A Distributed Parallel Random Walk Algorithm for Large-Scale Capacitance Extraction and Simulation

Mingye Song, Zhezhao Xu, Wei Xue, Wenjian Yu
Department of Computer Science and Technology, BNRist
Tsinghua University, Beijing, China
songmy16@mails.tsinghua.edu.cn,zhezhaoxu@gmail.com,xuewei@tsinghua.edu.cn,yu-wj@tsinghua.edu.cn

ABSTRACT

Due to the advantages on scalability and reliability, the floating random walk (FRW) algorithm has been widely adopted for calculating the capacitances among three-dimensional (3-D) conductors. This is evidenced by the industrial practice of interconnect capacitance extraction during the design of high-performance very large-scale integrated (VLSI) circuits. In this work, the FRW algorithm is enhanced through the distributed parallel computing. With an efficient and adaptive task allocation scheme, the communication among different computer nodes is largely reduced. A distributed algorithm for accelerating the space management is also proposed. They have been implemented with Message Passing Interface (MPI) and applied to the high-precision capacitance simulation for touchscreen design and the interconnect capacitance extraction of VLSI circuits. Experiments on a computer cluster show that the proposed techniques achieve up to 114X speedup while using 120 cores, and build up the space management structure for a VLSI case including two million conductor blocks in just 22 seconds (37X parallel speedup on 60 cores).

ACM Reference Format:
Mingye Song, Zhezhao Xu, Wei Xue, Wenjian Yu. 2018. A Distributed Parallel Random Walk Algorithm for Large-Scale Capacitance Extraction and Simulation. In *Proceedings of Great Lakes Symposium on VLSI 2018 (GLSVLSI'18).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3194554.3194568

1 INTRODUCTION

Capacitance extraction and simulation is a fundamental topic of computer-aided design (CAD) [4, 11]. For accurate validation of the delay, signal integrity, and other performance metrics, the interconnect capacitances should be accurately extracted while designing nanometer integrated circuits. Another relevant scenario is during the design of touchscreen. For validating its functionality, simulating the capacitances of the touch sensor structures becomes an important and frequent task. The geometry complexity in touchscreen design and high-accuracy demand for coupling capacitance pose a challenge to the existing capacitance simulation techniques [9].

GLSVLSI'18, May 23–25, 2018, Chicago, IL, USA

ACM ISBN 978-1-4503-5724-1/18/05…$15.00
https://doi.org/10.1145/3194554.3194568

High-precision capacitance simulation needs three-dimensional (3-D) field-solver methods. They can be classified into the following categories [11]: domain descretization method, including the finite difference methods (FDM) and the finite element methods (FEM), boundary element method, and the floating random work (FRW) method [1, 3, 7, 12]. The FRW method is more stable in accuracy and scalable to very large case, due to its nature of a discretization-free method. And as it is based on the Monte Carlo method, the FRW method has better parallelism, and becomes favorable nowadays due to the population of parallel computing infrastructures. A lot of work has been devoted to the FRW method to improve its efficiency and extend its ability [3, 9, 10, 13–15]. Most of them only consider the single-thread or multi-thread implementation on a single computer.

To tackle very large computing task, the parallel computing using GPU or multiple computers should be considered [1, 8, 9, 13]. In [1, 13], GPU accelerated FRW algorithms were proposed showing several tens times speedup over the FRW algorithm with serial-computing. Although the parallel computing on GPU is generally more energy-efficient, implementing a GPU-friendly FRW algorithm requires much algorithmic change, and thus induces much labor on software development and maintenance. And, due to the workload divergence among different walk paths, parallelizing FRW on GPU is not easy, or can hardly achieve high parallel speedup [13]. In contrast, the distributed parallel FRW algorithm on computer cluster or the *cloud* is more feasible. This has been evidenced by industrial practice for multi-net or full-chip extraction tasks. Its basic idea is to assign subsets of nets to each machine and then collect their results on a host machine [8]. This method with coarse-grained workload distribution can cause large workload unbalance, and is not suitable for the extraction of a single net. In [9], a distributed parallel FRW algorithm was proposed to speed up the single-net computation for obtaining accurate coupling capacitances in the touchscreen design. It distributes the random walk procedure to the computers in a cluster, and achieves 67X speedup at most using 120 CPU cores. This method has two major drawbacks: 1) There is still much inter-machine communication which leads to not high enough parallel speedup. 2) It does not distribute the construction of the space management structure among the machines, which makes it inefficient while handling cases with a large number of conductors.

In this work, we focus on fine-grained workload distribution during the FRW based capacitance simulation, which results in an efficient parallel FRW algorithm on computer cluster. We propose an adaptive task allocation scheme and a distributed construction algorithm for the grid-based spatial structure, to minimize the communication cost. They accelerate the random walk procedure and

the space management step respectively, without loss of accuracy. With this distributed parallel FRW algorithm, we have tested structures from touchscreen design and VLSI design. The experiments on a cluster show that the parallel speedup is up to 4.9X larger than that of the algorithm in [9]. And, with 60 cores the space management structure can be built in 22 seconds for a VLSI case including two million conductor blocks. It corresponds to a 37X speedup on the space management construction step.

2 PRELIMINARIES

The FRW method for capacitance extraction is based on the integral formula for electric potential [7, 12]:

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

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 enclosed by S is called transition domain, which includes $\boldsymbol{r}$. $P(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ can be regarded as a probability density function (PDF). With the Monte Carlo method, $\phi(\boldsymbol{r})$ can be estimated as the statistical 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, the charge of conductor i is:

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

where $F(\boldsymbol{r})$ is the dielectric permittivity at point $\boldsymbol{r}$, $\tilde{P}(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ is the PDF for sampling on $S^{(1)}$ which may be different from $P(\boldsymbol{r}, \boldsymbol{r}^{(1)})$, and $\omega(\boldsymbol{r}, \boldsymbol{r}^{(1)})$ is the weight value [7, 12]. Thus, Q_i can be estimated as the statistical 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 decribed 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 statistical mean of the weight values for the walks terminating at conductor j approximates the capacitance C_{ij} between conductors i and j.

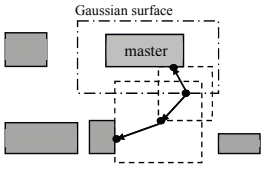


Figure 1: Two examples of random walk in the FRW method for capacitance extraction (a 2-D top view).

The cubic transition domain is widely adopted because it fits well the Manhattan-shaped interconnects in VLSI circuit. This means larger 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.

Another key is the space management techniques [14], which are useful for quickly finding the nearest conductor for constructing the transition cube, especially for simulating a case including thousands of conductor blocks. A grid-Octree hybrid spatial structure was proposed in [14], which enables better efficiency of conductor inquiry than existing grid or Octree based structures. In [15], a multi-thread parallel algorithm was proposed for constructing the Octree structure. On a machine with 8 cores it achieves 4.5X parallel speedup.

The flowchart of the distributed parallel FRW algorithm in [9] is shown in Fig. 2. Its focus is on accelerating the random walk process without paying attention to the construction of the space management structure. However, a block-level or chip-level VLSI structure can involve more than a million conductor blocks. Constructing the spatial structure for such large case can be much more costly than running the FRW procedure to extract a net's capacitances.

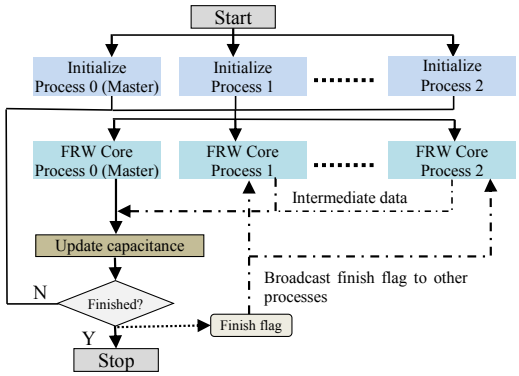


Figure 2: The Flowchart of the distributed parallel FRW algorithm in [9].

3 EFFICIENT DISTRIBUTED PARALLEL FRW ALGORITHM

3.1 Basic Idea

To speed up the FRW algorithm in dealing with both VLSI and touchscreen design problems, many techniques have been adopted, such as space management techniques [11, 15], dielectric pre-characterization method [9], etc. However, for high-accuracy and/or large-scale simulation, the cost of FRW algorithm is still expensive.

Distributed parallel computing is a crucial technique to solve the large and high-accuracy capacitance calculation problems. For large circuit with many conductor blocks, the construction of space management is slow relatively compared with the random walk procedure. Notice that more conductors usually mean faster convergence of the random walk procedure. For touchscreen design, the random walk procedure needs more walks to achieve the demanded high accuracy, so it becomes the bottleneck. In these scenarios, we need a distributed parallel FRW algorithm with fine-grained workload distribution, where efficiently parallelizing both the space management construction and the random walk procedure becomes crucial.

3.2 Distributed Parallel Random Walk Procedure

The random walk procedure of the FRW algorithm is suitable for parallelization naturally, since each random walk is independent to each other. In [12], the parallel FRW algorithm is implemented with

pthread APIs. However, the speedup is limited by the condition of a single machine and the number of CPU cores. To achieve higher speedup, in [9], a parallel FRW algorithm was implemented on a Cluster Environment with MPI (see Fig. 2). First, the algorithm initializes each process including parsing input file and loading GFT/WVT tables. After that, each process starts to execute the random walk procedure independently and all processes send intermediate data to the master process every m walks (typically $m = 1000$). Process 0 serves as master process and is responsible for collecting the intermediate data from other processes. Once the termination criteria is satisfied, the master process broadcasts the finish flag to all other processes to terminate the whole computation. Because all processes except master process need to send intermediate result to the master process frequently, the communication overhead is large. If the simulation needs a plenty of random walks, the parallel speedup of this scheme will become disappointing. For tackling the problem, we figure out a new parallel random walk algorithm to reduce the communication overhead.

From [11], we can derive that the estimated error of capacitance err is inversely propositional to the square root of number of walks n_{walk} as follows,

$$err \propto \frac{1}{\sqrt{n_{walk}}}, \tag{3}$$

where n_{walk} means the total number of walks. For a given termination criterion of the FRW algorithm ϵ, the program stops when $err \leq \epsilon$ is satisfied. If we have $m_{process}$ processes, each process should be assigned $\frac{n_{walk}}{m_{process}}$ walks to balance workload. According to Eq. (3), we can derive the error of capacitance achieved with the random walk performed on each process, which corresponds to a FRW procedure set the following termination criterion:

$$\epsilon' = \sqrt{m_{process}} \cdot \epsilon. \tag{4}$$

This means we just set the termination criterion in Eq. (4) to each process at the beginning, and then no more communication among processes is needed during the FRW procedure.

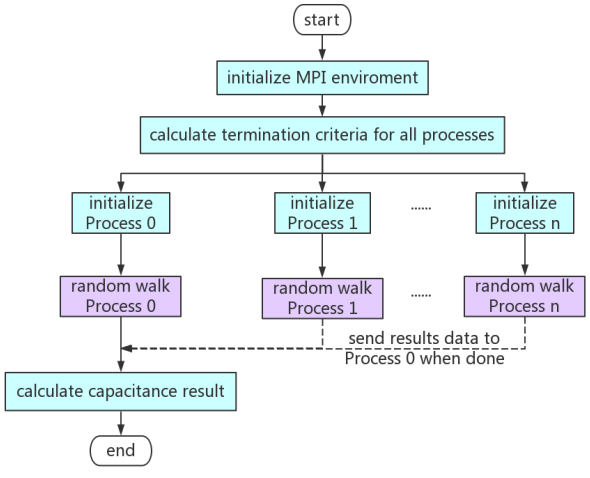


Figure 3: Flowchart of the proposed distributed parallel FRW procedure.

With this idea, we propose a distributed parallel FRW procedure which is shown in Fig. 3. It avoids the communication during the FRW procedure. After every process finishes their work, they send result data to the master process. On the master process the final capacitance result is calculated. Obviously, this method has little communication overhead.

The proposed walk assignment strategy attempts to divide the whole task into independent equal small tasks. However, its basis, Eq. (3) holds approximately. Therefore, if all processes terminates after reaching the condition in (4) sometimes the final capacitance cannot satisfy the error criterion ϵ. To overcome this drawback, a trick is setting the distributed termination criterion a little stricter than the result of Eq. (4). This is empirically proven an effective way to guarantee the accuracy of final result.

The above treatment is based on an assumption that the performance of different machines in the cluster is the same, but it is not always true. When it comes to the fact that the performance of machines are different, an adaptive allocation scheme should be conducted. Before we start our task, we run a quick and standard random walk program to measure actual performance of each machine. Assuming that the runtime for the testing procedure on computing node j is t_j and we have m computing nodes each corresponding to a MPI process, we should allocate

$$n_{walk}^{(j)} = \frac{\frac{1}{t_j}}{\sum_{i=1}^{m} \frac{1}{t_i}} \cdot n_{walk} \tag{5}$$

random walks to process j. Based on Eq. (3), we then should set the termination criterion ϵ_j' for process j:

$$\epsilon_j' = \sqrt{\frac{\sum_{i=1}^{m} \frac{1}{t_i}}{\frac{1}{t_j}}} \cdot \epsilon. \tag{6}$$

This will achieve the best balance of workload, and results in the shortest runtime of the whole distributed parallel computing.

3.3 Distributed Parallel Space Management

In order to improve the procedure of building space management, parallelization is a key technique. There are several spatial data structures, such as uniform grid and Octree. Octree is applied in [12, 15], while for parallel application, sending partial information of Octree between processes and rebuilding the whole data structure are more complex than those of uniform grid. And, as shown in [11], the grid with candidate list has similar performance to the Octree structure, and they both are slightly inferior to the grid-Octree hybrid spatial structure. As the grid structure is more suitable for parallel computing, we choose the grid-based approach to develop the distributed parallel FRW algorithm.

Fig. 4 shows the grid with candidate list structure. Every grid cell contains a list of candidate list that consists of the conductor blocks which can be the nearest to the points in the grid cell. Generating these candidate lists are the major work of construction of this grid spatial structure. During construction, conductor blocks nearby should be inserted into the candidate list according to the judgment of domination relationship. Because the candidate lists of different grid cells are independent to each other, the grid cells can be assigned to different processes or threads without conflict. After

all processes finish the candidate list generation for assigned cells, the information sent among processes simply includes conductor IDs and grid cell IDs and they are enough to rebuild the whole spatial structure for the further computation. For the cell with too long candidate list, it can be divided as a second-level grid structure.

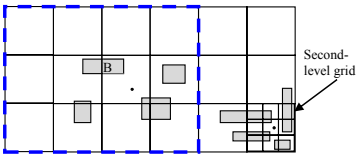


Figure 4: A two-level grid structure. The neighbor cells of conductor B are outlined with the *blue dashed* line [11].

The flowchart of the proposed distributed parallel space management is shown in Fig. 5. Firstly, we split the whole grid cells into cell sets, then assign them to different processes separately and subsequently calculate the candidate list for each cell. Finally, we can gather all information together to build the complete spatial data structure.

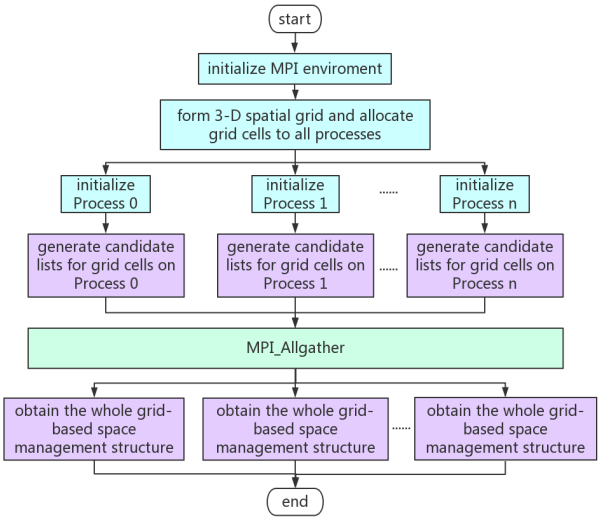
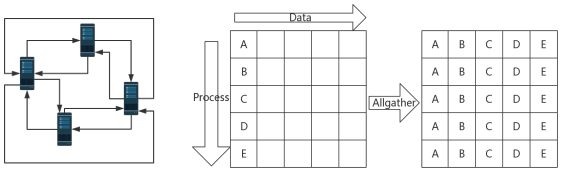


Figure 5: Flowchart of the proposed algorithm on parallel space management construction.

It should be pointed out that the communication cost between processes is not negligible. A well-designed communication scheme benefit the speedup and vice versa. There are two issues. One is how to communicate between processes and the other is what to send.

For the first issue, it is common to use send/receive pairs, which means to use MPI_ISend and MPI_IRecv functions in MPI. MPI_ISend and MPI_IRecv are similar to MPI_Send and MPI_Recv, and the difference between them is that the former functions are non-blocking [6]. If we use MPI_ISend and MPI_IRecv functions, all the $m_{process}$ processes broadcast the message to all the other processes and they send $m_{process}(m_{process} - 1)$ messages totally (see Fig. 6(a)). This makes too many messages occupy the channel. And, synchronization delay becomes prominent when the number of processes is large. So, it is time-consuming and not efficient. We propose to use collective communication in MPI, which is MPI_Allgather function. It gathers data from all tasks and then distributes the combined data to all tasks (see Fig. 6(b)) [2, 5]. Compared with simple send/receive pairs, collective communication performs better. The programmer should allocate sufficient size of receiving buffer to avoid overflowing. The required size of receiving buffer depends on the actual case and how the space management construction task is allocated. Too large and too small receiving buffer both cause problem. For this reason, we cannot allocate a constant-size receiving buffer prior to the computation and the buffer should be dynamically allocated. Actually, for a given case, no matter how many processes are set, the sum of lengths of candidate lists is constant if the grid partition is set. So, the size of receiving buffer does not increase as the number of processes increases. We can calculate its upper bound according to the maximum length of messages. Before each process sends the message of the candidate lists, we let them firstly send the length of messages so that we can find out the maximum length. Therefore, we can set and allocate a sufficiently large receiving buffer. As shown in Fig. 6(b), the receiving buffer can be treated as a two-dimensional array where each row stores all the candidate lists from one single process. Because MPI_Allgather does not return the length of message, we should also set a finish flag to mark the end of each receiving message. This communication scheme is efficient both on storage and latency.



(a) Send/receive (b) Allgather

Figure 6: Two communication mechanisms.

For the second issue, a trivial way is to send a type of structure that contains conductor block ID and grid cell ID, which are both represented as integer. The rebuilding step is inserting the conductor block into the corresponding cell according to the grid cell ID. However, it is obvious that some information is redundant because blocks in the same grid cell share the same grid cell ID. Inspired by sparse matrix storage format, we design a compressed grid cell format that we combine all conductor block IDs in the same cell together along with grid cell ID and use a division flag to distinguish blocks from each other. Each process is responsible to a cell set which includes the grid cells requesting the generation of candidate list. Suppose $size_{max}$ is the maximum total number of candidate conductors in a cell set, and $m_{process}$ is the number of processes. The trivial way needs at most

$$Buffer\_size_1 = 8m_{process} \cdot size_{max} \tag{7}$$

bytes storage for the receiving buffer. In contrast, the compressed one only needs at most

$$Buffer\_size_2 = 4m_{process} \cdot size_{max} + 4n_{cell} \tag{8}$$

bytes for the receiving buffer, where n_{cell} is the number of grid cells. Notice $m_{process} \cdot size_{max}$ is multiplies of the number of all conductor blocks in the problem, and the number of all conductor blocks is much larger than n_{cell} (e.g., $n_{cell} = 2048$ in our experiments). Therefore, the compressed message format can almost reduce the size of sent messages and the buffer size to the half.

4 EXPERIMENTAL RESULTS

The proposed FRW methods are all implemented in C++ with MPI. They are tested with structures from both touchscreen design and VLSI design. If not explicitly stated otherwise, all experiments are carried out on a computer cluster where each node includes 12-core Intel Xeon X5670 CPU at 2.93GHz and nodes are connected with the infiniband QDR network. Half of the computing nodes are equipped with 32GB memory and half 48GB.

Five structures are tested. The first three are from real touchscreen design [9], and the last two are from VLSI design. Their details are listed as follows.

Case 1: This case contains 1423 conductor blocks in two metal layers. The dielectric layers have relative permittivity of 4.0, 3.2, 4.0, 1.0. The metal heights in the two layers are 70nm and 220nm.

Case 2: This case is a small structure with 11 conductor blocks in two metal layers. The dielectric layers have relative permittivity of 4.0, 3.5, 7.0. The heights of the two metal layers are both 100nm.

Case 3: This case contains 808 conductor blocks in four metal layers. The dielectric layers have relative permittivity of 3.9, 6.5, 3.5, 6.5, 4.2, 3.2, 4.0, 1.0. The heights of the four metal layers are 340nm, 220nm, 400nm and 70nm, respectively.

Case 4: This case contains 484,441 conductor blocks forming 12,149 nets. Different from the former cases with non-Manhattan shape generally, the latter two cases are mostly Manhattan shape.

Case 5: This case contains 2,302,995 conductor blocks forming 1,163,751 nets.

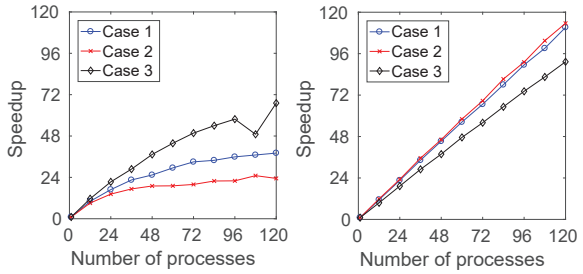
4.1 Test Cases from Touchscreeen Design

The first three cases are tested with the distributed parallel FRW algorithm. Because these cases do not include many conductors, the multi-process parallel space management construction is not applied. We have also implemented the algorithm in [9] for comparison. To obtain accurate coupling capacitances, the FRW algorithms are run with 0.1% 1-σ error criterion on the self-capacitance. The parallel runtime of the algorithm in [9] and the proposed algorithm are listed in Table 1, with different number of processes.

Because of the communication and synchronization costs among different processes, the speedup of the algorithm in [9] is far from the ideal situation. The best speedup is only 67.1X with 120 processes and the worst is 23.4X. In contrast, the proposed algorithm with task assignment based on estimated error reduces the communication cost significantly. It achieves up to **114X** speedup for Case 2 with 120 processes, which is about **4.9X** larger that of the algorithm in [9]. For Case 2, the FRW algorithms consumes as many as 5,000,000 random walks, ten times larger than that for Case 3. Because the proposed method minimizes the communication during the random walk procedure, it performs best for Case 2.

Table 1: The parallel runtime of the algorithm in [9] and the proposed FRW algorithm.

$m_{process}$	Case 1		Case 2		Case 3	
	[9](s)	ours(s)	[9](s)	ours(s)	[9](s)	ours(s)
1	3661	3668	12904	12838	7452	5937
12	363	321	1425	1074	644	621
24	217	162	905	554	349	307
36	163	107	747	362	259	205
48	144	81	678	278	200	157
60	124	65	676	221	170	125
72	111	55	647	187	150	106
84	108	47	590	158	138	91
96	102	41	588	141	129	80
108	99	37	516	124	152	72
120	96	33	551	113	111	65



(a) The method in [9] (b) The proposed method

Figure 7: The parallel speedup vs. number of processes for Cases 1-3.

The comparison between two algorithms is also illustrated in Fig. 7. It is apparent that the proposed method achieves better performance. Moreover, the trick guaranteeing that the given termination criterion ϵ is satisfied involves a relatively small runtime cost, but ensures the accuracy.

An experiment is also carried out to evaluate the proposed adaptive workload distribution scheme. We test the parallel FRW algorithms with the uniformly distributed criterion Eq. (4) and the nonuniformly distributed criterion Eq. (6) respectively, on a cluster with two machines. One machine is equipped with Intel Xeon E5-2630 2.3GHz CPU with 24 cores, and the other is with Xeon E5-2630 2.4GHz CPU with 32 cores. The experimental results show that in 32 processes, the parallel FRW algorithm with the nonuniformly distributed criterion further reduces the runtime from 13.32 seconds to 10.32 seconds by **22.5%** compared with the algorithm with uniformly distributed criterion. This validates the effectiveness of the proposed technique.

4.2 Test Cases from VLSI Design

We use Case 4 and 5 to test the distributed parallel grid-based space management construction technique. As we have introduced, the space management construction may dominate the overall runtime in those cases with a large number of conductor blocks and just a

Table 2: The results on space management construction for two large VLSI cases.

$m_{process}$	Case 4		Case 5	
	time(s)	speedup	time(s)	speedup
1	52.12	1.00	824.11	1.00
12	6.55	7.96	76.72	10.74
24	3.90	13.36	42.55	19.37
36	3.15	16.55	29.91	27.55
48	2.58	20.20	23.75	34.70
60	2.27	22.96	22.05	**37.37**

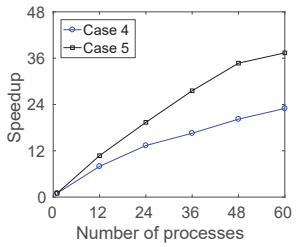


Figure 8: The parallel speedup on space management construction vs. the number of processes.

few of nets for extraction. The experimental results on the VLSI design cases are listed in Table 2.

The parallel speedup of the space management construction is also shown in Fig. 8. For Case 5, with 60 processes the runtime for building the grid with candidate list is reduced from 824.11 seconds in serial computing to 22.05 seconds, which means a **37.4X** speedup. For this larger case with two million conductor blocks the proposed algorithm demonstrates satisfied performance. Notice for Case 4, the construction time has been reduced to just 2 seconds.

The runtime breakdown for exacting a single net with the proposed algorithm is listed in Table 3. It is for Case 5, with 0.5% 1-σ error criterion on self-capacitance. t_{sp} and t_{walk} denotes the runtime for space management construction and the random walk procedure, respectively. Accordingly, $speedup_{sp}$ and $speedup_{walk}$ are the parallel speedup for the both parts. t_{pre} denotes the preparation time in the FRW algorithm that contains all the work done before starting the random walk procedure, including parsing input file, loading GFT/WVT tables and space management construction. With 60 processes, the parallel speedup of space management construction is 37.4X, while that of random walk reaches 38.8X. The total runtime equals to $t_{pre} + t_{walk}$, which is 41.3 seconds. Notice there is about 12 seconds for parsing the input file including two million conductors, which is serially executed.

For Case 5, the maximum number of conductor blocks $size_{max}$ equals to 60,850 in 60 processes. According to Eq. (7) and (8), the trivial message format needs 27.85MB receiving buffer for each process, while the proposed message format only needs 13.94MB. It reveals that the memory overhead is well acceptable.

5 CONCLUSIONS

Parallelization is crucial for capacitance calculation problems when high-accuracy demand and/or a large number of conductors are involved. With an adaptive task allocation scheme and a grid-based distributed space management construction technique, we propose an efficient FRW algorithm suitable for large computer cluster environment. The experiments validate the effectiveness and advantages of the proposed algorithm, with comparison with existing techniques.

Table 3: The runtime breakdown of the proposed algorithm for Case 5.

$m_{process}$	t_{sp}(s)	t_{pre}(s)	t_{walk}(s)	$speedup_{sp}$	$speedup_{walk}$
1	824.11	836.73	189.47	1.00	1.00
12	76.72	90.06	18.59	10.74	10.19
24	42.55	57.11	9.87	19.37	19.20
36	29.91	43.89	6.73	27.55	28.15
48	23.75	38.83	5.13	34.70	36.93
60	22.05	36.41	4.89	37.37	38.75

ACKNOWLEDGMENTS

This work is supported by NSFC (Grant No. 61422402). W. Xue's work is supported by the National Key R&D Program of China (Grant No. 2016YFA0602100) and NSFC (Grant No. 91530323). W. Yu is the corresponding author.

REFERENCES

[1] Narain D Arora, Steven Worley, and Dilip R Ganpule. 2015. FieldRC, a GPU accelerated interconnect RC parasitic extractor for full-chip designs. In *Proc. EDSSC*. 459–462.
[2] Blaise Barney. 2017. Message Passing Interface (MPI), Online tutorials. (2017). https://computing.llnl.gov/tutorials/mpi/.
[3] S. H. Batterywala and M. P. Desai. 2005. Variance reduction in Monte Carlo capacitance extraction. In *Proc. 18th Int. Conf. VLSI Design*. 85–90.
[4] A. N. Bhoj, R. V. Joshi, and N. K. Jha. 2013. 3-D-TCAD-based parasitic capacitance extraction for emerging multigate devices and circuits. *IEEE Trans. VLSI* 21, 11 (2013), 2094–2105.
[5] Victor Eijkhout. 2012. *Introduction to High Performance Scientific Computing*. Lulu.com.
[6] George M. Karniadakis and Robert M. Kirby. 2003. *Parallel scientific computing in C++ and MPI*. Cambridge University Press.
[7] Y. Le Coz and R. B. Iverson. 1992. A stochastic algorithm for high speed capacitance extraction in integrated circuits. *Solid-State Electronics* 35, 7 (1992), 1005–1012.
[8] N Sawhney, S. Batterywala, N. Shenoy, and R. Rudell. 2004. Parallelizing a statistical capacitance extractor. In *Proc. VLSI Design and Test*. 253–267.
[9] Zhezhao Xu, Wenjian Yu, Chao Zhang, Bolong Zhang, Meijuan Lu, and Michael Mascagni. 2016. A parallel random walk solver for the capacitance calculation problem in touchscreen design. In *Proc. GLSVLSI*. 1661–1666.
[10] Z. Xu, C. Zhang, and W. Yu. 2017. Floating random walk based capacitance extraction for general non-Manhattan conductor structures. *IEEE Trans. Computer-Aided Design* 36, 1 (2017), 120–133.
[11] Wenjian Yu and Xiren Wang. 2014. *Advanced Field-Solver Techniques for RC Extraction of Integrated Circuits*. Springer Science & Business.
[12] Wenjian Yu, Hao Zhuang, Chao Zhang, Gang Hu, and Zhi Liu. 2013. RWCap: A Floating Random Walk Solver for 3-D Capacitance Extraction of Very-Large-Scale Integration Interconnects. *IEEE Trans. Computer-Aided Design* 32, 3 (2013), 353–366.
[13] K. Zhai, W. Yu, and H. Zhuang. 2013. GPU-Friendly floating random walk algorithm for capacitance extraction of VLSI interconnects. In *Proc. DATE*. 1661–1666.
[14] C. Zhang and W. Yu. 2013. Efficient space management techniques for large-scale interconnect capacitance extraction with floating random walks. *IEEE Trans. Computer-Aided Design* 32, 10 (2013), 1633–1637.
[15] Chao Zhang and Wenjian Yu. 2014. Efficient techniques for the capacitance extraction of chip-scale VLSI interconnects using floating random walk algorithm. In *Proc. ASP-DAC*. 756–761.