Tsung-Wei Huang's Dissertation Statement

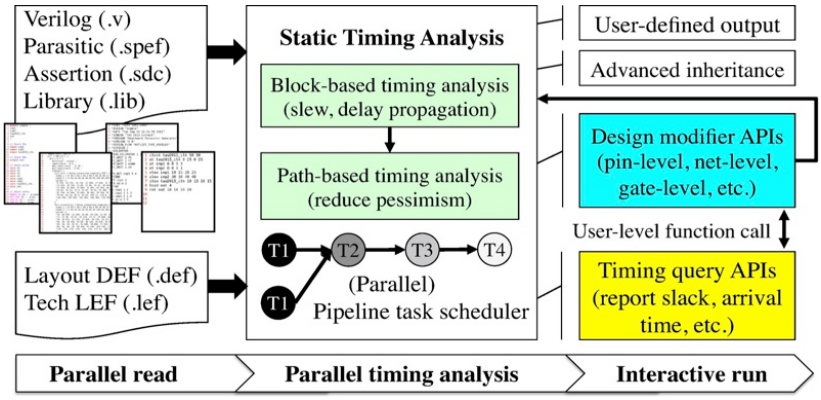
My PhD dissertation "Distributed Timing Analysis" is published in the Department of Electrical and Computer Engineering at the University of Illinois at Urbana-Champaign (UIUC) in 2017. It is a collection of my research works to solve critical challenges in Timing Analysis, Parallel Programming, and Distributed Computing. I summarize the contributions of my research as follows:

1. I developed *OpenTimer*, a high-performance static timing analysis tool for very large scale integration (VLSI) systems. OpenTimer developed efficient algorithms to facilitate timing-driven optimizations and synthesis flows that strive to deliver silicon-accurate timing signoff and signal integrity analysis.
2. I developed *DtCraft*, a general-purpose distributed programming system using data-parallel streams. The system streamlines the building of high-performance parallel and distributed applications.
3. I developed *Cpp-Taskflow*, a new task-based parallel programming library using modern C++. The programming model enables efficient implementations of parallel decomposition strategies.

All my research developments are open-source and being used by both industry and academic projects. In addition to academic values, I have invested a lot in software engineering to make my research results accessible. This not only leads to better user experience, but also helps facilitate the research community to develop derived work. Please visit http://twhuang.ece.illinois.edu for project links, papers, and awards.

OpenTimer: An Open-source High-performance Timing Analysis Tool

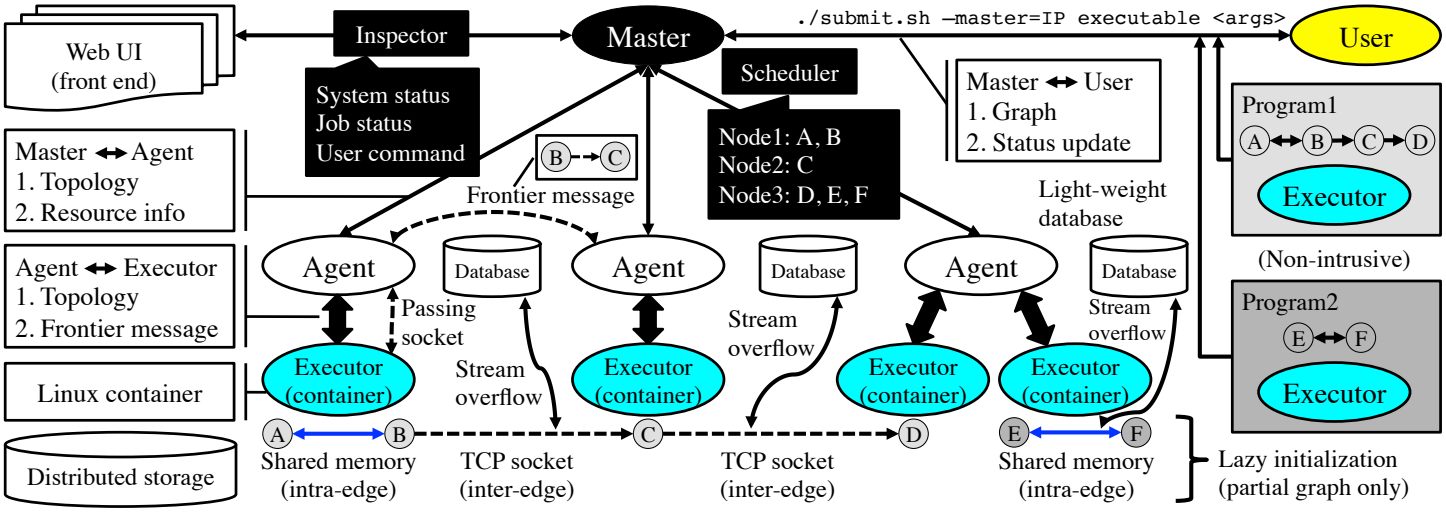
The first part of my dissertation introduced OpenTimer – an open-source static timing analysis (STA) tool that efficiently supports (1) industry standard input/output formats (.v, .lib, .spef), (2) parallel incremental timing, and (3) pessimism reduction through path-based analysis. The software architecture of OpenTimer is shown in the right figure.



One key contribution of OpenTimer is a new path-based timing analysis algorithm, where we introduced a $O(1)$ space/time path representation that can quickly peel out the critical paths. Our algorithm applies to almost all path-based analysis frameworks such as common path pessimism removal (CPPR), false-path assertions, path-specific variation modeling, and so on. Another notable result is our parallel incremental timing framework, where we introduced a fully asynchronous task model that can effectively captured the complex task dependencies in timing propagation to efficiently parallelize the incremental timing. This component is later on generalized to a standalone library called Cpp-Taskflow to help advance the parallel programming community. These promising results let OpenTimer win a number of awards including the top-3 winners at the ACM TAU Timing Analysis Contests in 2014 through 2016 and the Special Price for an Open-source EDA Tool in the 2016 LibreCore Design Contest, and a number of recognitions from the EDA community such as Golden Timers in the ICCAD CAD Contest and the ACM TAU Timing Analysis Contests.

DtCraft: A New Distributed Programming System using Data-parallel Streams

The second part of my dissertation introduced DtCraft – a general-purpose distributed programming system using data-parallel streams. Today's design complexities are too large to compute efficiently using the traditional vertical scaling method. Design problems must be decomposed into smaller pieces and distributed across machine clusters to mitigate the long runtimes. However, writing a distributed program is very difficult and requires a high-barrier amount of programming effort. As a result, we developed DtCraft, a general-purpose distributed programming system using data-parallel streams (see the figure below).



DtCraft introduced a powerful C++ programming model to streamline the building of distributed applications. Users need not to have understanding of distributed computing and can focus on high-level developments, leaving tricky system details handled by our systems. I have built a distributed timing analysis tool on top of DtCraft. The results showed 17x code reduction compared with a hard-coded MPI framework. In addition to timing, I have applied DtCraft to machine learning and graph algorithms and achieved 10x-20x speed-up over existing cluster computing frameworks including Hadoop MapReduce and Spark. I also presented a distributed integration of physical design flow using DtCraft. This new solution has achieved a promising milestone by shortening the design cycle from days or months to only a few hours. Prior to this work, no one fully understood how to integrate different components in EDA flow at cloud scale. This notable result was presented in the 2017 TAU keynote, 2017 China's Future Chip Conference, 2017 ICCAD, and 2018 ACM Multimedia Conference, where I also received the best open-source software award.

Cpp-Taskflow: Fast Parallel Task Programming using Modern C++

The third part of my dissertation introduced a new parallel programming paradigm for multicore architecture based on modern C++. While the project was initially forked to address the parallel incremental timing problem, it is now generalized to a standalone open-source project called Cpp-Taskflow. The goal of this project is to help C++ programmers quickly write parallel programs and implement efficient parallel decomposition strategies at scale. Cpp-Taskflow supports both loop-based and task-based parallelisms through a simple yet powerful task dependency graph description model (see the right code snippet). Our model empowers developers with both static and dynamic graph constructions and refinements together with robust C++ standard libraries. I have applied Cpp-Taskflow to build the core incremental timing engine of OpenTimer v2. Compared with the OpenMP-based implementation (v1), v2 achieved 2x speedup in incremental timing and 2x reduction on the software cost (see Table I). The real productivity gain is tremendous.

```
#include <taskflow/taskflow.hpp>
int main(){
  tf::Taskflow tf;
  auto [A, B, C, D] = tf.silent_emplace(
    [] () { std::cout << "Task A\n"; },
    [] () { std::cout << "Task B\n"; },
    [] () { std::cout << "Task C\n"; },
    [] () { std::cout << "Task D\n"; }
  );
  A.precede(B);  // A runs before B
  A.precede(C);  // A runs before C
  B.precede(D);  // B runs before D
  C.precede(D);  // C runs before D
  tf.wait_for_all();  // block until finish
  return 0;
}
```

The Cpp-Taskflow project is gaining increasing attention in the communities of C++, parallel programming, scientific computing, gaming, animation, etc. According to our user remarks, "*Cpp-Taskflow has the cleanest tasking API I have ever seen.*" The project recently received the best poster award from the official C++ conference for the contribution to open-source parallel programming library.

TABLE I: Software Costs of OpenTimer v1 and v2

Tool	Task Model	SLOC	Effort	Sched	Dev	Cost
v1	OpenMP 4.5	9,123	2.04	0.70	2.90	$275,287
v2	Cpp-Taskflow	4,482	0.97	0.53	1.83	$130,523

Effort: development effort estimate, person-years (COCOMO model)
Sched: largest estimated schedule of a component (COCOMO model)
Dev: estimated average number of developers (efforts / schedule)
Cost: total estimated cost to develop (average salary = $56,286/year).