Electronic Design Automation with Graphic Processors: A Survey

Yangdong Deng
Tsinghua University
China
dengyd@tsinghua.edu.cn

Shuai Mu
Tsinghua University
China
mus04ster@gmail.com

now
the essence of knowledge
Boston – Delft
Editorial Scope

Foundations and Trends® in Electronic Design Automation will publish survey and tutorial articles in the following topics:

- System Level Design
- Behavioral Synthesis
- Logic Design
- Verification
- Test
- Physical Design
- Circuit Level Design
- Reconfigurable Systems
- Analog Design

Information for Librarians

Foundations and Trends® in Electronic Design Automation, 2013, Volume 7, 4 issues. ISSN paper version 1551-3939. ISSN online version 1551-3947. Also available as a combined paper and online subscription.
Electronic Design Automation with Graphic Processors: A Survey

Yangdong Deng¹ and Shuai Mu²

¹ Institute of Microelectronics, Tsinghua University, Beijing, 100084, China, dengyd@tsinghua.edu.cn
² Institute of Microelectronics, Tsinghua University, Beijing, 100084, China, mus04ster@gmail.com

Abstract

Today’s Integrated Circuit (IC) architects depend on Electronic Design Automation (EDA) software to conquer the overwhelming complexity of Very Large Scale Integrated (VLSI) designs. As the complexity of IC chips is still fast increasing, it is critical to maintain the momentum towards growing productivity of EDA tools. On the other hand, single-core Central Processing Unit (CPU) performance is unlikely to see significant improvement in the near future. It is thus essential to develop highly efficient parallel algorithms and implementations for EDA applications, so that their overall productivity can continue to increase in a scalable fashion. Among various emergent parallel platforms, Graphics Processing Units (GPUs) now offer the highest single-chip computing throughput. A large body of research, therefore, has been dedicated to accelerating EDA applications with GPUs. This monograph is aimed to develop a timely review of the existing literature on GPU-based EDA computing. Considering the substantial diversity of VLSI CAD algorithms, we extend a taxonomy of EDA computing patterns, which can
be used as basic building blocks to construct complex EDA applications. GPU-based acceleration techniques for these patterns are then reviewed. On such a basis, we further survey recent works on building efficient data-parallel algorithms and implementations to unleash the power of GPUs for EDA applications.

**Categories and Subject Descriptors:** J.6 [Computer-Aided Engineering] — Computer-aided design (CAD).

**General Terms:** Algorithms, Design, Performance

**Additional Keywords and Phrases:** Electronic Design Automation (EDA), VLSI, GPU, Graphics Processor, GPGPU, logic simulation, circuit simulation, matrix, linear algebra, sparse matrix, graph traversal, graph algorithm, dynamic programming, simulated annealing, structured grid
# Contents

1 Introduction 1

2 GPU Architecture and Programming Model 9
   2.1 GPU Background 9
   2.2 GPU Hardware Architecture 13
   2.3 GPU Programming Model 27

3 EDA Computing Patterns 41

4 Accelerating Key Design Patterns on GPUs 47
   4.1 Dense Linear Algebra 47
   4.2 Sparse Linear Algebra 66
   4.3 Graph Algorithms 83
   4.4 Backtrack and Branch-and-Bound 98
   4.5 MapReduce 103
   4.6 Dynamic Programming 107
   4.7 Structured Grid 110

5 GPU-Accelerated EDA Applications 113
   5.1 System-Level Design 115
   5.2 RTL Design 118
   5.3 Physical Design 124
   5.4 Simulation 135
Introduction

As the foundation of information technology, Integrated Circuits (ICs) are playing a fundamental role in our society. In the foreseeable future, IC technology will still be one of the major enablers for sustainable development. To further improve the working efficiency and living standards of the human beings, the number of ICs deployed around the world will still be rapidly increasing in the future. It is predicted that 15X more transistors are going to be deployed in the next 5 years to “manage, store, and interpret data” [194].

At the same time, the complexity of ICs has been growing as indicated by Moore’s law to maintain the momentum towards increasing performance and functionality. Today, it is already feasible to integrate over 7 billion transistors on a consumer IC chip [187]. To conquer the overwhelming complexity of modern ICs, circuit designers depend on Electronic Design Automation (EDA) software to convert a design intention into working silicon. EDA tools, therefore, have to be scalable with the growing IC complexity, so that the design turnaround time can be kept in a reasonable level. Current EDA tools are facing challenges from two ends, big system and small physics [216]. The former means the integration of a whole hardware/software system onto a
single chip, while the latter involves the manufacturability, reliability, and other issues incurred by the shrinking physical size of IC fabrication processes. Both trends pose significant requirements to the processing throughput of EDA software.

In the past, the performance scalability of EDA tools had always been the result of two interacting factors, smarter algorithms and faster CPUs. The latter factor is especially handy because the same EDA algorithm automatically runs faster on a CPU with higher performance. In early 2000s, however, single-core CPU performance is saturating due to the inability to extract more instruction-level parallelism and improve power efficiency. Such a stall in computing performance had serious implications on the design turnaround time of IC design projects. Given the complexity of today’s IC designs, the runtime of EDA applications can still be excessive even using the best algorithm to date. For instance, a timing analysis will take a couple of hours to perform on a 5M-gate design. Such a runtime seriously constrains the number of optimization steps that can be conducted in a given design turnaround time, since virtually every post-synthesis optimization operation requires a run of timing analysis to validate the correctness. A runtime of a few hours suggests that only a small portion of the complete solution space can be explored and the design quality has to be relaxed. Another example is the circuit simulation problem. Given a Giga-Hertz phase-lock loop (PLL) circuit, a transient analysis needs to simulate the circuit for millions of cycles before the frequency can be stabilized. Thus a complete run will take months to finish on a single CPU. Besides, the continuously shrinking market window of today’s electronic appliances also poses challenging requirements to the productivity of EDA software.

In spite of the relative saturation of single-core CPU performance in the conceivable future, the semiconductor processes are still offering continuously growing integration capacity. As a result, all major CPU vendors switched to offer multi-core products since 2006. Multi-core processors are inevitably becoming the dominant computing platform for EDA applications. Accordingly, it is crucial to develop parallel solutions to EDA software such that the momentum of function increase in VLSI designs can be maintained [16].
In the past few years, major EDA vendors proposed R&D initiatives to take advantage of the computing power of multi-core processors [220]. At the present time, the POSIX threads or Pthreads [115] based multithreading has been the most popular programming model for multi-core CPUs. Multithreaded versions of cutting-edge EDA software have already been released. Such applications include parallel circuit simulator (e.g., [42, 240]), router (e.g., [241]), and physical verification (e.g., [126]). Among these, multithreaded parallel circuit simulation proves to be especially successful. Meanwhile, the academia also introduced parallel algorithms for many EDA applications (e.g., [110, 157, 170, 220]).

Despite their many successful applications, the multithreaded parallel programming model on multi-core CPUs still has serious limitations. A CPU thread is associated with a relatively high overhead in initialization, context switching, and synchronization [40]. Accordingly, P threads and similar programming models belong to the category of coarse-grain multithreading, which suggests parallel processing of tasks and/or large chunks of data of a problem. However, many complex EDA applications feature abundant fine-grain parallelism (i.e., data parallelism) exemplified by matrix and graph operations. A multi-core microprocessor at most supports a few tens of threads and cannot fully take advantage of the inherent fine-grain parallelism. In addition, the scalability of a coarse-grained multithreaded program is seriously limited by the thread management overhead. A context switching of a thread on a multi-core CPU takes a few hundreds of microseconds [145]. Generally, such an overhead will outweigh the speed-up of increasing parallelism when the number of threads is beyond a given level. A recent work showed that the performance of a highly optimized parallel logic simulator saturated at 15 threads on a 10-core CPU [201].

The above problems of multi-core processors as well as the pursuit for more computing power motivate EDA researchers and engineers to explore alternative parallel computing platforms. Recently, Graphic Processing Units (GPUs) have emerged as a new general-purpose computing platform [28, 195, 196]. GPUs were originally designed as application-specific ICs for graphics rendering. Pushed by the relentless pursuit for better visual experiences, GPUs evolved to offer both high
Introduction

Fig. 1.1 Comparison of peak throughput of CPUs and GPUs.

Figure 1.1 illustrates the comparison of peak throughput of CPUs and GPUs. In 2004, NV35 GPU began to deliver a higher level of performance than the best CPU at that time. Current GPUs outperform their multi-core CPU equivalents by a factor of over 30 in terms of peak computing throughput.

The above performance trend is depicted in Figure 1.1, where the computing throughputs of NVIDIA and AMD GPUs and Intel CPUs are compared in terms of Giga FLoating Operation Per Second (GFLOPS). We gathered performance data from publically available datasheets [5, 186]. GPU chip makers usually release multiple GPUs with varying performance levels at each technology node. Meanwhile, the above three companies have different schedules for releasing new products. In Figure 1.1 we only show the “flagship” GPU for each generation and take NVIDIA’s release schedule as the time reference. Clearly, GPU has been outperforming CPU since 2004 and the performance gap is still broadening.

Along with the high computing throughput, GPUs are also equipped with a high bandwidth memory bus because it is installed on the
Fig. 1.2 Comparison of peak throughput of CPUs and GPUs.

graphics card and dedicated to GPU applications. The memory characteristics of major GPUs are demonstrated in Figure 1.2. The bandwidth values of four generations of DDR memories, i.e., the memory standard for CPUs, are also depicted as reference. The latest NVIDIA and AMD GPUs have a peak memory bandwidth of 208 GB/s and 264 GB/s, respectively, while the current DDR3 memory standard only supports 17GB/s (the next generation DDR4 will double the bandwidth to 34GB/s) [122]. Certainly the superior memory bandwidth of GPUs will significantly benefit memory-intensive EDA applications.

Traditionally, GPUs are programmed with shading languages like OpenGL [191]. Although OpenGL can be used for general-purpose computing on GPUs (GPGPU), the resultant programming process is laborious and error-prone. To ease the programming effort of GPGPU, NVIDIA introduced the Compute Unified Device Architecture (CUDA) technology [178, 183] so that programmers can develop GPGPU programs in a C/C++ alike language with a few extensions. While CUDA
can only be used on NVIDIA GPUs, OpenCL is defined by a group of industry players as a standard cross-platform GPGPU language [134].

The synergy of GPU hardware and software has resulted in successful applications in a diverse range of scientific and engineering domains [28, 195]. On workloads with appropriate computing and memory accessing patterns, GPU can even attain a speed-up of over 100X. It is thus appealing to unleash the computing power of GPU for EDA applications.

Different CPUs, GPUs adopt a fine-grain multithreading model. Equipped with dedicated hardware for context switching, GPU threads are light-weighted and excel in massively data-parallel processing. Such an execution model makes GPU proper for EDA applications featuring data-parallelism. There is already a large body of literature presenting encouraging results on utilizing GPU to solve various EDA problems. GPGPU proved to be effective in such time consuming applications as system level design, logic simulation, timing analysis, power grid analysis, placement, and routing. The positive results suggest that the superior computing power of GPUs can be unleashed by developing carefully designed data-parallel algorithms and highly tuned implementations.

On the other hand, EDA applications pose unique challenges to the GPGPU model. The nature of circuits determines that the underlying data structures capturing IC designs tend to be irregular. Typical EDA applications are thus constructed on the basis of such irregular data structures as sparse matrix, tree, and graph. The resultant memory accessing patterns are less amenable to GPUs, which only have a limited capacity of cache and assume regular memory accesses to fully utilize its large memory bandwidth. Accordingly, current works on GPU-based EDA computing generally resort to two strategies: (1) identifying regular sub-problems in an EDA application and then use GPU as an accelerator for them; and (2) re-designing or re-structuring algorithms on GPU so as to convert irregular data accesses into (at least partially) regular ones. Another challenge is that EDA applications are

---

1 There exist special cases where the data structure can be quite regular. One such typical example is the power distribution network, which in many designs consists of a relatively regular power mesh.
extremely complex and cover many different domains of computations. Accordingly, an application-by-application parallelization approach can be infeasible. A viable line of attack, instead, is to identify the fundamental computing patterns and perform parallelization on them. Such a pattern-based strategy of parallel programming proves to be crucial for many other software applications [162].

In this monograph, we present an up-to-date survey on the progresses in GPU-accelerated EDA computing. Considering the high complexity of EDA applications, an essential objective of this work is to extract key computing patterns of EDA and present state-of-the-art GPU programming techniques to resolve such patterns. We believe that this approach will substantially ease the deployment of GPUs in future EDA software. This monograph focuses on using GPU to accelerate applications in the EDA domain, while the techniques also have wide applications in many other scientific and engineering domains. Interested readers please also refer to [Owens et al. 2007; Refs. 28 [195]] for surveys on applications in other disciplines.

The remainder of this monograph is organized as follows. Section 2 provides an overview of GPU hardware architectures and the corresponding data-parallel programming model. In Sections 3 and 4, we develop a taxonomy for the basic computing patterns of EDA applications and then review relevant GPU programming techniques for these patterns. In Section 5, we survey successful applications of GPU-accelerated EDA computing. In Section 6, we conclude this work and propose future research directions.
References


References


References


References


References


References


References


Full text available at: http://dx.doi.org/10.1561/1000000028
References


References

References

NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf, 2009.
[187] NVIDIA, “NVIDIA’s next generation CUDA® compute architecture: Kepler™ GK110.” (http://www.nvidia.com/content/PDF/kepler/NVIDIA-


References


References


