# A General-purpose Parallel and Heterogeneous Task Programming System for VLSI CAD

Invited Talk

Tsung-Wei Huang
Department of ECE
University of Utah
twh760812@gmail.com

## ABSTRACT

This paper introduces *Taskflow* to address the critical question of *"How can we make it easier to implement and deploy parallel computer-aided design (CAD) algorithms on large heterogeneous nodes with high performance and simultaneous high productivity?"* Parallelizing CAD is an extremely challenging job. Modern CAD applications exhibit unique computational patterns and user requirements that need very strategic decomposition to benefit from parallelism. Taskflow assists researchers and developers in the implementation complexity of parallel algorithms by introducing a new high-level programming model supported by an efficient runtime. By capitalizing on emerging parallelism comprising manycore central processing units (CPUs), graphics processing units (GPUs), and custom accelerators, Taskflow enables CAD to achieve new performance and productivity milestones that were previously out of reach.

## KEYWORDS

Parallel programming, computer-aided design

**ACM Reference Format:**
Tsung-Wei Huang, Department of ECE, University of Utah, twh760812@gmail.com . 2020. A General-purpose Parallel and Heterogeneous Task Programming System for VLSI CAD: Invited Talk. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '20), November 2–5, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3400302.3415750

## 1 INTRODUCTION

The ever-increasing design complexity in very-large-scale integration (VLSI) implementation will soon far exceed what many existing computer-aided design (CAD) tools are able to scale with reasonable design time and effort (see Figure 1). A key fundamental challenge is that CAD must incorporate *new parallel paradigms* comprising manycore central processing units (CPUs), graphics processing units (GPUs), and custom accelerators to allow more efficient design space exploration and optimization [1–3]. However, parallelizing CAD is an extremely challenging job. Modern CAD applications exhibit unique computational patterns and user requirements that need very strategic decomposition to benefit from parallelism [4, 5]. For example, timing analysis makes essential use of *dynamic control flow* to implement various computational patterns across *millions to billions dependent tasks* [6, 7]. It is too difficult to achieve transformational performance milestones without the aid of high-level programming models and system runtimes that target the unique parallelization challenges of CAD. This type of system innovation has profound impacts on the CAD community because it *complements* the current state-of-the-art by assisting everyone to tackle the challenges of implementing and deploying parallel CAD algorithms. Unfortunately, related system research has received very little attention in the CAD community.

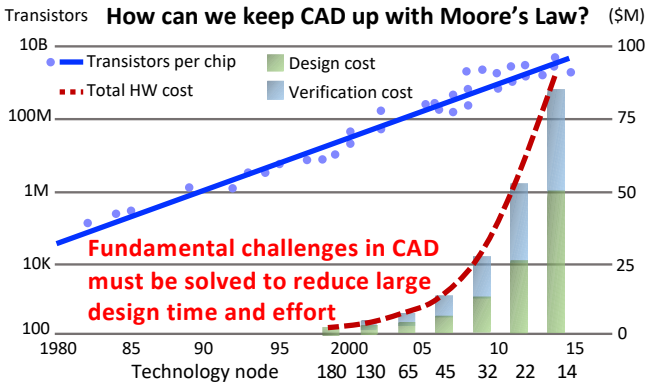


**Figure 1: Design cost versus design complexity [8].**

Over the past years, we have invested a lot of research and development (R&D) effort in existing programming systems from the scientific computing community, including pthread, OpenMP task [9], TPL [10], Cilk [11], StarPU [12], HPX [13], PaRSEC [14], Kokkos [15], XKAAPI [16], and Charm++ [17]. However, almost all existing programming systems fail to effectively parallelize CAD because they were not fundamentally designed with CAD's unique computational patterns and user requirements in mind, which we explain as follows:

- **Complex control flow.** Optimization-driven workloads such as logic synthesis, placement, and routing make essential use of


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

ACM ISBN 978-1-4503-8026-3/20/11…$15.00
https://doi.org/10.1145/3400302.3415750

*dynamic control flow* to implement various combinatorial and analytical algorithms that incorporate conditional, cyclic, and non-deterministic computational patterns. Existing task programming frameworks such as OpenMP [9], Kokkos [15], TBB [18], and PaRSEC [14] closely rely on *directed acyclic graph* (DAG) models to define tasks and their dependencies. Users implement control-flow decisions *outside* the graph description via either statically unrolling the graph across fixed-length iterations or resorting to client-side decisions. This organization typically results in rather complicated procedure that lack *end-to-end* parallelism.

- **Complex task dependencies.** Analysis-driven workloads such as timing and power analysis require computations to propagate through the circuit network. Different quantities (e.g., slew, arrival time) are often dependent on each other, either logical relation or physical net order, and are expensive to compute. The resulting task graph in terms of encapsulated function calls and task dependencies is typically very large. During incremental analysis, data can arrive sparsely or densely and can run on CPUs, GPUs, or more frequently a *mix*. Existing frameworks such as task functors [15], templates [18], and C-styled pointers [12] are restrictive to fixed-memory layout. There are no easy ways to describe up-front parallelism for an incremental loop where the circuit graph structure and task dependencies keep changing.

After years of research, we have arrived at a key conclusion: While designing parallel CAD algorithms is non-trivial, what makes parallelizing CAD an enormous challenge is the infrastructure work of *"efficiently expressing dependent tasks along with algorithmic control flow and scheduling them across heterogeneous computing resources."*

## 2 TUTORIAL GOAL

This tutorial paper presents *Taskflow*, a general-purpose parallel task programming system, we have been developing for years to streamline the building of parallel CAD tools. By capitalizing on emerging parallelism comprising manycore CPUs, GPUs, and custom accelerators, Taskflow enables CAD to achieve new performance and productivity milestones that were previously out of reach. Specifically, we cover three topics:

- **Programming model.** We present the novel powerful heterogeneous programming model inspired by the parallelization challenges of CAD such that developers can efficiently express a wide range of CAD workloads using minimal programming effort.
- **System runtime.** We present the efficient system runtime (i.e., task execution environment) to support our programming models with high performance. Our runtime solves many of the new scheduling challenges arising out of our models and optimize the system performance across latency, energy efficiency, and throughput.
- **Application.** We present two applications, static timing analysis and detailed placement, to which we have applied Taskflow. We show that when Taskflow is leveraged to implement parallel CAD algorithms, many implementation challenges can be largely mitigated, and new parallel CAD algorithms and frameworks can proliferate.

Taskflow is open-source at [19], including step-by-step tutorials, application programming interface (API) reference, and benchmarks. We are actively developing and maintaining Taskflow. Since its first release in 2018, we have accumulated more than 500K downloads and helped many research and academic projects (including our timing research [20, 21]) improve their performance through parallelism. We have demonstrated the promising performance of Taskflow in various CAD applications and scientific computing workloads [7, 22–24].

## REFERENCES


[1] Andrew B. Kahng. Reducing Time and Effort in IC Implementation: A Roadmap of Challenges and Solutions. In *IEEE/ACM DAC*, 2018.
[2] Yi-Shan Lu and Keshav Pingali. Can Parallel Programming Revolutionize EDA Tools? *Advanced Logic Synthesis*, 2018.
[3] Leon Stok. The Next 25 Years in EDA: A Cloudy Future? *IEEE Design Test*, 31(2):40–46, 2014.
[4] A. Ng and I. L. Markov. Toward quality EDA tools and tool flows through high-performance computing. In *ISQED*, pages 22–27, 2005.
[5] B. Catanzaro, K. Keutzer, and Bor-Yiing Su. Parallelizing CAD: A timely research agenda for EDA. In *ACM/IEEE DAC*, pages 12–17, 2008.
[6] Tsung-Wei Huang and Martin D. F. Wong. OpenTimer: A high-performance timing analysis tool. In *IEEE/ACM ICCAD*, pages 895–902, 2015.
[7] Tsung-Wei Huang, Chun-Xun Lin, Guannan Guo, and Martin Wong. Cpp-Taskflow: Fast Task-based Parallel Programming using Modern C++. In *IEEE IPDPS*, pages 974–983, 2019.
[8] DARPA: Intelligent Design of Electronic Assets (IDEA). https://www.darpa.mil/program/intelligent-design-of-electronic-assets.
[9] E. Ayguade, N. Copty, A. Duran, J. Hoeflinger, Y. Lin, F. Massaioli, X. Teruel, P. Unnikrishnan, and G. Zhang. The Design of OpenMP Tasks. *TPDS*, 20(3):404–418, 2009.
[10] Daan Leijen, Wolfram Schulte, and Sebastian Burckhardt. The Design of a Task Parallel Library. In *ACM OOPSLA*, pages 227–241, 2009.
[11] Charles E. Leiserson. The Cilk++ concurrency platform. *The Journal of Supercomputing*, 51(3):244–257, 2010.
[12] Cédric Augonnet, Samuel Thibault, Raymond Namyst, and Pierre-André; Wacrenier. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurr. Comput. : Pract. Exper.*, 23(2):187–198, 2011.
[13] Hartmut Kaiser, Thomas Heller, Bryce Adelstein-Lelbach, Adrian Serio, and Dietmar Fey. HPX: A Task Based Programming Model in a Global Address Space. In *PGAS*, pages 6:1–6:11, 2014.
[14] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J. J. Dongarra. PaRSEC: Exploiting Heterogeneity to Enhance Scalability. *Computing in Science Engineering*, 15(6):36–45, 2013.
[15] H. Carter Edwards, Christian R. Trott, and Daniel Sunderland. Kokkos: Enabling manycore performance portability through polymorphic memory access patterns. *Journal of Parallel and Distributed Computing*, 74(12):3202 – 3216, 2014.
[16] T. Gautier, J. V. F. Lima, N. Maillard, and B. Raffin. XKaapi: A Runtime System for Data-Flow Task Programming on Heterogeneous Architectures. In *IEEE IPDPS*, pages 1299–1308, 2013.
[17] Laxmikant V. Kale and Sanjeev Krishnan. Charm++: A Portable Concurrent Object Oriented System Based on C++. In *ACM ASPLOS*, page 91–108, New York, NY, USA, 1993.
[18] Intel oneTBB. https://github.com/oneapi-src/oneTBB.
[19] Taskflow. https://github.com/taskflow/taskflow.
[20] Tsung-Wei Huang, Guannan Guo, Chun-Xun Lin, and Martin Wong. OpenTimer 2.0: A New Parallel Incremental Timing Analysis Engine. *IEEE TCAD*, 2020.
[21] Zizheng Guo, Tsung-Wei Huang, and Yibo Lin. GPU-accelerated Static Timing Analysis. In *IEEE/ACM ICCAD*, pages 1–8, 2020.
[22] Chun-Xun Lin, Tsung-Wei Huang, Guannan Guo, and Martin Wong. An Efficient and Composable Parallel Task Programming Library. In *IEEE HPEC*, pages 1–7, 2019.
[23] Chun-Xun Lin, Tsung-Wei Huang, Guannan Guo, and Martin Wong. A Modern C++ Parallel Task Programming Library. In *ACM Multimedia Conference*, pages 2285–2287, 2019.
[24] Guannan Guo, Tsung-Wei Huang, Chun-Xun Lin, and Martin Wong. An Efficient Critical Path Generation Algorithm Considering Extensive Path Constraints. In *ACM/IEEE DAC*, pages 1–6, 2020.