# Cpp-Taskflow: Fast Parallel Programming with Task Dependency Graphs

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

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

Guannan Guo
Dept. of ECE, UIUC
IL, USA
guannan4@gmail.com

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

## ABSTRACT

As we move to multi-core era, it's important for software developers to apply multi-threading to increase the performance of their applications. However, parallel programming is much more difficult than writing a sequential program, especially when complex parallel patterns exist in the application. In this paper, we introduce Cpp-Taskflow [1], a zero-dependency library written in modern C++17 to help programmers quickly build parallel task dependency graphs. Cpp-Taskflow has a very neat and expressive API that allows users to master multi-threading in just a few minutes. Compared with existing libraries, Cpp-Taskflow is more cost-efficient in performance scaling and software integration.

## 1 INTRODUCTION

Parallel computing is getting increasingly prevalent and important as a single chip now can accomodate multiple processing units and is no doubt a pivotal component in building high-performance software. Parallel computing is also very useful to EDA applications as the recent trend [2] [3] [4][5] shows the EDA tools gain significantly speedup by scaling to multicores. Programming a parallel application is a challenging task because of the indeterministic nature of the concurrent thread execution. The parallelism might lead to unexpected result if one does not properly control the execution flow. Therefore, compared to writing sequential code, developers need to devote more efforts to correctly implement a parallel program.

C++ is the most widely used programming language for developing EDA applications due to its high performance and it has included essential facilities to support parallel computing in the standard library [6], such as the *thread* object, *mutex*, *lock* and etc. The standard library enables the programmers to build the parallel applications from scratch but it is not scalable primarily because of the low-level thread managements. In addition to the standard library, several libraries are invented to expedite the the development of parallel program such as OpenMP [7] and Intel Thread Building Blocks (TBB) [8]. OpenMP provides users a set of predefined directives to annotate the desired parallelism and the compiler generates the parallel code based on the annotations. Intel TBB is a template library that supports many APIs for different parallel execution patterns. Although both libraries solve the low-level thread management problem of standard library, they require compiler support and library installation which causes the integration into existing projects quite inconvenient. Furthermore, both libraries are not expressive in API to describe parallel applications especially when tasks dependencies become complex.

In this paper, we present Cpp-Taskflow, a header-only library implemented in modern C++17. The main idea of Cpp-Taskflow is to let users describe the dependency among tasks as a directed graph and the tasks will be executed in parallel following the given dependency. This model is very intuitive to use and is sufficient to express most parallel execution patterns. The thread execution is automatically undertaken by Cpp-Taskflow and users only need to focus on maximizing the parallelism without wrestling with the low-level thread managements. The library is header-only which allows effortless integration into existing software. In following sections, we introduce the programming model of Cpp-Taskflow and the associated APIs for implementing a parallel program.

## 2 CPP-TASKFLOW

Cpp-Taskflow uses a graph to represent the dependency between tasks. A node in the graph encapsulates a task to be executed and an edge indicates the execution order of two tasks. A task will be executed

## 3 ACKNOWLEDGMENT

## REFERENCES

[1] Cpp-Taskflow. https://github.com/cpp-taskflow/cpp-taskflow.
[2] Wen-Hao Liu, Wei-Chun Kao, Yih-Lang Li, and Kai-Yuan Chao. Multi-threaded collision-aware global routing with bounded-length maze routing. In *Proceedings of the 47th Design Automation Conference*, DAC '10, pages 200–205, New York, NY, USA, 2010. ACM.
[3] Tsung-Wei Huang and Martin D. F. Wong. Opentimer: A high-performance timing analysis tool. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '15, pages 895–902, Piscataway, NJ, USA, 2015. IEEE Press.
[4] L. Stok. Developing parallel eda tools [the last byte]. *IEEE Design Test*, 30(1):65–66, Feb 2013.
[5] Yi-Shan Lu and Keshav Pingali. *Can Parallel Programming Revolutionize EDA Tools?*, pages 21–41. Springer International Publishing, Cham, 2018.
[6] Thread support library. https://en.cppreference.com/w/cpp/thread.
[7] OpenMP. https://www.openmp.org/.
[8] Intel Threading Building Blocks. https://www.threadingbuildingblocks.org/.