INTEGRATION, the VLSI journal 45 (2012) 46–60

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

Towards accelerating irregular EDA applications with GPUs

Hao Qian *, Yangdong Deng, Bo Wang, Shuai Mu

Institute of Microelectronics, Tsinghua University, Beijing, China

ARTICLE INFO

Article history:
Received 5 October 2010
Received in revised form
1 April 2011
Accepted 27 May 2011
Available online 23 June 2011

Keywords:
GPU
CUDA
Sparse matrix
EDA
Data parallel
Static timing analysis
Sparse matrix vector product
Conjugate gradient
Breadth first search
Survey propagation
RTL simulation
CMB
Message-passing

ABSTRACT

Recently graphic processing units (GPUs) are rising as a new vehicle for high-performance, general purpose computing. It is attractive to unleash the power of GPU for Electronic Design Automation (EDA) computations to cut the design turn-around time of VLSI systems. EDA algorithms, however, generally depend on irregular data structures such as sparse matrix and graphs, which pose major challenges for efficient GPU implementations. In this paper, we propose high-performance GPU implementations for a set of important irregular EDA computing patterns including sparse matrix, graph algorithms and message-passing algorithms. In the sparse matrix domain, we solve a core problem, sparse-matrix vector product (SMVP). On a wide range of EDA problem instances, our SMVP implementation outperforms all prior work and achieves a speedup up to 50 × over the CPU baseline implementation. The GPU based SMVP procedure is applied to successfully accelerate two core EDA computing engines, timing analysis and linear system solution. In the graph algorithm domain, we developed a SMVP based formulation to efficiently solve the breadth-first search (BFS) problem on GPUs. We also developed efficient solutions for two message-passing algorithms, survey propagation (SP) based SAT solution and a register-transfer level (RTL) simulation. Our results prove that GPUs have a strong potential to accelerate EDA computing through designing GPU-friendly algorithms and/or re-organizing computing structures of sequential algorithms.

1. Introduction

Integrated circuits (ICs) have become the most complex machine made by the human being. Today IC designers depend on the Electronic Design Automation (EDA) software to properly handle the ever-increasing IC complexity in a timely fashion. The computing demand for EDA software is still fast rising with the advent of 32 nm technology node. For instance, IC designers have to spend a couple of hours to perform a timing analysis on a 10 M-gate design, while a gate level simulation of a full chip could take weeks or even months. Another example is the circuit simulation problem. Given a Giga-Hertz phase-lock loop (PLL) circuit, a transient analysis would need to simulate the circuit for millions of cycles and thus a complete run would take months to finish. However, today's electronic appliances typically have a fixed market window as short as 6 months [1]. Due to the prohibitive CPU time for design implementation and verification, such a tight schedule suggests that only a small portion of the complete solution space can be explored in the design process if the productivity of EDA software cannot scale accordingly. On the other hand, the IC product development cost can reach $100M at 32 nm technology node [2]. Such an overwhelming cost suggests that IC designers have to perform even more intensive verification to minimize the possibility of buggy designs. As a result, future EDA software tools have to deliver even higher computing throughput.

In the history of EDA technology, the constantly increasing processing capability mainly came from the synergy of two forces: (1) development of smarter algorithms and/or more efficient software implementations and (2) scaling of CPU performance. Unfortunately, now the single-CPU performance is relatively saturating. Since the semiconductor process is still offering growing integration capacity, multi-core processors are inevitably becoming the dominant computing resources for EDA applications. It is thus essential to develop parallel solutions for the EDA industry such that the momentum of function increase in VLSI designs can be maintained [3].

Recently, general purpose computing on graphic processing units (GPUs) has become a very important trend of high performance computing [4]. Unlike multi-core CPUs that generally utilized a task level parallelism, graphic processing units (GPUs) exploit a data parallel programming model. Upon receiving a workload, a GPU would launch tens of thousands of fine-grain threads concurrently, with each thread executing the same program but on a different data set. Modern GPUs could deliver

* Corresponding author.
E-mail address: cyqh1028@hotmail.com (H. Qian).

0167-9260/$ - see front matter Crown Copyright © 2011 Published by Elsevier B.V. All rights reserved.
doi:10.1016/j.vlsi.2011.05.004

a very high computing throughput. For example, NVIDIA's flagship GPU, Fermi, could reach a peak floating-point throughput of 1.5 TFLOPS [5,6]. On workloads with appropriate computing and memory access patterns, GPU could even attain a speedup of over 100 ×.

Accordingly, it is appealing to unleash the computing power of GPU for EDA applications. There are already a few papers, e.g., [7,8], and [9], presenting encouraging results on utilizing GPU to solve specific EDA problems. However, a comprehensive investigation on the foundation of GPU-based EDA computing is still critical, especially because EDA applications mainly depend on irregular data structures that are less amenable to GPUs.

The irregular data access patterns are determined by the very nature of VLSI circuits. In a typical gate level netlist, while most gates would only connect to a small but non-fixed number of neighboring cells, certain gates could have hundreds of fan-outs. Hence, the resultant data structures encoding the netlist have to be irregular. One example is the connection matrix required by the quadratic and force-driven placement algorithms, (e.g., [10] and [11]). Based on our experiments on ISPD2006 benchmark circuits [12], such matrices are extremely sparse, where most rows only have 3–5 non-zeros and a very small number of rows having hundreds or thousands of non-zeros. Major irregular EDA computing patterns include sparse matrix manipulations and graph algorithms. In fact, the authors of [3] identified major EDA applications and surveyed the underlying computing patterns. Out of the 17 major EDA applications surveyed in [3], 15 applications are built on top of graph algorithms and 4 applications involved sparse matrix computations.

Although it has long been known that sparse matrix operation and graph algorithms possess rich data level parallelism [13], it is extremely challenging to efficiently implement them on GPUs. For example, the computation of sparse-matrix vector product (SMVP) has been widely considered as one tough problem for GPUs and only marginal speedup can be accomplished until recently. In [14], Bell and Garland introduced a novel solution on NVIDIA GPUs for the SMVP problem. Their GPU implementation could attain a throughput of ~10 GFLOPS on problem instances from different engineering domains where relatively long strip of non-zeros exist. However, our experiments using the code released with [15] indicate that the speedup is still limited on the problem instances from EDA applications. As the second example, the GPU implementations for the breadth-first search (BFS) problem proposed in [17] could only achieve a good speedup on randomly created graphs where the number of edges on a node is well bounded. For graphs extracted from real-world applications, the GPU implementations in [17] do not have much advantage over their CPU equivalents. The inefficiency for the above two domains is largely due to GPU's design philosophy, which is to devote most die area on computing resources but little on caches. For irregular applications where the memory access patterns are unpredictable, GPUs would have difficulty to hide memory latency. Another major hurdle for high performance on GPUs is the poor load balance induced by the irregularity. When performing parallel operations on sparse matrices and graphs, the workload for each basic unit of parallel execution varies dramatically. However, GPU executes computation in a batched manner, where a group of threads would have exactly the same instruction schedule. Therefore, the slowest thread would determine the execution time for this group of threads.

As a first step toward a systematic parallelization of EDA applications and based upon our preliminary work [18], we explore efficient GPU solutions for two typical computing patterns, sparse matrix and graph algorithmic operations. First, we developed an efficient GPU implementation for the sparse matrix vector product (SMVP) problem, which is the central piece of sparse matrix operations. On a wide range of EDA problem instances, the GPU based SMVP implementation could outperform all previously published results and achieve a speedup of up to **50 ×**. We then applied the SMVP procedure to expedite two important EDA computing patterns, circuit delay calculation and conjugate gradient based linear system solution. A speed-up factor of one order of magnitude is observed on both problems. Next we extended our work to the graph algorithm domain by solving the classical breadth-first graph traversal problem through a SMVP based formulation, which is more aligned to the data parallel model of GPUs. In addition, the techniques developed for SMVP can also be used to efficiently accelerate a survey propagation (SP) based SAT solver [19] by a factor of around **20 ×**. We also develop GPU solutions to accelerate another key EDA application, register transfer level (RTL) simulation. Our GPU based RTL simulator can be faster by its CPU counterpart by a factor of over 18. The most important message delivered in this work is that the computing power of GPU can be successfully unleashed through properly re-organizing sequential computing structures and/or re-designing data parallel algorithms.

The remaining of this paper is organized as follows. In Section 2, we review the typical irregular computing patterns found in EDA applications. Next, the hardware architecture of NVIDIA GPUs and the corresponding data parallel programming model are presented in Section 3. In Section 4, we propose efficient techniques to solve the sparse matrix vector product problem on GPUs. The performance of the GPU implementations is also compared with their CPU equivalents. Section 5 discusses how to apply our GPU based techniques to accelerate a series of EDA applications such as static timing analysis and the solution of linear systems for circuit placement. In Section 6 we explain how the sparse-matrix vector product pattern and its underlying techniques can be used to accelerate graph algorithm problem, breadth-first graph traversal. In Section 7, we introduce our GPU based message-passing solutions for a survey propagation based satisfiability (SAT) problem solver and a GPU-accelerated RTL simulator. Finally, we conclude the paper and outline future research directions.

2. Irregular data structures of EDA applications

Many scientific and engineering applications are based on regular data structures such as linear arrays and matrices (i.e., dense matrices). The access of data generally follows a predictable pattern in such data structures. For instance, when computing the product of a matrix and a vector, the data accessing pattern is fixed as long as the dimension of the matrix is known. In addition, a good level of load balance is relatively easy to achieve on a parallel computer. In the matrix vector product example, the workload can be evenly distributed if the computation of one matrix row and the vector is assigned to a single processing element (e.g., a CPU core). Hence, each processing element would need to complete exactly the same amount of job. By taking advantage of the predictability and good load balance, the regular data structures are relatively amenable to parallel computers.

On the other hand, EDA applications are usually based on irregular data structures such as sparse matrix and graphs. The irregularity is determined by the very nature of IC circuit structure. Fig. 1(a) illustrates a simple but commonly found circuit pattern. The circuit can be stored in an EDA database as a graph like the one illustrated in Fig. 1(b). In such a graph, every signal is treated as a node and two nodes share an edge if they are connected through a gate. Such a graph is obviously irregular because the number of edges connected to one node is not uniform across the graph. In addition, the graph is also very sparse in the sense that most nodes only have a small number of edges connecting to neighboring nodes, although a few high-fanout