GPU Acceleration in VLSI Back-end Design: Overview and Case Studies (Invited Talk)

Yibo Lin
CECA, CS Department
Peking University
yibolin@pku.edu.cn

ABSTRACT

The semiconductor industry keeps seeking for reducing the design time and efforts in modern integrated circuit implementation, which often incorporates billions of transistors. Among the entire design flow, back-end design involving the physical implementation takes a significant portion of the design time. Recent advances in GPU acceleration bring new opportunities to speedup the design closure. In this tutorial, we review the efforts in literature and current status on accelerating the back-end design automation algorithms. We summarize the challenges in the key design stages such as placement, routing, and timing anaylsis, and provide several case studies on how to enable massive parallelism in practice.

1 INTRODUCTION

The semiconductor industry is encountering two challenges: a) increasingly large designs; b) complicated design constraints and objectives. These challenges lead to more computation efforts required to search for legal solutions that can satisfy all the constraints, eventually causing longer design cycles and slow convergence to optimal objectives, especially in the back-end design flow. Thus, the industry keeps in seeking for reducing design time and effort in modern integerated circuit (IC) implementation incorporating billions of transistors.

Recent advances in GPU bring new opportunities for high-performance design automation. With new GPU architectures like Volta, Turing, and Ampere that integrate ultra-fast tensor cores and new wire-based communication protocol NVLink, NVIDIA reported exponential growth in both the number of floating point operations per second (FLOPS) and the peak memory bandwidth in the past 10 years [1].

However, even with such progress in GPU hardware, which has already achieved competitive performance in accelerating computation tasks in fields like machine learning and scientific computing, obtaining notable benefits from GPU acceleration in the back-end algorithms is still challenging. The main reasons are as follows: a) diverse workloads; b) complicated computation kernels; c) interleaving efficiency bottlenecks. Firstly, VLSI back-end design flow usually contains placement, routing, timing analysis, etc., involving algorithms in combinatorial optimization, graph theory, and greedy heuristics. The algorithms may work on matrices, graphs, and even geometries with both regular and irregular computation patterns, resulting in diverse workloads that are difficult to parallelize. Secondly, The computation kernels of back-end algorithms are usually complicated with all kinds of customized heuristics, quite different from those clean and simple kernels extensively explored in the high-performance computing (HPC) community, like matrix multiplication, single-source shortest path, and page rank. We usually cannot directly apply the techniques developed in HPC to accelerate practical kernels. Thirdly, there is not any single kernel that takes significant (more than 80%) amount of the runtime. On the contrast, we often observe many kernels taking similar runtime portions such that we have to accelerate all of them for notable speedup. This leads to dramatic development overhead and also requires efficient system-level integration to put all kernels together. Therefore, GPU acceleration on back-end design automation still remains to be extensively investigated with the emerging computation power and urgent demands for efficiency.

By incorporating both algorithmic innovation and acceleration techniques, recent studies have demonstrate that GPU acceleration is promising to boost the efficiency of key steps in the back-end flow, including placement, routing, and timing analysis. In this paper, we will review related progress and summarize the major challenges and ideas in each specific design stage.

The rest of the paper is organized as follows. Section 2 details the efforts in GPU accelerated placement; Section 3 reviews the status in GPU accelerated routing; Section 4 illustrates the progress in GPU accelerated timing analysis. We also provide case studies along with related sections to show how GPU acceleration helps to boost the efficiency in these sections. Section 5 concludes the paper.

2 GPU ACCELERATION FOR VLSI PLACEMENT

In this section, we review the efforts on GPU-accelerated VLSI placement. Placement usually consists of three stages: global placement (GP), legalization (LG), and detailed placement (DP). Global placement determines the rough locations of standard cells in the layout with possibly small overlaps. Legalization removes all the overlaps to satsify all the design rules. Detailed placement further refines the placement solution for better quality. In general, GP and DP are more timing-consuming than LG, so current acceleration efforts have been concentrated on these two steps, especially on GP.

2.1 Global Placement

GP typically tries to solve the following mathematical programming,

$$\begin{aligned} \min_{x,y} \quad & WL(x,y), \\ \text{s.t.} \quad & D(x,y) \leq d_t, \end{aligned} \tag{1}$$

where $WL(\cdot)$ denotes the wirelength objective and $D(\cdot)$ denotes the density at any location of the layout.

To solve this problem, two computation kernels must be implemented: wirelength gradient and density map accumulation. Lin et

ICCAD '20, November 2–5, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415765

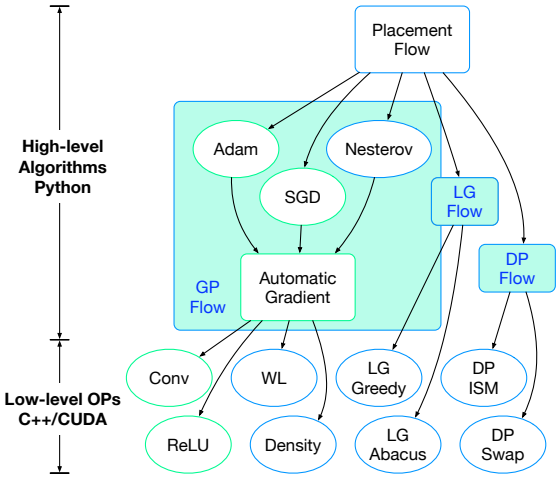

Figure 1: Software architecture of DREAMPlace [10].

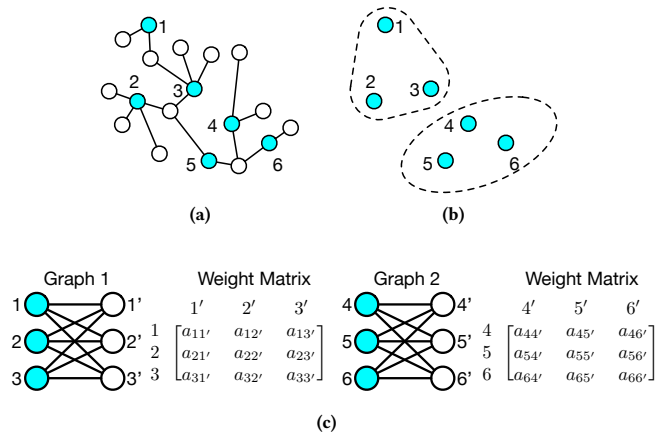

Figure 2: Steps for batch-based independent set matching [13]. (a) Maximal independent set extraction. (b) Independent set partitioning. (c) Batched bipartite matching.

el propose two efficient kernel implementations for wirelength gradient and density map computation on GPU [2]. There are several works exploring GPU-accelerated GP based on different placement algorithms, e.g., [3] based on mPL6 [4] (15× speedup), [5, 6] based on TimerWolf (2−5× speedup), *DREAMPlace* [7] based on eplace/RePlAce [8] (30 − 40× speedup). Most works have been focusing on pure wirelength minimization and recent DREAMPlace [7] has an option to consider routability with feedback from external routers as well. All these efforts can achieve promising speedup with almost no quality degradation. While GP is a step suitable for GPU acceleration, it has not yet been a standard practice as the placement algorithms keep evolving, and the overhead of implementing algorithms on both CPU and GPU is high. To tackle this problem, DREAMPlace [7] introduces deep learning toolkit `PyTorch` [9] to develop placement engines. As Figure 1 shows, these toolkits wrap low-level operators that can run on multiple platforms and decouple the high-level algorithm implementation from the low-level operator acceleration. Thus, developing placement engines upon such toolkits can naturally achieve multi-platform support and high efficiency.

2.2 Detailed Placement

DP is a critical step for incremental optimization. In advanced technology nodes, as the design closure becomes more difficult, DP can be invoked many times for optimization of various objectives. Typical DP algorithms often follow greedy local search procedures, involving combinatorial optimizations and graph algorithms, which are in general hard to parallelize. Meanwhile, DP usually consists of many different algorithms with completely discrepant procedures, so there is almost no generic way for acceleration. Hence, the efforts on accelerating DP are much less than GP and rather fragmented.

Recently, Dhar et al accelerate the dynamic programming kernels in a row-based interleaving algorithm [11], originally from [12]. The key idea is to fill a 3D dynamic programming table in parallel and they demonstrate 7× speedup over 20-thread CPU implementation.

We further present systematic acceleration of DP algorithms, *ABCDPlace* [13], covering widely used DP algorithms, i.e., global swap, independent set matching, and local reordering. We overcome the challenge of lack of parallelism in DP by exploiting batched execution of cells physically or logically far away from each other. For example, in global swap, we can simultaneously compute the costs of swap candidate cells located at different regions; in independent set matching, we extract an independent set of cells from the entire netlist, which is, in other words, logically apart from each other, so that we can solve bipartite assignment independently; in local reordering, we simultaneously permute sequences of cells at different placement rows that do not share common nets. Figure 2 provides a concrete view on making the independent set matching algorithm parallelizable. In the first step, we perform maximal independent set extraction to the entire circuit netlist; this step can be solved with parallel Blelloch's algorithms [14]. In the second step, we partition the independent set to group cells physically closing to each other; this step can be solved with parallel k-means clustering. In the last step, we can solve each bipartite matching independently with parallel auction algorithm [15]. Eventually, more than 15× speedup can be achieved over single-thread CPU on million-size designs without quality degradation, while the speedup from the 20-thread CPU implementation saturates at 2 − 5×. Although we design the algorithms with similar intuitions for acceleration, the actual implementations are completely different, requiring high development efforts; e.g., the lines of source code for DP is much more than that for GP [7]. This is probably one of the reasons why there lack efforts on DP acceleration. So far, all the published works on DP acceleration are still pure wirelength-driven.

3 GPU ACCELERATION FOR VLSI ROUTING

Routing is known as the most time-consuming step in the back-end design flow. Due to its complexity, modern routing is usually divided into global routing and detailed routing at different granularity, while the kernel routing algrithms are mostly based on maze routing.

Exploiting GPU for routing has several challenges: a) lack of parallelism for each net; b) divergence of computation patterns between nets; c) huge random memory access. As most nets are local, the overhead of exploring parallel single-net routing is high due to its lack of parallelism. However, the existence of large nets makes it time-consuming to route with only a single thread. Such heterogeneity of nets also leads to quite different computation patterns at runtime, causing high synchronization overhead. As routing is conducted on large graphs or grids, poor memory locality often brings overhead in the efficiency as well. In the literature, GPU acceleration on both FPGA and ASIC routing has been investigated.

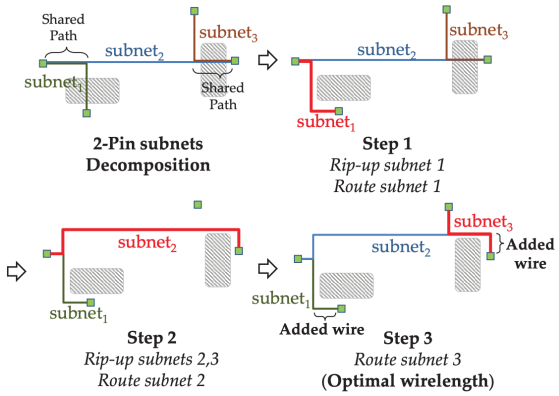

Figure 3: Net decomposition to increase the parallelism [19].

3.1 FPGA Routing

FPGA routing essentially tries to find disjoint paths on a routing resource graph. Shen et al propose the first GPU-accelerated FPGA routing leveraging GPU-friendly single-source shortest path (SSSP) kernels [16, 17]. Instead of using the Dijkstra or A* search algorithm, they adopt the Bellman-Ford algorithm, which is suitable for massive parallelization on GPU. To increase the parallelism, they also limit the search bounding boxes such that nets with bounding boxes apart from each other can be routed in parallel. With both single-net and multi-net parallelism, they demonstrate 21× speedup over the sequential VPR router [18].

3.2 ASIC Routing

ASIC routing is different from FPGA routing, as it works on grids instead of graphs, and the design rules are more complicated. Han et al develop a GPU-accelerated global router [19] and compare with an efficient router *NCTUgr 2.0* [20] on CPU. By decomposing the multi-pin nets into 2-pin nets, as shown in Figure 3, they increase the parallelism and simplify the routing of the each net. Eventually, they achieve 2.5 – 3.9× speedup with 2.5% wirelength degradation. They later improve the scheduling strategy for net-level concurrency and GPU implementation of the maze routing algorithm [21], achieving 4.0× speedup with 1% wirelength degradation compared with another academic router *NTHU-Route 2.0* [22]. So far, we have not seen any work that can achieve significant speedup without any quality loss, for making GPU acceleration even faster than distributed CPU computing is nontrivial. Meanwhile, there is no acceleration work considering complicated design rules in detailed routing yet.

4 GPU ACCELERATION FOR TIMING ANALYSIS

Timing analysis is a ubiquitous step in both front-end and back-end design. It is frequently invoked to guide timing optimization. The runtime characteristics of timing analysis in the back-end flow are different from that in the front-end. In back-end design, computing interconnet delays takes quite much time, since physical wiring needs to be considered. Current efforts on GPU-accelerated timing analysis mostly lie in static timing analysis (STA) and statistical STA (SSTA).

4.1 Static Timing Analysis

In static timing analysis, timing engines need to accomplish a series of tasks: net delay computation, cell delay computation, and timing propagation. Net delay can be computed from the RC tree of each net with parasitics extracted from routing solutions. Cell delay is

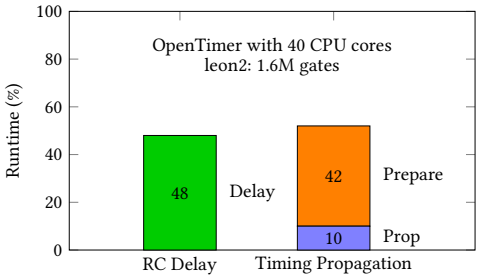

Figure 4: Runtime breakdown for OpenTimer to compute full timing on a million-gate circuit using 40 CPUs [23].

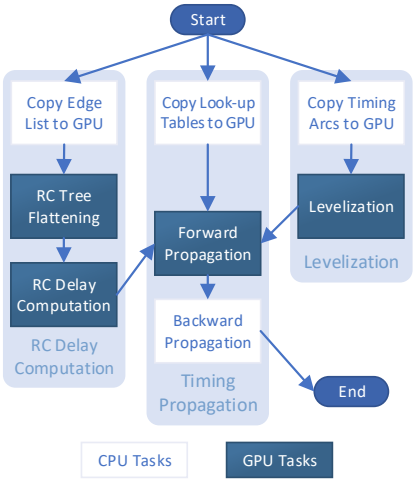

Figure 5: Overall taskflow of the GPU-accelerated STA [23].

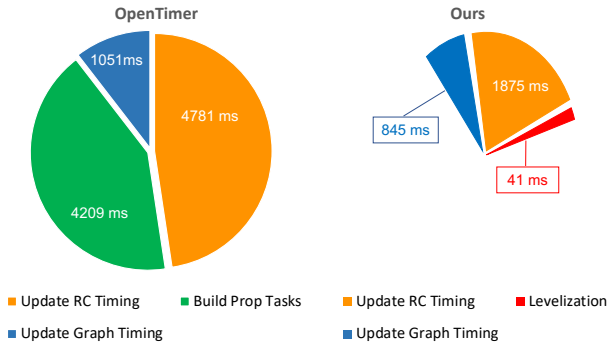

Figure 6: Runtime breakdown of leon2 (21M nodes) [23].

usually computed from lookup tables (LUTs) given the input slews and load capacitance of each cell. Timing propagation finishes the computation of arrival time (w. forward propagation), required arrival time (w. backward propagation), and slack. It needs to be noted that the computation patterns of slews and load capacitance are similar to that of the forward propagation.

Recent works have investigated LUT-based cell delay computation and timing propagation on both ASIC and FPGA [24, 25], while net delays are not considered and levelization is done on CPU. Our study on the state-of-the-art open-source timing engine *OpenTimer* [26] reveals that these two accelerated steps are not actually the runtime bottleneck [23], as shown in Figure 4. Net delay computation and levelization take more than 90% of the total runtime in a full timing analysis. Therefore, we propose acceleration techniques for net delay computation considering Elmore delay models with precomputed

breadth-first search ordering and hybrid CPU-GPU levalization to compose a fully accelerated STA engine. The overall flow is illustrated in Figure 5, where a majority of the computation steps have been moved to GPU. We leave the backward propagation on CPU because it only takes a small portion of the total runtime and the benefits from acceleration are marginal. Eventually, we demonstrate up to 3.69× speedup on million-size benchmarks over 20-thread CPU implementation for full timing analysis. Figure 6 shows the comparison on the runtime breakdown before and after GPU acceleration on a circuit with 21M nodes. We observe significant speedup on the runtime bottlenecks.

4.2 Statistical Static Timing Analysis

SSTA is another variation of timing analysis based on Monte Carlo simulations. Such kind of analysis naturally fits massive parallelization on GPU and thus there have been several works exploring such directions [27–29].

5 CONCLUSION

In this tutorial, we have reviewed the recent efforts on accelerating placement, routing, and timing analysis with GPU. These are fundamental steps for routability and timing optimization in the back-end design flow. From the current status, we summarize the high-level challenges for GPU acceleration as follows:

- lack of parallelism and irregular computation patterns as mentioned in Section 1;
- high expectation to quality and inevitable quality degradation;
- lack of available baseline implementations and high development overhead.

GPU acceleration has not been extensively investigated in design automation yet. There lack mature and standard acceleration paradigms for both academia and industry to follow, and supporting both CPU/GPU is too expensive for most existing tools.

5.1 Future Directions

Future efforts on GPU acceleration can include following directions:

- algorithmic innovation to accelerate practical design stages, such as timing- or routability-driven placement, detailed routing, timing analysis with industrial-strength delay models;
- pushing the speed limit on really hard kernels, such as batched bipartite matching, maze routing, and timing propagation.
- universal frameworks or programming models that can support CPU/GPU programming naturally, such as Tensorflow/PyTorch in deep learning, or something even more fundamental.

All these aspects remain to be explored. When it comes to GPU acceleration, at least one magnitude of speedup over multi-thread CPU is often expected. However, this is unreasonable, unfair, and not good to the advancement of the field. As the design automation problems are extremely hard, pushing the cutting edges little by little is very meaningful and will attract more research efforts.

ACKNOWLEDGE

This project is supported in part by the Beijing Municipal Science and Technology Program (No. Z201100004220007) and the National Key Research and Development Program of China (No. 2019YFB2205000).

REFERENCES

[1] "Leader in gpu computing," https://www.nvidia.com/en-gb/about-nvidia/ai-computing/, Tech. Rep., 2019.
[2] C.-X. Lin and M. D. Wong, "Accelerate analytical placement with gpu: A generic approach," in *DATE*. IEEE, 2018, pp. 1345–1350.
[3] J. Cong and Y. Zou, "Parallel multi-level analytical global placement on graphics processing units," in *ICCAD*. ACM, 2009, pp. 681–688.
[4] T. Chan, J. Cong, and K. Sze, "Multilevel generalized force-directed method for circuit placement," in *ISPD*. ACM, 2005, pp. 185–192.
[5] A. Al-Kawam and H. M. Harmanani, "A parallel gpu implementation of the timber wolf placement algorithm," in *2015 12th International Conference on Information Technology-New Generations*. IEEE, 2015, pp. 792–795.
[6] B. Bredthauer, M. Olbrich, and E. Barke, "Stp-a quadratic vlsi placement tool using graphic processing units," in *International Symposium on Parallel and Distributed Computing (ISPDC)*. IEEE, 2018, pp. 77–84.
[7] Y. Lin, Z. Jiang, J. Gu, W. Li, S. Dhar, H. Ren, B. Khailany, and D. Z. Pan, "DREAMPlace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement," *IEEE TCAD*, 2020.
[8] C.-K. Cheng, A. B. Kahng, I. Kang, and L. Wang, "Replace: Advancing solution quality and routability validation in global placement," *IEEE TCAD*, 2018.
[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "PyTorch: An imperative style, high-performance deep learning library." Curran Associates, Inc., 2019, pp. 8024–8035.
[10] Y. Lin, D. Z. Pan, H. Ren, and B. Khailany, "DREAMPlace 2.0: Open-source gpu-accelerated global and detailed placement for large-scale vlsi designs," in *China Semiconductor Technology International Conference (CSTIC)*, Shanghai, China, June 2020.
[11] S. Dhar and D. Z. Pan, "GDP: GPU accelerated detailed placement," Sept 2018.
[12] S. W. Hur and J. Lillis, "Mongrel: hybrid techniques for standard cell placement," in *ICCAD*, 2000, pp. 165–170.
[13] Y. Lin, W. Li, J. Gu, H. Ren, B. Khailany, and D. Z. Pan, "Abcdplace: Accelerated batch-based concurrent detailed placement on multi-threaded cpus and gpus," *IEEE TCAD*, 2020.
[14] G. E. Blelloch, J. T. Fineman, and J. Shun, "Greedy sequential maximal independent set and matching are parallel on average," *CoRR*, vol. abs/1202.3205, 2012. [Online]. Available: http://arxiv.org/abs/1202.3205
[15] D. P. Bertsekas, "A new algorithm for the assignment problem," *Mathematical Programming*, vol. 21, no. 1, pp. 152–171, 1981.
[16] M. Shen and G. Luo, "Corolla: Gpu-accelerated fpga routing based on subgraph dynamic expansion," in *FPGA*, 2017, pp. 105–114.
[17] M. Shen, G. Luo, and N. Xiao, "Exploring gpu-accelerated routing for fpgas," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 30, no. 6, pp. 1331–1345, 2018.
[18] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed *et al.*, "Vtr 7.0: Next generation architecture and cad system for fpgas," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 7, no. 2, pp. 1–30, 2014.
[19] Y. Han, K. Chakraborty, and S. Roy, "A global router on gpu architecture," in *ICCD*. IEEE, 2013, pp. 78–84.
[20] W.-H. Liu, W.-C. Kao, Y.-L. Li, and K.-Y. Chao, "NCTU-GR 2.0: multithreaded collision-aware global routing with bounded-length maze routing," *IEEE TCAD*, vol. 32, no. 5, pp. 709–722, 2013.
[21] Y. Han, D. M. Ancajas, K. Chakraborty, and S. Roy, "Exploring high-throughput computing paradigm for global routing," *IEEE Transactions on Very Large Scale Integration Systems (TVLSI)*, vol. 22, no. 1, pp. 155–167, 2013.
[22] Y.-J. Chang, Y.-T. Lee, J.-R. Gao, P.-C. Wu, and T.-C. Wang, "NTHU-Route 2.0: a robust global router for modern designs," *IEEE TCAD*, vol. 29, no. 12, pp. 1931–1944, 2010.
[23] Z. Guo, T.-W. Huang, and Y. Lin, "Gpu-accelerated static timing analysis," in *ICCAD*. IEEE Press, November 2020.
[24] H. H.-W. Wang, L. Y.-Z. Lin, R. H.-M. Huang, and C. H.-P. Wen, "Casta: Cuda-accelerated static timing analysis for VLSI designs," in *2014 43rd International Conference on Parallel Processing*. IEEE, 2014, pp. 192–200.
[25] K. E. Murray and V. Betz, "Tatum: Parallel timing analysis for faster design cycles and improved optimization," in *2018 International Conference on Field-Programmable Technology (FPT)*. IEEE, 2018, pp. 110–117.
[26] T. Huang, G. Guo, C. Lin, and M. D. F. Wong, "Opentimer v2: A new parallel incremental timing analysis engine," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2020.
[27] K. Gulati and S. P. Khatri, "Accelerating statistical static timing analysis using graphics processing units," in *2009 Asia and South Pacific Design Automation Conference*. IEEE, 2009, pp. 260–265.
[28] J. Cong, K. Gururaj, W. Jiang, B. Liu, K. Minkovich, B. Yuan, and Y. Zou, "Accelerating Monte Carlo based SSTA using FPGA," in *Proceedings of the 18th annual ACM/SIGDA international symposium on Field programmable gate arrays*, 2010, pp. 111–114.
[29] Y. Shen and J. Hu, "GPU acceleration for PCA-based statistical static timing analysis," in *2015 33rd IEEE International Conference on Computer Design (ICCD)*. IEEE, 2015, pp. 674–679.