Scaling Up Physical Design: Challenges and Opportunities

Guojie Luo[1,3]
gluo@pku.edu.cn

Wentai Zhang[1]
rchardx@pku.edu.cn

Jiaxi Zhang[1]
zhangjiaxi@pku.edu.cn

Jason Cong[2,3,1]
cong@cs.ucla.edu

[1]Center for Energy-Efficient Computing and Applications, School of EECS, Peking University
[2]Computer Science Department, University of California, Los Angeles
[3]PKU-UCLA Joint Research Institute in Science and Engineering

ABSTRACT

Due to the continuous scaling of integration density and the increasing diversity of customized designs, there are increasing demands on the scalability and the customization of EDA tools and flows. Commercial EDA tools usually provide an interface of TCL scripting to extract and modify the design information for a flexible design flow. However, we observe that the current TCL scripting is not designed for the complete netlist extraction, resulting in a significant degradation in performance. For example, it takes over 20 minutes to extract the complete netlist of a 466K-cell design using TCL. This extraction may be repeated several times when interfacing between the existing EDA platforms and the actual distributed EDA algorithms. This drastic decrease in efficiency is a great barrier for customized EDA tool development. In this paper, we propose to build a distributed framework on top of TCL to accelerate the netlist extraction, and use the distribution detailed placement as an example to demonstrate its capability. This framework is promising in scaling out physical design algorithms to run on a cluster.

CCS Concepts

•Hardware → Physical design (EDA); Methodologies for EDA; •Computing methodologies → *Parallel algorithms; Distributed algorithms;*

Keywords

Physical Design; Detailed Placement; FPGA; TCL; Distributed Computing; Spark

1. INTRODUCTION

The computational demand of physical design and EDA tools keeps growing, due to the increasing complexity of electronics designs. The semiconductor industry is involved in the new applications including medical technology, automotive, robotics, and energy systems, which in turn exposes some complex design problems and thus needs greater computational power [8].

The EDA vendors have already adopted a distributed storage solution for data management [5, 6]. Some EDA tools provide multi-threaded solutions to relieve the runtime issue. However, the emerging and mainstream distributed computing infrastructures are not fully adopted by EDA tools, due to the cost of rewriting software and the unclear pricing models. In the meanwhile, EDA users have various needs for customized tools in their flow. In addition to the core physical design steps, users may develop additional tools for their specific needs. There are strong needs for powerful and extensible framework to design EDA tools and flows.

A related topic is putting EDA tools and solutions in the cloud [30]. These recent efforts mainly spread across the applications of training, demonstrations, and web-based collaboration. These can be viewed as the interaction-intensive end of the design platform, which create opportunities and hide the complexity for the development of parallel and distributed computing tools in the compute-intensive end. And there is another example in academia, called bX [27], to provide computational power for regression testing of EDA algorithms.

Here, we investigate a distributed computing framework for EDA algorithms and flows. We take advantage of the progress in the computational engines (e.g., Spark [37]) in the big data ecosystem, and design a framework capable of interfacing the existing commercial EDA platforms. In this way, the academia will be able to design more distributed EDA algorithms and test with industrial-grade design examples; and the industry will have a low cost to migrate some of the design tools to this distributed framework. The interface implements design query and modification through TCL scripting supported by mainstream EDA tools. TCL scripting is a good candidate to implement a general interface across commercial EDA tools, and there have been some practices of customized in-house EDA tools developed using TCL [35, 16]. However, we observe that existing TCL support is not designed for high-throughput queries. The extraction of the whole netlist takes an ineligible amount of time. We propose a distributed parser to efficiently read design data from existing EDA platforms, and provide a solution to maintain data consistency when interact distributed

ISPD'16, April 03-06, 2016, Santa Rosa, CA, USA
© 2016 ACM. ISBN 978-1-4503-4039-7/16/04...$15.00
DOI: http://dx.doi.org/10.1145/2872334.2872342

EDA algorithm with existing tools. We also demonstrate the capability of this framework using a distributed detailed placement algorithm.

The remainder of this paper is organized as follows. Section 2 states the background and related work. Section 3 proposes the design and details of a distributed EDA framework with an application example of detailed placement. Then Section 4 describes the experimental results. Section 5 discusses the challenges and opportunities for a scalable EDA framework.

2. BACKGROUND AND RELATED WORKS

First, we summarize the latest efforts in putting EDA in the cloud. In the meanwhile, we give a short introduction to the techniques like Spark and Docker in the big data and cloud ecosystem. Though it may not be an urgent task for the EDA companies to take advantage of the progress in these techniques, it is about time to design new distributed physical design algorithms for scalability. In the next section, we will propose a distributed computing framework that can build on top of existing EDA platforms.

A concept related to the scalability of EDA tools is the cloud EDA. The EDA vendors have been investigated the opportunity of offering the solutions in the cloud [30]. Cadence provides Hosted Design Solutions [14] as a production design environment in the private cloud. Synopsys has put its functional verification solution VCS in the cloud [19]. And Mentor Graphics has a cloud-based SystemVision [7] for modeling and design of electro-mechanical systems. Besides, there have been multiple companies putting their products in the cloud, including Altium, OneSpin, Plunify, Tabula, etc. Recently, IBM provides its high-performance services for EDA through SiCAD [21].

On the other hand, the MapReduce [13] programming model and infrastructure have been used widely for scalable data processing in the big data ecosystem. Users of MapReduce implements all the computations using only two functions, map and reduce, where a unit computation takes the input of <key,value> pairs and generates the output of another set of <key,value> pairs. Though the MapReduce model seems less flexible and has lower peak performance [29] than the traditional message-passing model [17], it increases the productivity and has even better performance for the programs written by non-experts in distributed computing. Spark [37] is an open-source data-parallel computation engine using the MapReduce programming model. Different from Hadoop [34], a previous open-source implementation of MapReduce, Spark enables efficient iterative algorithms and interactive queries by keeping data in memory using the Resilient Distributed Datasets (RDDs). Moreover, it supports general computation DAGs and allows optional specification of data partitioner to avoid data movement. Since most EDA algorithms are iterative, Spark is a suitable engine to scale out EDA tools.

Linux container is a lightweight virtualization technique, which provides an isolated kernel namespace for the processes, file system and network without running a full OS on virtual hardware. Docker [4] is a representative implementation. The comparative study of virtual machines (VMs) and containers shows that containers have equal or better performance than VMs and lower overhead in OS interaction [15] . The basic idea of container is illustrated in Figure 1. We will show in the next section that container is useful to manage existing EDA platforms in the environment of Linux clusters.

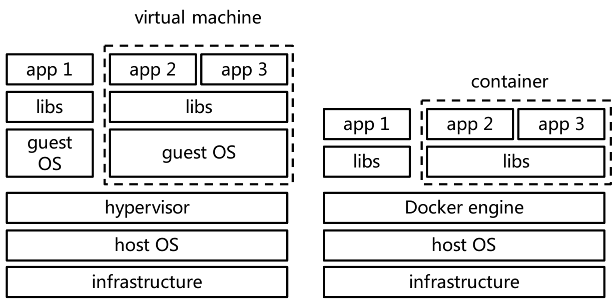


Figure 1: Virtual machine vs. container.

Inspired by the techniques above, we propose a distributed EDA framework for the scalability of physical design algorithms. Comparing with existing cloud EDA solutions, it is a computational infrastructure that can deploy either in the public cloud or private cloud.

3. FRAMEWORK DESIGN

3.1 Overview

The overall design of our proposed framework for distributed EDA algorithms and flows is illustrated in Figure 2. Existing EDA platforms are supported using TCL scripts for design data extraction and modification. The design data and the FPGA architecture or technology information are presented on top of the distributed computing engine of Spark. While we can follow OpenAccess as the data model of post-synthesis design, it is promising to design and develop portable physical design algorithms and flows in the distributed computing engine of Spark.

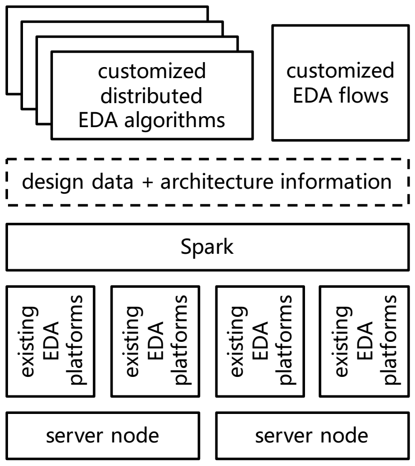


Figure 2: The distributed EDA framework.

3.2 Interfacing Existing EDA Platforms

TCL is a de facto standard for mainstream commercial EDA products. One has to import the design data from existing EDA platforms to the distributed computing framework using TCL scripts. The outline of the parser code is shown in Listing 1, which is implemented for the Xilinx Vivado platform. The scripts are only slightly different for the Altera Quartus platform.

Listing 1: Extract netlist from Vivado using TCL

```
foreach cell [get_cells ...] {
  puts ...
}
foreach net [get_nets ...] {
  foreach pin [get_pins -of $net] {
    var cell [get_cells -of $pin]
    puts ...
  }
}
```

However, we observe the TCL execution on existing FPGA EDA platforms is relatively slow. For example, it takes over 20 minutes to extract the complete netlist of a 466K-cell design using TCL. This extraction may be repeated several times when interfacing between the existing EDA platforms and the actual distributed EDA algorithms. And this runtime cannot be directly reduced given more computing resources. In order to solve this issue and make the distributed computing framework meaningful, we propose a parallel parser to accelerate the execution of TCL scripts. The parser design is illustrated in Figure 3. In this example, four instances of Vivado execute on two server nodes. In this implementation, the parser reads partial data from each instance, and then combine and convert the design to the in-memory RDD in Spark. The RDD can be partitioned into sub-designs for further MapReduce steps, with an example shown in the next subsection.

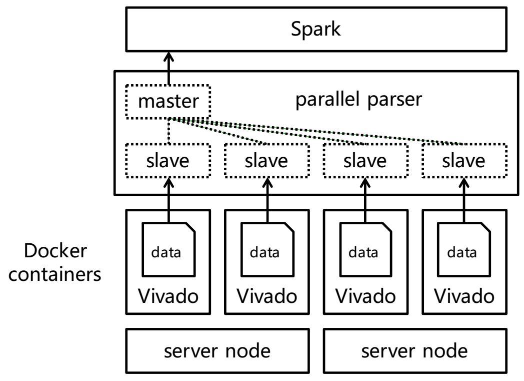
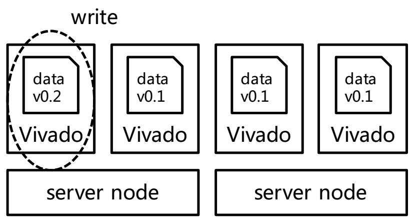
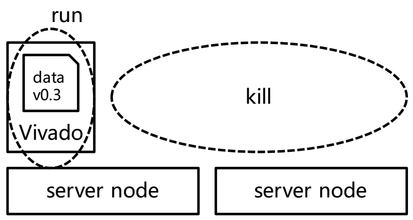


Figure 3: Parallel reads for efficient parsing.

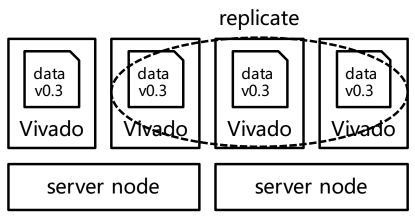
After a design is processed by a distributed algorithm in Spark, the result has to be written back if a conventional step in Vivado is needed. This procedure is briefly listed in Figure 4. First, as shown in Figure 4a, the updated design (from data v0.1 to v0.2) in Spark is written back to a single instance of Vivado. Second, the outdated instances (data v0.1) are stopped and removed; at the same time, the remaining instance can run a conventional step in Vivado to obtain a further updated design (data v0.3), as shown in Figure 4b. Last, the remaining instance can be replicated as in Figure 4c, so that the parallel parsing can be performed to run another distributed algorithm in Spark. The instances of Vivado are managed using the Docker technique, where they can be killed or replicated conveniently. The checkpoint and restore operations are able to save and restore the in-memory data, and thus can be used to replicate a live instance [31].



(a) Write data back to a single instance when a conventional step in Vivado is needed.



(b) Kill the outdated instances, and run a conventional step in the remaining instance.



(c) Replicate the instance with the latest data, and get ready for the next parallel parsing.

Figure 4: Writing strategy for data consistency.

In this way, we are able to develop new distributed algorithms in Spark on top of Vivado. The runtime of the TCL-based parser is accelerated by parallel reads from multiple identical instances of Vivado. The data consistency for data write-backs can be guaranteed using the Docker technique to management the Vivado instances. This methodology is straightforward to be applied to any other mainstream EDA platforms that support TCL scripting.

3.3 Example Application: Detailed Placement

The detailed placement algorithms [23, 12, 28] usually include global swapping and local swapping. As an illustrative example, we implement a brute-force algorithm of local swapping on Spark to examine the capability of the distributed EDA framework.

The information produced by previous TCL parsing stage contains the lists of the cells, nets, initial placement, and feasible placement locations. The netlist and the placement region is processed and partitioned into "DP tile" structures after a few MapReduce steps. Each DP tile consists of its partial netlists and subregion information needed by the regional local swapping algorithm. The set of DP tiles is the RDD for a map operator to do parallel swapping. This map operator takes a DP tile as the input and generates a new DP tile after local swapping.

Compared with the conventional sequential solution, our method partitions the whole FPGA into $N \times N$ DP tiles and performs local swapping in each tile. During the swapping in each individual tile, we sweep a sliding window and enumerate all possible permutations of the cells in this window to pick a partially best solution. A size of 3×2 and 2×3 are selected for the sliding window, and the $6! = 720$ permutations of each window can be examined in a reasonable amount of time. After completing one iteration, the best permutation of cells is committed, and the sliding window moves to next position. We send all the tiles in N rounds, and a group of map operations process N tiles in parallel. These N tiles are chosen in a way that there are not any pair of tiles on the same row or column, so that the estimation of wirelength improvement in each tile is consistent. The partitioning and local swapping scheme are illustrated in Figure 5.

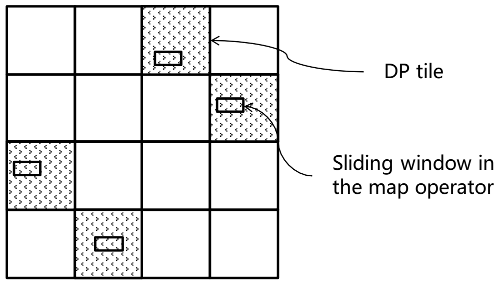


Figure 5: Distributed detailed placement scheme.

4. EXPERIMENTAL RESULTS

Our experiments are run on a Linux cluster with four nodes, each with two 6-core Intel Xeon Processor E5-2620 v3 at 2.40GHz and 64GB memory. The distributed computing engine is powered by Spark 1.5.2 and HDFS of Hadoop 2.6.3. The four nodes are connected by Gigabit Ethernet.

The summary of the test cases is described in Table 1. The test cases are obtained from the Titan benchmarks [24]. They are synthesized and placed using Xilinx Vivado 2015.3 targeting VC707 (part name XC7VX485TFFG1761-2). The total number of logic cells and a short description are also included in the table.

Table 1: Summary of the test cases

name	#cells	description
SLAM_spheric	87K	spherical coordinates algorithm for SLAM
bitcoin_miner	222K	Two-core version of the bitcoin FPGA miner
guassianblur_d1	466K	One of the pipelined loops for 3D Gaussian convolution

4.1 Distributed TCL Parser

After synthesis and placement, the next step in our experiments is to load the data from Xilinx Vivado to memory. We use TCL scripts to extract the design information.

The parsing time of the three test cases in Vivado is illustrated in Figure 6. The part of "load design" is a one-time execution to load a design in Vivado. The parts of "foreach cell" and "foreach net" correspond to the execution of TCL scripts in Listing 1. These two parts are usually executed multiple times when there are multiple interactions between the Spark program and the Vivado services. The runtime of TCL execution is too long for such interactions, and hurts the speedup from any distributed algorithm in Spark.

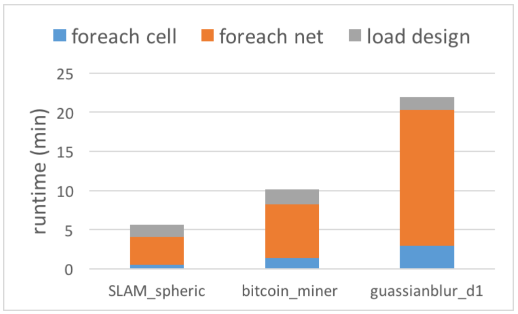


Figure 6: Decomposition of parsing time.

Thus, we use the parallel parser as illustrated in Figure 3 previously, and achieve about $3\times$ speedup with $3\times$ memory using four instances of Vivado. The TCL runtime and memory consumption are shown in Table 2. The runtime can be further reduced using more Vivado instances.

Table 2: TCL runtime time and memory consumption using four instances of Vivado.

test case	TCL time		memory	
	(min)	decr.	(GB)	incr.
SLAM_spheric	1.0	4.0×	6.4	3.2×
bitcoin_miner	2.1	3.9×	8.0	3.2×
guassianblur_d1	6.9	2.9×	11.0	3.0×

The current support of live replication of Vivado instances using Docker and CRIU [1] is experimental. The experiments in [10] show that it takes about 9 seconds to checkpoint and restore a container with 1GB memory. These features are under development by the communities of Linux containers and we can expect a runtime improvement in the near future.

4.2 Distributed Detailed Placement

The distributed detailed placement algorithm is written in Python, and is executed by the command "spark-submit –executor-memory 4G dplace.py".

The runtime results of the distributed detailed placement are listed in Table 3, and the quality of wirelength improvement is similar to the sequential version. The results show great potentials in speedup in the distributed computing.

Table 3: Runtime of distributed detailed placement with different number of parallel tiles

test case	runtime (min)	
	1 tile	48 tiles
SLAM_spheric	36	18
bitcoin_miner	51	20
guassianblur_d1	611	25

5. CHALLENGES AND OPPORTUNITIES

In the previous sections, we demonstrate a proof-of-concept for a distributed EDA framework to scale out the physical design algorithms. In this section, we highlight the challenges and opportunities to attract the efforts of the physical design community to design and develop new distributed algorithms.

The following are the necessary components to make the distributed EDA framework for scalable physical design generic and useful.

Algorithmic kernels. These algorithmic kernels are preferred to be either a map operator in the MapReduce distributed computing paradigm for the partitioned design data, or a composed series of MapReduce operations. There have already been some widely-used physical design tools developed. FLUTE [11] is one of such examples, which is adopted by most global routers in the ISPD routing contests [26, 25] to construct rectilinear Steiner minimal trees for multi-pin nets. The algorithmic kernels are analogous to the cognitive computing services for language, speech, vision and data on IBM Bluemix [3]. It is necessary and challenging to build and maintain a library for such algorithmic kernels.

Standard interfaces. The standard interfaces include the ones connecting some algorithmic kernels to form a complete EDA algorithm, as well as the ones connecting existing EDA platforms and the distributed computing framework. Though the format of raw design data will keep changing due to new design rules and new objectives, it is feasible to provide a conversion operator as an algorithmic kernel for backward compatibility. The relatively stable standard interfaces across different EDA platforms and raw design data will extend the lifetime of an algorithmic kernel and help the growing of the algorithmic library. OpenAccess [18] sets a good example of an open-source data model and API for physical design. Given the necessity of a distributed computing framework to scale out physical design, it is about time to re-define a new set of standard interfaces in such context.

Flow composition. The current implementation of OpenAccess only supports the flow composition by tool-by-tool inter-operation. Given the algorithmic kernels and standard interfaces, the distributed computing framework will be able to support both tool-by-tool inter-operation (macro-flow composition) and the connection of algorithmic kernels (micro-flow composition). The former one is conventional, and it will help the innovations in EDA design flow when there are sufficient amount of tools supported in the framework. The latter one is promising to keep up the innovations in EDA algorithms. For example, there have been a series of routibility-driven placement contests [33, 32, 36, 9]. Most of these algorithms share similar algorithmic kernels, and varies in some of these kernels and the detailed tuning of the flow. Such research activities can attract boarder attentions, if the existing kernels and flows can be reused; so that a group new to this area does not need to start from scratch but can focus on the innovation of the critical kernels (e.g., the routability estimator, the inflation strategy, the placement objective, etc.).

The distributed computing framework have potential benefits in the following aspects.

Scalability. There are extensive efforts to develop the distributed computing engines like Spark and Tachyon [22], which are motivated by big data applications. The porting of EDA algorithms on new distributed engines can take advantage of such progress in the big data ecosystem, and keep up with the scaling of design complexity in the long run. Though there are results [29] showing that Spark is one order of magnitude slower than MPI for specific data sets, it has a data management infrastructure better in handling node failure and data replication. Moreover, these emerging distributed engines lower the barrier to get involve with distributed computing, and bring more EDA experts to implement algorithmic kernels and flows in the framework.

Reproducible results. It is hopefully the algorithmic kernels and flows can be encapsulated with its dependent dynamic linking libraries using the Linux container technologies like Docker. In this way, they are executable in any mainstream Linux clusters without configuration or compilation issues and generate reproducible results. There is also an opportunity to provide "cloud" services for such distributed EDA framework, so that the design data, benchmarks and design flows can be shared in the community similar to GitHub [2]. Moreover, when the flows are executed in the cloud by the masses on some existing benchmarks, it is possible to apply the idea of data deduplication to skip the execution of the first few stages if a flow has been executed before. It will save runtime for the development of late-stage physical design algorithms.

Collaborative innovation. The collaborative innovation comes with the standard interfaces and execution of the algorithmic kernels and flows in the distributed computing framework. And it is promising to bridge the gap between industry and academia. The opportunity of "cloud" services to share design data, benchmarks and design flows is a way to boost collaborations. On one hand, when the results of the algorithmic kernels and a design flow from academia are reproducible, it will be easier for the industry to try the flow and get direct access to the new ideas from academia. On the other hand, since the framework is compatible with existing EDA platforms, the industry can set up an evaluation system like ImageNet [20] for academia to submit their tools and flows in an executable form, with industry-grade design data without worrying about sensitive data leakage.

Education. The distributed computing framework creates opportunities for instructors to provide a design flow to students in a quick way. The students are not only able to see an example of the whole design flow (the highest-level of a composed flow) conveniently, but will also be much easier than nowadays to replace a design step with their own algorithm. The lower barrier to getting familiar with design flows and experiment on EDA algorithms is promising to attract more students understand the EDA field.

6. CONCLUSION

In this paper, we propose a distributed computing framework for extreme-scale EDA algorithms development. Furthermore, we outline the challenges and opportunities of how such framework will benefit the innovations in both EDA algorithms and flows, as well as the collaboration between industry and academia.

Specifically, our proposed framework enables the design and development of new distributed EDA algorithms while being compatible with commercial EDA design platforms. This framework uses TCL language to interact with existing EDA platforms, and converts the design information to a distributed in-memory data structure in Spark. The current TCL support in existing EDA platforms is mainly designed for customized flows and lightweight customized tools. As a result, we observe that the extraction of the complete design information using TCL takes a significant amount of time. And this will cancel out the speedup from the distributed EDA algorithms in our proposed framework according to Amdahl's law. To solve this issue, we start multiple instances to open the same design in multiple server nodes, and extract the design data in a distributed way. The design data is stored in memory for further processing in the distributed computing engine of Spark. To demonstrate the proposed framework, we implement a distributed detailed placement algorithm and show a substantial speedup.

In the end, we summarize the challenges and opportunities to scale out physical design algorithms in a distributed framework. It is promising that such framework will accelerate the innovations in both large-scale EDA algorithms as well as new EDA design flows.

7. ACKNOWLEDGMENTS

This work is partly supported by National Natural Science Foundation of China (NSFC) Grant 61202073, Research Fund for the Doctoral Program of Higher Education of China (MoE/RFDP) Grant 20120001120124, and Beijing Natural Science Foundation (BJNSF) Grant 4142022.

8. REFERENCES

[1] CRIU, a project to implement checkpoint/restore functionality for Linux in userspace. http://www.criu.org/. [Online; accessed Feb 5, 2016].

[2] GitHub. https://github.com/. [Online; accessed Feb 5, 2016].

[3] IBM Watson Developer Cloud. http://www.ibm.com/smarterplanet/us/en/ibmwatson/developercloud/. [Online; accessed Feb 5, 2016].

[4] What is Docker? https://www.docker.com/what-docker. [Online; accessed Feb 5, 2016].

[5] EMC Isilon Storage Best Practices for Electronic Design Automation. Technical Report H11909, EMC Corporation, 2013.

[6] EMC Isilon NAS: Performance at Scale for Electronic Design Automation. Technical Report H13233.1, EMC Corporation, 2014.

[7] systemvision.com: A Cloud-based Engineering Community for System Modeling & Design. Technical Report MGC 04-15 1033380-w, Mentor Graphics Corporation, 2015.

[8] R. I. Bahar, A. K. Jones, S. Katkoori, P. H. Madden, D. Marculescu, and I. L. Markov. Workshops on Extreme Scale Design Automation (ESDA) Challenges and Opportunities for 2025 and Beyond. Technical report, 2014.

[9] I. S. Bustany, D. Chinnery, J. R. Shinnerl, and V. Yutsis. ISPD 2015 Benchmarks with Fence Regions and Routing Blockages for Detailed-Routing-Driven Placement. In *Proceedings of the 2015 Symposium on International Symposium on Physical Design (ISPD'15)*, pages 157–164, New York, New York, USA, 2015.

[10] Y. Chen. Checkpoint and Restore of Micro-service in Docker Containers. In *Proceedings of the 3rd International Conference on Mechatronics and Industrial Informatics*, number Icmii, pages 915–918, Paris, France, 2015. Atlantis Press.

[11] C. Chu and Y. C. Wong. FLUTE: Fast lookup table based rectilinear steiner minimal tree algorithm for VLSI design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(1):70–83, 2008.

[12] J. Cong and Min Xie. A Robust Mixed-Size Legalization and Detailed Placement Algorithm. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(8):1349–1362, 2008.

[13] J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In *Proceedings of the 6th Conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, page 10, Berkeley, CA, USA, 2004.

[14] L. Drenan. Cadence Hosted Design Solutions: Software-as-a-service capability for the semiconductor industry. Technical Report 702 6/13 SA/DM/PDF, Cadence Design Systems, 2013.

[15] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio. An updated performance comparison of virtual machines and Linux containers. Technical Report RC25482 (AUS1407-001), IBM Research, 2014.

[16] J. Friesen. An approach for better debuggability of Tcl- driven EDA methodologies. In *CDNLive Silicon Valley 2015*, 2015.

[17] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[18] M. Guiney and E. Leavitt. An introduction to openaccess an open source data model and API for IC design. In *Proceedings of the Asia and South Pacific Conference on Design Automation (ASP-DAC'06).*, pages 434–436, 2006.

[19] D. Hsu. EDA in the Clouds: Myth Busting. *Synopsys Insight*, 2011.

[20] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR'09)*, pages 248–255, 2009.

[21] R. C. Johnson. IBM Renting Its EDA Tools, 2015.

[22] H. Li, A. Ghodsi, M. Zaharia, S. Shenker, and I. Stoica. Tachyon: Reliable, Memory Speed Storage for Cluster Computing Frameworks. In *Proceedings of the ACM Symposium on Cloud Computing (SOCC'14)*, pages 1–15, New York, New York, USA, 2014.

[23] Min Pan, N. Viswanathan, and C. Chu. An efficient and effective detailed placement algorithm. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD'05)*, pages 48–55, 2005.

[24] K. E. Murray, S. Whitty, S. Liu, J. Luu, and V. Betz. Timing-Driven Titan: Enabling Large Benchmarks and Exploring the Gap between Academic and Commercial CAD. *ACM Transactions on Reconfigurable Technology and Systems*, 8(2):1–18, 2015.

[25] G.-J. Nam, C. Sze, and M. Yildiz. The ISPD global routing benchmark suite. In *Proceedings of the 2008 international symposium on Physical design (ISPD'08)*, page 156, New York, New York, USA, 2008.

[26] G.-J. Nam, M. Yildiz, D. Z. Pan, and P. H. Madden.

ISPD placement contest updates and ISPD 2007 global routing contest. In *Proceedings of the 2007 international symposium on Physical design (ISPD'07)*, page 167, New York, New York, USA, 2007.

[27] A. Ng and I. Markov. Toward Quality EDA Tools and Tool Flows Through High-Performance Computing. In *Proceedings of the Sixth International Symposium on Quality of Electronic Design (ISQED'05)*, pages 22–27, 2005.

[28] S. Popovych, H.-H. Lai, C.-M. Wang, Y.-L. Li, W.-H. Liu, and T.-C. Wang. Density-aware Detailed Placement with Instant Legalization. In *Proceedings of the 51st Annual Design Automation Conference on Design Automation Conference (DAC'14)*, pages 1–6, New York, New York, USA, 2014.

[29] J. L. Reyes-Ortiz, L. Oneto, and D. Anguita. Big Data Analytics in the Cloud: Spark on Hadoop vs MPI/OpenMP on Beowulf. *Procedia Computer Science*, 53(1):121–130, 2015.

[30] L. Stok. The Next 25 Years in EDA: A Cloudy Future? *IEEE Design & Test*, 31(2):40–46, 2014.

[31] M. Tessel, M. Crosby, and D. Mónica. Full Sail Ahead: What's Next For Container Technology. In *LinuxCon + CloudOpen + ContainerCon NA 2015*, 2015.

[32] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei. The DAC 2012 routability-driven placement contest and benchmark suite. In *Proceedings of the 49th Annual Design Automation Conference (DAC'12)*, page 774, New York, New York, USA, 2012.

[33] N. Viswanathan, C. J. Alpert, C. Sze, Z. Li, G.-J. Nam, and J. A. Roy. The ISPD-2011 routability-driven placement contest and benchmark suite. In *Proceedings of the 2011 international symposium on Physical design (ISPD'11)*, page 141, 2011.

[34] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, 4th editio edition, 2015.

[35] L. Wu. Accelerating Physical Design Flow in Laker with TCL Applications and Third Party Tool Integration. In *SNUG Taiwan 2015*, 2015.

[36] V. Yutsis, I. S. Bustany, D. Chinnery, J. R. Shinnerl, and W.-H. Liu. ISPD 2014 benchmarks with sub-45nm technology rules for detailed-routing-driven placement. In *Proceedings of the 2014 on International symposium on physical design (ISPD'14)*, pages 161–168, New York, New York, USA, 2014.

[37] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, pages 15–28, San Jose, CA, 2012.