Late Breaking Results: Distributed Timing Analysis at Scale

Tsung-Wei Huang
ECE Dept, UIUC, IL
twh760812@gmail.com

Chun-Xun Lin
ECE Dept, UIUC, IL
clin99@illinois.edu

Martin D. F. Wong
ECE Dept, UIUC, IL
mdfwong@illinois.edu

ABSTRACT

As the design complexities continue to grow, the need to efficiently analyze circuit timing with billions of transistors is quickly becoming the major bottleneck to the overall chip design flow. In this work we introduce a distributed timer that (1) has scalable performance, (2) can be seamless integrable to existing EDA applications, (3) enables transparent resource management, (4) has robust fault-tolerant control. We evaluate the distributed timer using a set of large industry benchmarks on a cluster with 24 nodes. The results show that the proposed timer achieves full accuracy over all designs with high performance and good scalability.

ACM Reference Format:
Tsung-Wei Huang, Chun-Xun Lin, and Martin D. F. Wong. 2019. Late Breaking Results: Distributed Timing Analysis at Scale. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3316781.3322470

1 PROPOSED DISTRIBUTED TIMER

Our distributed timer supports fundamental timing routines which are typically categorized to *design transforms* and *timing queries* [8, 9]. Design transforms refer to operations that are applied to modify the design, including pin-level modifiers (`disconnect_pin`, `connect_pin`), net-level modifiers (`remove_net`, `insert_net`), and gate-level modifiers (`insert_gate`, `repower_gate`, `remove_gate`). Timing queries let users probe the design to report timing information (`report_slack`, `report_at`, `report_rat`, `report_worst_paths`). In order to efficiently collaborate with optimization programs, the timer needs to deal with these operations in an interactive and incremental (online) manner.

The distributed timer is built on top of the state-of-the-art open-source software, OpenTimer [1, 9]. OpenTimer is a high-performance timing analysis tool that aims for a single machine node with multi-threading enabled through task parallelism [2]. It provides practical supports such as industry data formats (.v, .spef, .lib, etc.), both block-based and path-based timing, unwanted path pessimism removal, and parallel incremental processing. The central engine of OpenTimer is a pipeline scheduler, from which multiple and different timing operations can be efficiently executed. On this basis, The distributed timer is designed to be at framework level, rather than forcing a rewrite of numbers of codes that have already been written for OpenTimer. Figure 1 presents the software architecture of the distributed timer.

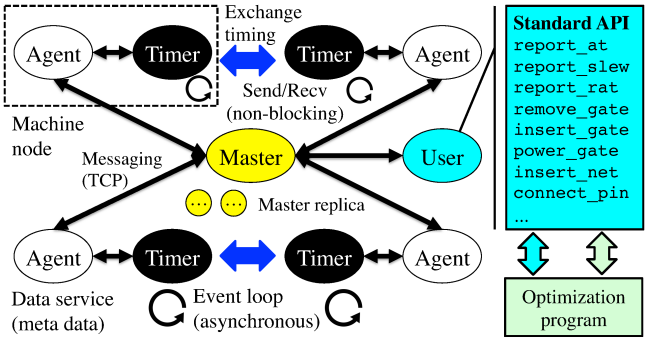


Figure 1: Software architecture of the proposed distributed timer.

Our design philosophy is to provide users a centralized viewpoint of the entire distributed system. The distributed timer consists of a *master* daemon that manages *agent* daemons running on each cluster node, and *timer instances* that are coordinated by these agents [11] [3] [4]. A timer instance is a local timer that runs on a single design partition and *persists* in memory through an event-driven loop. Master is the major gateway to external users, through which operations are submitted and results are collected. For each operation, master decides and delivers the tasks to agents, which in turn request local timers to derive the results and return them back to the master. Agents are responsible for all boundary timing data transmission among multiple timer instances. To avoid single point of failure, we apply the state machine replication for master [10].

2 EXPERIMENTAL RESULTS

Our distributed timer is written in C++14 language and compiled with GCC 6.0 on a Linux machine. We have approximately written thousands of lines of code (70% on the networking library and 30% on distributed timing). The evaluation is taken on a small cluster of 24 nodes with each node configured with a single Intel i5 3.2 GHz CPU and 1 GB memory. We select four different sets of tens of design partitions from industry benchmarks that have been released to the public domain by ACM TAU 2014–2016 Timing Analysis Contests [5–7]. Each design partition has tens of thousands of gates and is connected to each other through primary inputs and primary outputs. The benchmark consists of a set of design modifiers and timing queries as well as a golden reference generated on flattened designs (no partitions).

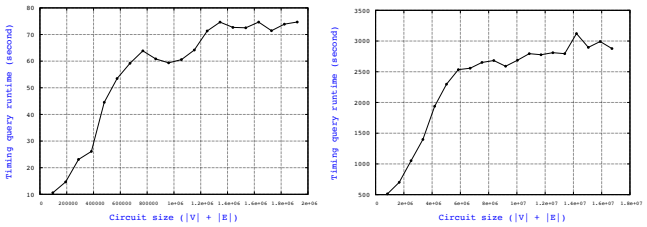
DAC '19, June 2–6, 2019, Las Vegas, NV, USA

ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3322470

Table 1: Overall experimental results

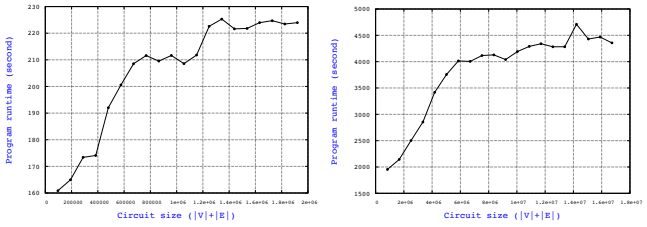
Benchmark	$\|V\|$	$\|E\|$	$\|B\|$	$\|P\|$	Accu	CPU
Design_1	122.4K	124.8K	4.9K	12	100%	2.0s
Design_2	680.7K	688.0K	10.5K	14	100%	3.9s
ac97_ctrl	848.7K	1.1M	2.6K	20	100%	3.7m
des_perf	7.4M	9.2M	7.4K	20	100%	1.2h

Table 1 demonstrates the benchmark statistics and our overall experimental results. The notation $|V|$ ($|E|$) denotes the total number of nodes (edges) in a flattened view. The notation $|B|$ denotes the total number of boundary pins (primary inputs and primary outputs) that are connected between agents and master. The notation "Accu" denotes the accuracy of our distributed timing report. Design_1 and Design_2 are small benchmarks while ac97_ctrl and des_perf contain millions of pins in the circuit graphs (up to 7,431,740 pins for des_perf). Notice that we do not compare our distributed timing with the single-machine timing since timing the flattened design on a single machine fails to fit into the main memory [12]. Roughly speaking, Our distributed timer achieves full accuracy on the timing report compared to the golden reference. This has shown that the message passing between agents and master is handled correctly.



(a) Runtime of ac97_ctrl on timing queries in terms of different circuit sizes. **(b) Runtime of des_perf on timing queries in terms of different circuit sizes.**

Figure 2: Total runtime on timing queries



(a) Total program runtime of ac97_ctrl in terms of different circuit sizes. **(b) Total program runtime of des_perf in terms of different circuit sizes.**

Figure 3: Total program runtime

The overall performance of the distributed timer is plotted in Figures 3a - 3b. Since ac97_ctrl and des_perf are relatively larger than the others, we conduct the experiments on these two benchmarks to demonstrate the performance. ac97_ctrl and des_per are each with twenty partitions. We totally run about 14,000 operations for ac97_ctrl and 140,000 operations for des_per. These operations consist of various design modifiers and timing queries. Because of the laziness, the runtime on design modifier operations is negligible compared to timing queries. We only draw the runtime on timing queries and the total program runtime (timing queries, design modifiers, communication).

Figure 2a and Figure 2b show the runtime on timing query operations of ac97_ctrl and des_perf in terms of different circuit sizes. In general, the runtime increases as the circuit size grows. Some sharp growth is observed at, for example, circuit sizes 383,984 and 1,151,952 in ac97_ctrl. The reason is that we place more partitions than the available number of cores on a machine node due to the hardware limitation. Otherwise, the runtime curve grows slowly as the circuit sizes increase. We next discuss the total program runtime of ac97_ctrl and des_perf. As shown in Figures 3a and 3b, the total runtime curve follows a similar trend as that of timing queries. This is expected since timing queries are more expensive than design modifiers in terms of computation and communication. On average, the rate of runtime growth per unit increase of the circuit size is about 0.03 milliseconds for ac97_ctrl and 0.15 milliseconds for des_perf, respectively. Even though such evidence might be hardware-dependent, it quantifies the scalability of our approach.

In summary, these experimental results have demonstrated the correctness of our distributed timing analysis implementation. Our distributed timer can scale very well (almost linearly) as the circuit size grows and can analyze the timing on very large designs (up to 7 millions vertices and 9 millions edges).

3 ACKNOWLEDGMENT

This work is supported by NSF Grant CCF-1718883 and DARPA Grant FA-650-18-2-7843.

REFERENCES

[1] OpenTimer, https://github.com/OpenTimer/OpenTimer

[2] T.-W. Huang, C.-X Lin, Guannan Guo and M. D. F. Wong, "Cpp-Taskflow: Fast Task-based Parallel Programming using Modern C++," in *Proc. IEEE IPDPS*, 2019.

[3] T.-W. Huang, C.-X Lin, Guannan Guo and M. D. F. Wong, "A General-purpose Distributed Programming System using Data-parallel Streams", in *ACM MM*, 2018.

[4] T.-W. Huang, C.-X Lin and M. D. F. Wong, "DtCraft: A High-performance Distributed Execution Engine at Scale", in *IEEE TCAD*, 2018.

[5] J. Hu and D. Sinha and I. Keller, "TAU 2014 contest on removing common path pessimism during timing analysis: Special session paper: Common path pessimism removal (CPPR)," in *Proc. IEEE/ACM ICCAD*, 2014, pp. 591–591.

[6] J. Hu and G. Schaeffer and V. Garg, "TAU 2015 contest on incremental timing analysis," in *Proc. IEEE/ACM ICCAD*, 2015, pp. 882–889.

[7] http://www.tauworkshop.com/2016/contest.html

[8] J. Bhasker and R. Chadha, "Static Timing Analysis for Nanometer Designs: A Practical Approach," *Springer*, 2009.

[9] T.-W. Huang and M. D. F. Wong, "OpenTimer: A high-performance timing analysis tool," in *Proc. IEEE/ACM ICCAD*, 2015, pp. 895–902.

[10] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed, "ZooKeeper: wait-free coordination for internet-scale systems," *Proc. USENIX*, 2010

[11] T.-W. Huang and C.-X Lin and M. D. F. Wong, "DtCraft: A distributed execution engine for compute-intensive applications," *Proc. ACM/IEEE ICCAD*, 2017

[12] T.-W. Huang, Martin D. F. Wong, D. Sinha, K. Kalafala, and N. Venkateswaran "A Distributed Timing Analysis Framework for Large Designs," *Proc. IEEE/ACM DAC*, 2016