymbol{r}$ and $P(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ is called surface Green's function. The domain containing $\boldsymbol{r}$ is called transition domain, whose surface is S. $P(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ can be regarded as a probability density function (PDF). With the Monte Carlo method, $\phi(\boldsymbol{r})$ can be estimated as the stochastic mean of $\phi(\boldsymbol{r}^{(1)})$.

When computing the capacitances related to a master conductor i, one should first construct a Gaussian surface G_i enclosing it (see Fig. 1). According to the Gauss theorem, and by substituting (1) into the electric field intensity (i.e. the gradient of electric potential), one can derive the expression of conductor i's charge:

$$Q_i = \oint_{G_i} F(\boldsymbol{r}) g \oint_{S^{(1)}} \omega(\boldsymbol{r}, \boldsymbol{r}^{(1)}) \tilde{P}(\boldsymbol{r}, \boldsymbol{r}^{(1)}) \phi(\boldsymbol{r}^{(1)}) d\boldsymbol{r}^{(1)} d\boldsymbol{r}. \quad (2)$$

$F(\boldsymbol{r})$ is the dielectric permittivity at point $\boldsymbol{r}$, $\tilde{P}(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ is the PDF for sampling on $S^{(1)}$, and $\omega(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ is called weight value [5], [6]. Thus, Q_i can be estimated as the stochastic mean of sampled values on G_i, which is further the mean of sampled potentials on $S^{(1)}$ multiplying the weight value. If the potential of a sample point $\boldsymbol{r}^{(1)}$ is unknown, Eq. (1) is substituted into (2) recursively. The computation can be regarded as a floating random walk (FRW) procedure. The walk starts from the Gaussian surface, and repeatedly jumps from a transition domain's center to its surface, until reaching conductor surface. After performing a number of walks, the stochastic mean of the weight values for the walks terminating at conductor j approximates the capacitance C_{ij} between conductors i and j. Notice that the cubic transition domain is widely adopted because it fits well the Manhattan-shaped interconnects in ICs. This means large probability for terminating a walk earlier. The sampling probability and weight value for a cube domain can be pre-calculated and tabulated, so as to accelerate the sampling operation.

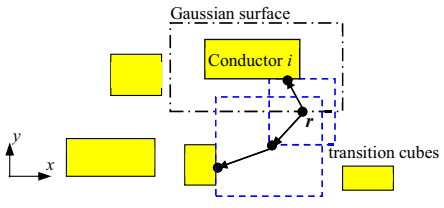

Fig. 1. Two random walks starting from $\boldsymbol{r}$ (denoted by consecutive segments with arrows) in the FRW method for capacitance simulation (2-D top view).

The FRW method is essentially a Monte Carlo (MC) method, where each random walk is a MC trial returning an estimate for certain capacitance and zeros for others. Due to the central limit theorem, the mean of a sequence of estimates approximately obeys the normal distribution whose standard deviation (Std) can be estimated with the variance of the sequence. So, the Std (called 1-σ error) depicts the accuracy of the capacitance result (mean value). It also brings an advantage of the FRW method. One can set a 1-σ error as accuracy criterion. And, once it is attained the FRW method can stop performing the random walks. In practice, the algorithm repeatedly checks the error after every certain number of walks and does not terminate until it is below the specified criterion.

B. Techniques for Parallel FRW Algorithms

The independence among the random walks makes the FRW method suitable for parallelization. In [6], an efficient multi-thread FRW algorithm for the multicore/multi-CPU platform was proposed. It makes the threads executing the random walks independently, except for checking the termination criterion per 1000 walks. Its flowchart is shown in Fig. 2.

To tackle larger computing task, the parallel FRW algorithms based on graphics processing units (GPUs) and distributed computing were also proposed [3], [7]–[9]. Due to the workload divergence among different walk paths, parallelizing FRW on GPUs is not easy, or can hardly achieve very high speedup. Thus, the distributed parallel FRW algorithm on computer cluster is more available. Such an algorithm was proposed in [3] and then improved in [9]. Both the random walk procedure and space management have been largely accelerated. It should be pointed out, though the existing work have made highly efficient parallel FRW algorithms, the

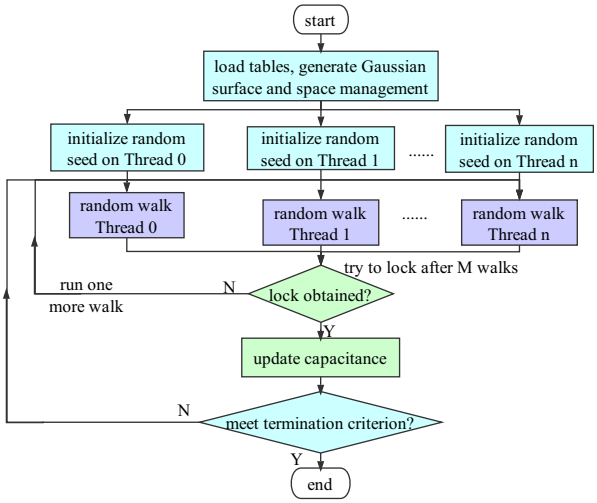

Fig. 2. Flowchart of the multi-thread FRW algorithm in [6].

reproducibility issue was not addressed. And, none of them can ensure the reproducibility of results.

III. Reproducible Parallel FRW Solvers

In this section, we propose techniques to realize reproducible parallel FRW capacitance solvers.

A. The Problem of Existing Parallel FRW Algorithms

Same as the MC method, the reproducibility of the FRW method ultimately rests on the reproducibility of the random numbers used [17]. For serial FRW algorithm, the latter can be easily accomplished by fixing the state or seed of the random number generator. However, for parallel algorithm things become complicated. Firstly, there are multiple random number sequences generated in shared-memory or distributed-memory environment. Secondly, the uncertainties in the work allocation and execution order among threads/processes largely affect the reproducibility.

For the multi-thread FRW algorithm in [6], there are two operations which use random number. One is picking a random point on Gaussian surface, and the other is picking a random point on the surface of cubic transition domain. Even if we made sure that same random number sequences are generated in different runs, the reproducibility of the algorithm would not be guaranteed. Let us look at Fig. 2, the flowchart of the multi-thread algorithm, where the parts using random numbers are in purple and those related to the critical section of parallel computing are in light green (same coloring applies to other flowcharts in this paper). To achieve higher parallel efficiency, the work allocation of this algorithm follows the strategies: 1) M walks are performed for each thread before the capacitance value is updated (typically $M = 1000$); 2) for entering the critical section, a thread tries the lock (mutex), and if it fails it performs one more walk before next try. This brings uncertainty to the number of walks between a thread's two consecutive capacitance updates.

For the distributed FRW algorithms [3], [9], how to fix the random number sequences for the distributed processes was not mentioned. And, the uncertainties in work allocation and execution order prevent them from guaranteeing the reproducibility.

B. An Approach Including Synchronization with Barrier

To improve the reproducibility of the algorithm in Fig. 2, we can try not performing the extra walks after failing to obtain the lock. Instead, the thread just waits until obtaining the lock, which means it always performs a multiply of M walks. However, this cannot guarantee the reproducibility yet. Consider an extreme example with two threads. In the first run, thread 0 obtains the lock after performing M walks, and then the termination criterion is met after it updates the capacitance. So, the program terminates and the capacitance results are derived from the walks performed by thread 0. While in the second run, thread 1 obtains the lock after performing M walks. It also causes the termination of program, but the results are obtained from the walks performed by thread 1. Due to the difference of random number sequences in thread 0 and 1, the results of the both runs are different.

An approach including synchronization with barrier can resolve the problem, as described in Fig. 3. In such a way, every M walks are regarded as a cycle, and there is the synchronization of threads after all threads complete a cycle. With this approach, each time the program checks the termination criterion all threads have performed same cycles of random walks. This fixes work allocation. And as updating capacitance is simply sum operation, the execution order of threads affects little the results. Thus, this approach results in a reproducible multi-thread FRW algorithm. Its overhead is the waiting time for synchronization and the cost of barrier. Notice the barrier can be efficiently implemented with Pthreads, like using mutex and semaphores [18]. However, for the case with lots of cycles and threads, this approach will lose efficiency.

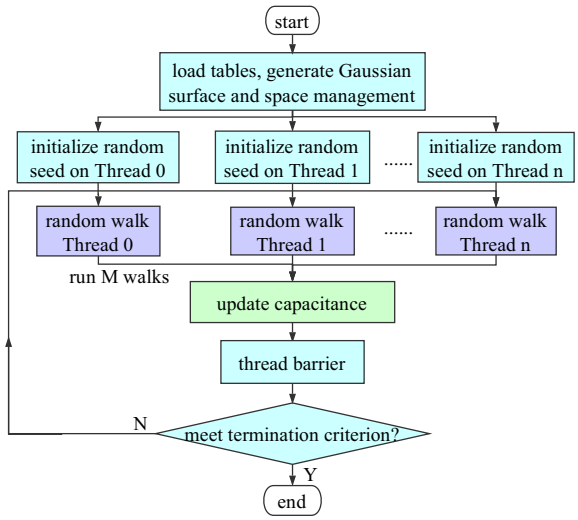

Fig. 3. Flowchart of a multi-thread FRW algorithm including synchronization with barrier.

We also employ different seeds, which are fixed, for the random numbers used in different threads. Due to the high quality of the random number generator [19], this preserves the independence among these random sequences (and thus the result accuracy) while guaranteeing the reproducibility.

C. An Approach with Allocated Accuracy Criteria

To pursue both the reproducibility and high efficiency, we propose a new approach for the multi-thread FRW algorithm. It statically allocates the work and termination criteria to the threads. We consider the case with accuracy criterion. Notice that the FRW algorithm is like a MC procedure that calculates integral $I = \int_a^b f(x)dx$ with $I \approx \bar{I} = \frac{1}{n}\sum_{i=1}^{n} f(x_i)$. In the FRW context, $f(x_i)$ is the weight value or zero returned from each random walk, and I stands for the wanted capacitances. According to the central limit theorem $\bar{I} \sim N(I, \sigma^2)$, and

$$\sigma = \sqrt{var(\bar{I})} = \sqrt{var(f(x))/n}\ , \tag{3}$$

where σ is the 1-σ error. Because the variance of integrand function $var(f(x))$ is a constant, we see that the 1-σ error of capacitance result from the FRW algorithm, denoted by err, is

inversely propositional to the square root of number of walks, i.e.

$$err \propto \frac{1}{\sqrt{N_{walk}}} \ , \tag{4}$$

where N_{walk} means the total number of walks. For a given accuracy criterion of the FRW algorithm ε, the program stops when $err \leq \varepsilon$ is satisfied. If we have n_{thread} threads run on same cores, each thread should be assigned N_{walk}/n_{thread} walks to balance workload. According to (4), we can derive the error of capacitance achieved with the random walks performed on each thread, which corresponds to a FRW procedure set the following termination criterion:

$$\varepsilon' = \sqrt{n_{thread}} \cdot \varepsilon \ . \tag{5}$$

This means we can just set ε' as the accuracy criterion to each thread at the beginning, and then no more synchronization is needed during the FRW procedure.

The multi-thread algorithm with allocated accuracy criterion is illustrated in Fig. 4. The reproducibility is attained because random walks on each thread can be considered as a single-thread run. The critical section exists only when each thread finishes its own FRW procedure. Compare with the method in the above subsection, this approach has better efficiency.

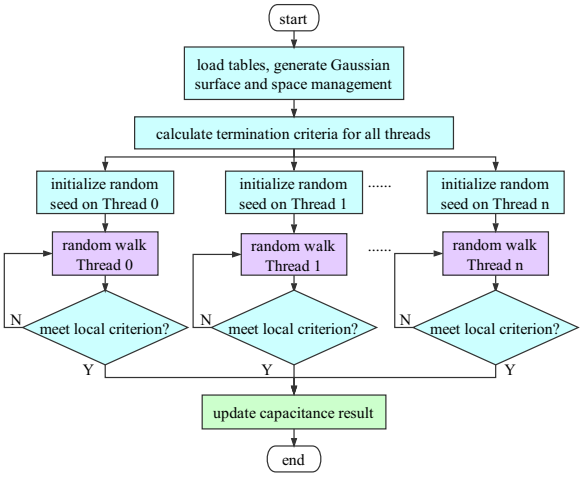

Fig. 4. Flowchart of a multi-thread FRW algorithm with allocated accuracy criteria.

D. Consideration for the Distributed Parallel Version

The two approaches guaranteeing the reproducibility can also be applied to the distributed FRW algorithm. The message passing model is used to realize the synchronization. Fig. 5 shows the flowchart of the distributed FRW algorithm including the synchronization approach, where the dashed line indicates the communication between processes. Different from the multi-thread algorithm, each process sends the intermediate data to process 0 after every M walks and then waits. Process 0 updates the capacitance after receiving the data from all the other processes, and then checks the termination criterion. We set the random seeds dependent on process ID. They are fixed and different for each process.

Fig. 6 shows the flowchart of the distributed FRW algorithm with allocated accuracy criteria. It is very similar to

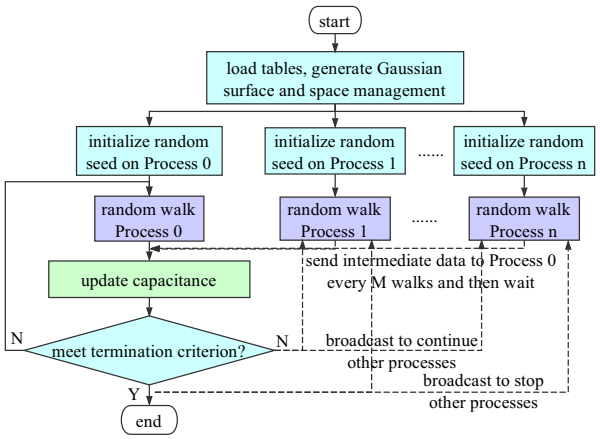

Fig. 5. Flowchart of a distributed parallel FRW algorithm including synchronization with barrier.

the multi-thread algorithm in Fig. 4, except that process 0 receives messages from all other processes and calculates the capacitance. Actually, there is another difference which should be mentioned. For a distributed computing platform, the performance of each computing core may vary. In this case the work or criterion allocation scheme in last subsection becomes inadequate. If a core (or a computing node) runs fast, we should allocate more work or a smaller accuracy criterion to the process running on it. Suppose we have a testing program run on all the nodes, for measuring their runtimes on each core. Now, we run the distributed FRW algorithm with n_{proc} processes, and the j-th process resides in the core on which the testing program's runtime is T_j. Then, to balance workload and achieve high efficiency we assign $N_{walk}^{(j)}$ random walks to process j:

$$N_{walk}^{(j)} = \frac{\frac{1}{T_j}}{\sum_{i=1}^{n_{proc}} \frac{1}{T_i}} \cdot N_{walk} \ . \tag{6}$$

According to (4), this is equivalent to setting the termination criterion ε_j' for process j.

$$\varepsilon_j' = \sqrt{\frac{\sum_{i=1}^{n_{proc}} \frac{1}{T_i}}{\frac{1}{T_j}}} \cdot \varepsilon \ , \tag{7}$$

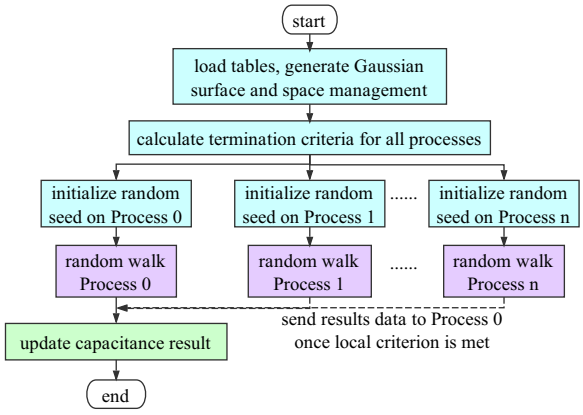

Fig. 6. Flowchart of a distributed parallel FRW algorithm with allocated accuracy criteria.

where ε is the target accuracy criterion. This approach achieves the best workload balance. As long as the set of cores do not vary, the results are reproducible in the subsequent runs.

IV. Implementing the "Jump Start" Feature

The variance of function $f(x)$ in (3) equals approximately:

$$var(f(x)) \approx \frac{\sum_{i=1}^{n}[f(x_i)-\bar{I}]^2}{n-1} = \frac{\sum_{i=1}^{n} f(x_i)^2 - \left[\sum_{i=1}^{n} f(x_i)\right]^2/n}{n-1}. \quad (8)$$

In the context of FRW algorithm, n corresponds to the number of walks N_{walk} and $\bar{I}$ means the calculated capacitance after performing N_{walk} walks. With (3) and (8), one can estimate the 1-σ errors for all capacitances and let the program terminate after meeting a specified accuracy criterion.

The idea of "jump start" run is to recycle a previous run of FRW algorithm so as to save the runtime for subsequent runs for the same case. This is possible if we can pass some information of the previous run, basically N_{walk}, the sums and square sums of weight values (for each conductor), and the states of random number generators, to the next run. Then, they are combined with those from performing more walks to obtain new capacitances and 1-σ errors with (3) and (8). This can be accomplished by storing these intermediate data from FRW based simulation to a hard-disk file. We set an input argument to the FRW solver switching "jump start" on or off. If this feature is on, it will read the hard-disk file to see if the intermediate data for the current test case is available. If that's true, the current accuracy criterion will be compared with the 1-σ error derived from the stored data. No more FRW walks is needed if the criterion has already met. Otherwise, more walks are performed, and the intermediate data is reused. This "jump start" technique works whether the accuracy demand or the number of walks is set as the termination criterion. As long as a case is previously simulated with the FRW solver, the performed walks can be reused for the subsequent run of the same case with same master conductor. With the "jump start" feature, the runtime of FRW solver can be reduced without loss of accuracy and reproducibility. Obviously, this technique adapts to parallel FRW algorithms providing that multiple threads/processes are able to access a common file.

V. Experimental Results

The parallel FRW algorithms are all implemented in C++ coded with Pthreads and Message Passing Interface (MPI). We first validate the reproducibility of the proposed algorithms and compare the reproduction rate (RR) and runtime between different approaches. Then, we validate the "jump start" feature to show its benefit. Various structures from VLSI circuit and touchscreen design have been tested on two Linux servers and a computer cluster. The Linux servers are equipped with Intel Xeon E5-2650 2.3GHz CPU with 24 cores and E5-2630 2.4GHz with 32 cores, respectively. The cluster has homogeneous computing nodes. Each node includes 12-core Intel Xeon X5670 CPU at 2.93GHz and nodes are connected with the infiniband QDR network. Due to page limit, below we present the experimental results on two typical test cases, and the runtime is measured on the Linux server with Xeon E5-2650 CPU or the cluster.

Case 1 contains three parallel wires, whose length is 2000nm and wire spacing is 70nm. **Case 2** is a touchscreen structure from [3] with 11 conductor blocks in two metal layers. Three dielectric layers have relative permittivities of 4.0, 3.5, 7.0, and their heights are all 100nm.

To verify the reproducibility, four algorithms are tested: the **original** approach is the baseline multi-thread algorithm in [6] and the random seeds therein are fixed; the **wait w/o extra walk** and **sync. w/ barrier** are the approaches presented in Section III.B; the **criterion allocation** approach is that proposed in Section III.C. The results of these multi-thread algorithms are listed in Table I, with varied number of threads and accuracy criterion. ϵ in the table stands for the relative criterion, i.e. the ratio of accuracy criterion ε to self capacitance. "Cap." is the computed self capacitance. "RR" is defined as the ratio of the number of times that the first run's result appears again in the following runs to the total number of runs. The results having the same 8 significant digits is considered the same. Here we run each simulation for 100 times. T_{FRW} is the runtime of FRW procedure and T_{total} is the overall runtime including loading tables and the generation of Gaussian surface, etc. "Speedup" is the parallel speedup for the FRW procedure. From the results we see that both the approach including synchronization and the approach with allocated criteria guarantee the reproducibility (with 100% RR).

TABLE I
Computational Results of Four Multi-Thread FRW Solvers. (Capacitance Unit: 10^{-15} F, Time Unit: second)

Settings	Methods	Cap.	RR	T_{FRW}	T_{total}	Speedup
Case 1, n_{thread}=2, ϵ=0.005	original [6]	0.365	0%	1.24	1.42	1.73
	wait w/o extra walk	0.364	32%	1.26	1.44	1.71
	sync. w/ barrier	0.364	100%	1.30	1.48	1.66
	criterion allocation	0.366	100%	1.27	1.45	1.70
Case 1, n_{thread}=16, ϵ=0.005	original [6]	0.364	0%	0.22	0.40	6.15
	wait w/o extra walk	0.364	15%	0.24	0.42	5.86
	sync. w/ barrier	0.364	100%	0.30	0.48	4.73
	criterion allocation	0.366	100%	0.24	0.42	5.86
Case 2, n_{thread}=2, ϵ=0.005	original [6]	78704	0%	270.5	271.0	1.96
	wait w/o extra walk	78712	28%	286.6	287.1	1.85
	sync. w/ barrier	78724	100%	319.3	319.8	1.66
	criterion allocation	78729	100%	274.7	275.2	1.93
Case 2, n_{thread}=16, ϵ=0.005	original [6]	78710	0%	33.7	34.2	15.7
	wait w/o extra walk	78719	13%	51.7	52.2	10.25
	sync. w/ barrier	78728	100%	60.6	61.1	8.74
	criterion allocation	78746	100%	34.0	34.5	15.6

And, the latter shows up to **77%** more parallel speedup than the former (15.6X vs. 8.74X for the last simulation). Compared with the original multi-thread algorithm, the proposed approach sacrifices less than 5% on the efficiency (the worst case is 6.15X speedup decreasing to 5.86X speedup).

Similar experiments have conducted on the cluster, and the results of distributed parallel FRW algorithms are listed in Table II. The results verify the reproducibility of proposed techniques again. And, larger efficiency benefit of the approach with allocated criteria over the approach including synchronization is revealed. It brings **4.8X** more parallel speedup for the simulation with 120 processes (113.6X vs. 23.4X). We have also compared the results for same case obtained on different Linux servers and the cluster, and validated that the reproducibility of proposed approach is not restricted by same machine. As long as the number of threads equals the number of processes, the multi-thread program is able to produce the same results as the distributed parallel program.

TABLE II
COMPUTATIONAL RESULTS OF TWO MULTI-PROCESS FRW APPROACHES. (CAPACITANCE UNIT: 10^{-12} F, TIME UNIT: SECOND)

Settings	Methods	Cap.	RR	T_{FRW}	T_{total}	Speedup
Case 2, n_{proc}=60, ϵ=0.001	sync. w/ barrier	78.73	100%	676	677	19.1
	criterion allocation	78.75	100%	221	222	58.1
Case 2, n_{proc}=120, ϵ=0.001	sync. w/ barrier	78.72	100%	551	552	23.4
	criterion allocation	78.73	100%	113	114	113.6

The experimental results for validating the "jump start" feature are listed in Table III. For each case we set two relative accuracy criterion ϵ_1 and ϵ_2, and ϵ_2 is stricter than ϵ_1. $Time_1$ is the runtime with ϵ_1 inputted, and $Time_2$(JS) denotes the runtime for achieving ϵ_2 while enabling the "jump start". $Time_2$(w/o JS) denotes the simulation time to achieve ϵ_2 directly (without enabling "jump start"). From the table we see that the "jump start" technique can remarkably reduce the simulation time (e.g., from 51.6 seconds to 18.5 seconds).

TABLE III
COMPUTATIONAL RESULTS DEMONSTRATING THE BENEFIT OF THE "JUMP START" RUN. (TIME UNIT: SECOND)

Settings	ϵ_1	ϵ_2	$Time_1$	$Time_2$(JS)	$Time_2$(w/o JS)
Case 1, n_{thread}=16	0.005	0.004	0.40	0.31	0.56
	0.005	0.002	0.40	1.32	1.56
Case 2, n_{thread}=16	0.010	0.005	8.4	25.9	34.2
	0.005	0.004	34.2	18.5	51.6

VI. CONCLUSIONS

In practice, parallel FRW algorithms suffer from a reproducibility issue and the long runtime for some scenarios requesting high accuracy. In this paper, we propose a unified technique with allocated accuracy criteria for both multi-thread and distributed parallel algorithms, so as to guarantee the reproducibility of capacitance simulation. The technique has negligible overhead compared with the existing parallel FRW algorithms, and can bring 4.8X more parallel speedup than a synchronization based technique for reproducibility. A "jump start" feature is also implemented to improve the reusability of FRW based simulation. It remarkably saves the runtime for consecutive runs of a same case with varied accuracy criteria.

VII. ACKNOWLEDGEMENT

This work was supported by National Natural Science Foundation of China (No. 61422402, 61872206). Part of this work was completed on the "Explorer 100" cluster system of Beijing National Research Center for Information Science and Technology.

REFERENCES

[1] W. Yu and X. Wang, *Advanced Field-Solver Techniques for RC Extraction of Integrated Circuits*, Springer Inc., Apr. 2014
[2] A. N. Bhoj, R. V. Joshi, and N. K. Jha, "3-D-TCAD-based parasitic capacitance extraction for emerging multigate devices and circuits," *IEEE Trans. VLSI*, 21(11): 2094-2105, 2013.
[3] Z. Xu, W. Yu, C. Zhang, et al., "A parallel random walk solver for the capacitance calculation problem in touchscreen design," in *Proc. GLSVLSI*, May 2016, pp. 1661-1666.
[4] T. Dillinger, "Field-solver parasitic extraction goes mainstream," 2017, https://www.semiwiki.com/forum/content/7154-field-solver-parasitic-extraction-goes-mainstream.html
[5] Y. Le Coz and R. B. Iverson, "A stochastic algorithm for high speed capacitance extraction in integrated circuits," *Solid-State Electronics*, 35(7): 1005-1012, 1992.
[6] W. Yu, H. Zhuang, C. Zhang, et al., "RWCap: A floating random walk solver for 3-D capacitance extraction of VLSI interconnects," *IEEE Trans. CAD*, 32(3): 353-366, 2013.
[7] N. D. Arora, S. Worley, and D. R. Ganpule, "FieldRC, a GPU accelerated interconnect RC parasitic extractor for full-chip designs," in *Proc. IEEE EDSSC*, 2015, pp. 459-462.
[8] K. Zhai, W. Yu, and H. Zhuang, "GPU-friendly floating random walk algorithm for capacitance extraction of VLSI interconnects," in *Proc. DATE*, Mar. 2013, pp. 1661-1666.
[9] M. Song, Z. Xu, W. Xue, et al., "A distributed parallel random walk algorithm for large-scale capacitance extraction and simulation," in *Proc. GLSVLSI*, May 2018, pp. 189-194.
[10] C. Zhang, W. Yu, Q. Wang, et al., "Fast random walk based capacitance extraction for the 3-D IC structures with cylindrical inter-tier-vias," *IEEE Trans. CAD*, 34(12): 1977-1990, 2015.
[11] Z. Xu, C. Zhang, and W. Yu, "Floating random walk based capacitance extraction for general non-Manhattan conductor structures," *IEEE Trans. CAD*, 36(1): 120-133, 2017.
[12] W. Yu, Z. Xu, B. Li, et al., "Floating random walk based capacitance simulation considering general floating metals," *IEEE Trans. CAD*, 37(8): 1711-1715, 2018
[13] P. Maffezzoni, Z. Zhang, S. Levantino, et al., "Variation-aware modeling of integrated capacitors based on floating random walk extraction," *IEEE Trans. CAD*, 37(10): 2180-2184, 2018.
[14] N. Revol and P. Theveny, "Numerical reproducibility and parallel computations: Issues for interval algorithms," *IEEE Trans. Computers*, 63(8): 1915-1924, 2014.
[15] S. Collange, D. Defour, S. Graillat, et al., "Numerical reproducibility for the parallel reduction on multi- and many-core architectures," *Parallel Computing*, 49: 83-97, 2015.
[16] M. Taufer, O. Padron, P. Saponaro, et al., "Improving numerical reproducibility and stability in large-scale numerical simulations on GPUs," in *Proc. IEEE IPDPS*, 2010, pp. 1-9.
[17] M. Mascagni, "Reproducibility in stochastic simulation," *the Workshop on Numerical Reproducibility at Exascale (NRE'2016)*, SC'2016, Salt Lake City, UT, Nov. 2016.
[18] Peter S. Pacheco, *An Introduction to Parallel Programming*, Elsevier Inc., 2011.
[19] M. Matsumoto and T. Nishimura, "Mersenne twister: A 623-dimensionally equidistributed unifrom pseudo-random number generator," *ACM Trans. Modeling and Computer Simulation*, 8(1): 330, 1998.