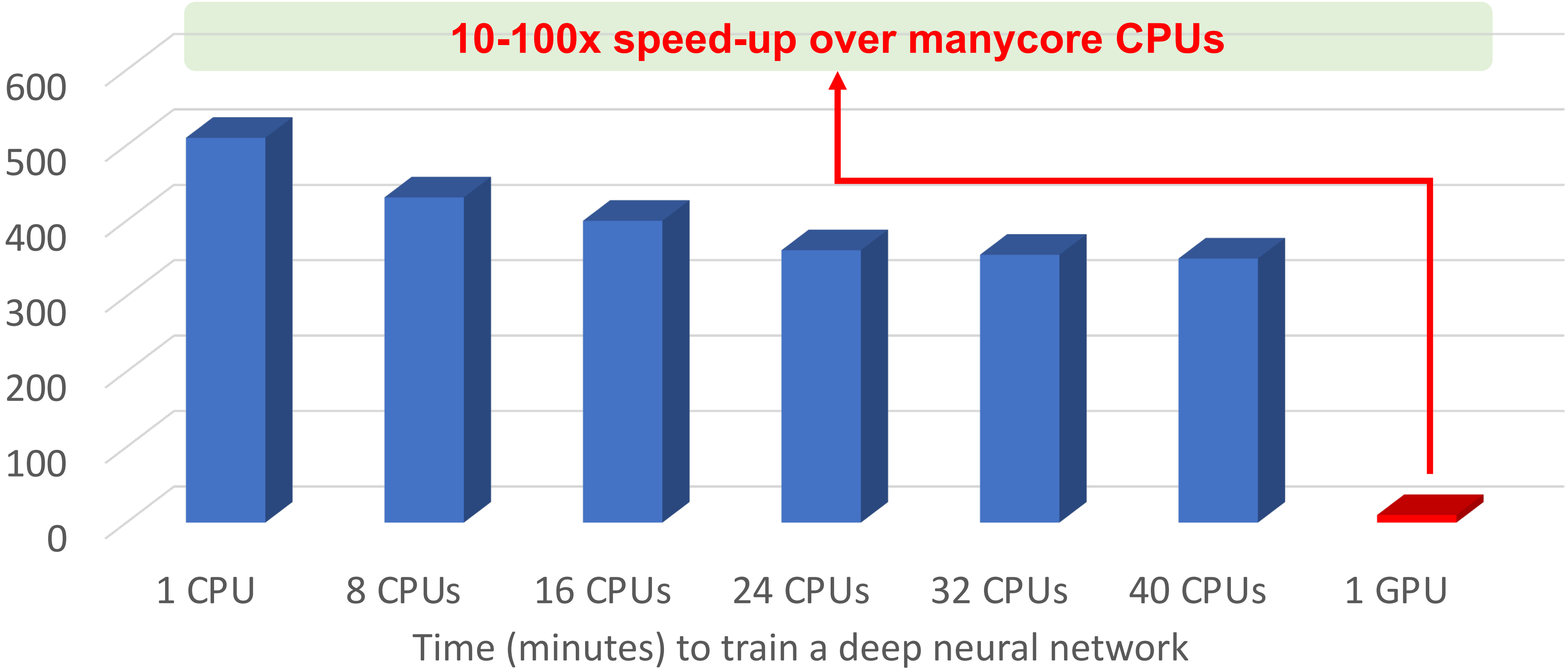
Taskflow: A General-purpose Task-parallel Programming System

Dr. Tsung-Wei (TW) Huang, Assistant Professor
Department of Electrical and Computer Engineering
University of Utah, Salt Lake City, UT
https://tsung-wei-huang.github.io/

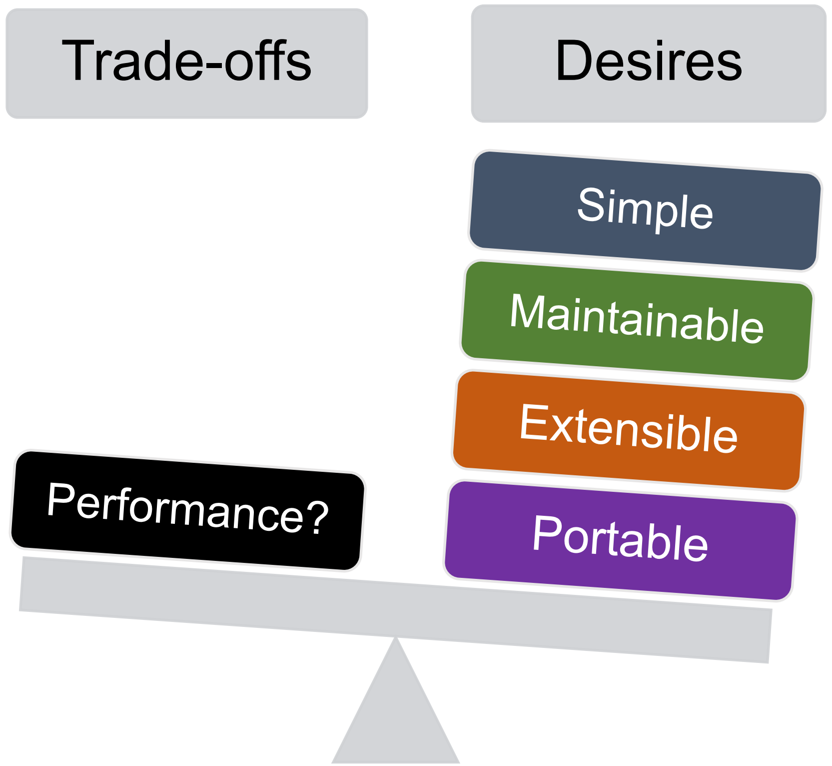
Why Parallel Computing?

- Advances performance to a new level previously out of reach



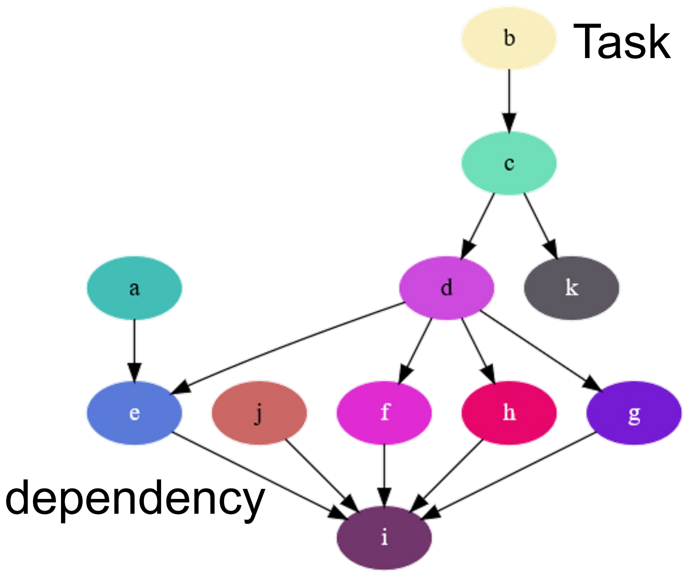
Parallel Programming is a “Big” Challenge

- **You need to deal with A LOT OF parallelization details**
  - Parallelism abstraction (software + hardware)
  - Concurrency control
  - Task and data race avoidance
  - Dependency constraints
  - Scheduling efficiencies (load balancing)
  - Performance portability
  - …
- **And, don’t forget about trade-offs**
  - Desires vs Performance



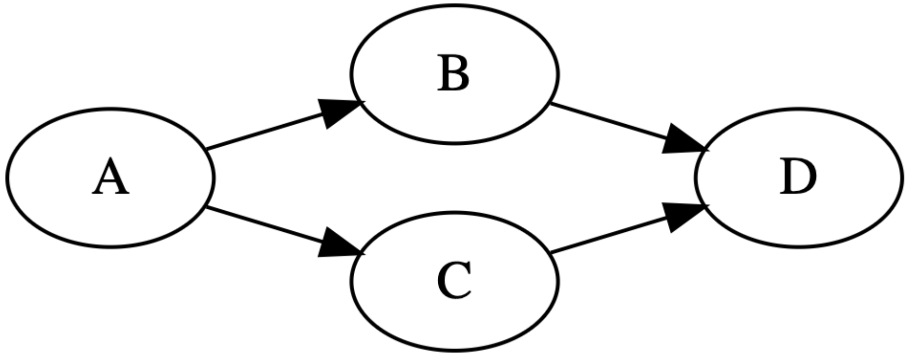
Need a New Programming Solution

- **Why existing parallel programming systems are not sufficient?**
  - Good at loop parallelism but weak in large and irregular task parallelism
  - Count on directed acyclic graph (DAG) model that cannot handle control flow
- **Envisioning from the evolution of parallel programming:**
  - Task parallelism is the best model for heterogeneous computing
- **Plenty of challenges to be solved …**
  - New applications demand new tasking models
    - Cost of control flow becomes more important
  - New accelerators demand new schedulers
    - Must value performance portability
  - Sustainability over hardware generations
  - …



Our Solution: Taskflow

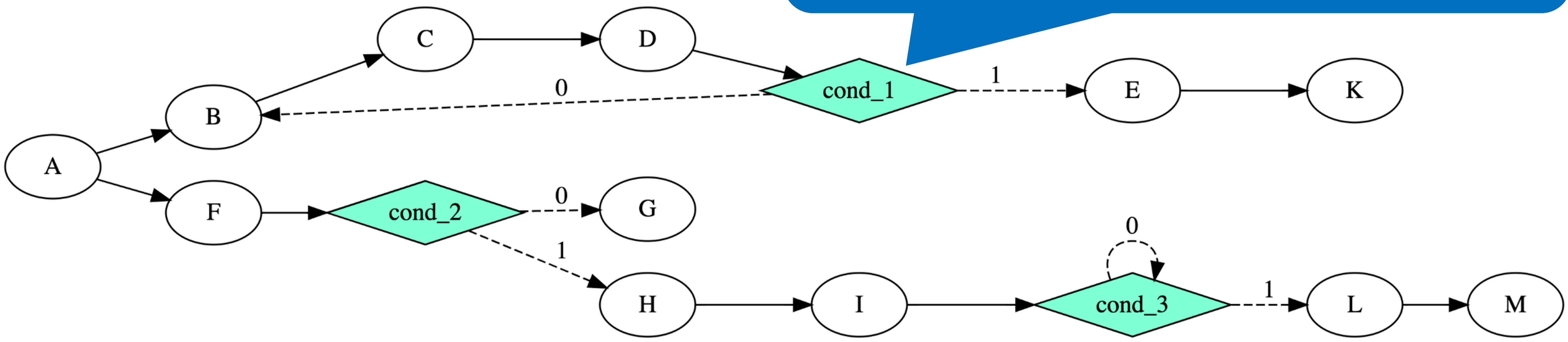
```cpp
#include <taskflow/taskflow.hpp>   // Taskflow is header-only, no wrangle with installation
int main(){
  tf::Taskflow taskflow;
  tf::Executor executor;
  auto [A, B, C, D] = taskflow.emplace(
    [] () { std::cout << "TaskA\n"; }
    [] () { std::cout << "TaskB\n"; },
    [] () { std::cout << "TaskC\n"; },
    [] () { std::cout << "TaskD\n"; }
  );
  A.precede(B, C);  // A runs before B and C
  D.succeed(B, C);  // D runs after    B and C
  executor.run(taskflow).wait();
  return 0;
}
```



Control Taskflow Graph Programming (CTFG)

```
// CTFG goes beyond the limitation of traditional DAG
auto cond_1 = taskflow.emplace([](){ return decision1(); });
auto cond_2 = taskflow.emplace([](){ return decision2(); });
auto cond_3 = taskflow.emplace([](){ return decision3(); });
cond_1.precede(B, E);          // cycle
cond_2.precede(G, H);          // if-else
cond_3.precede(cond_3, L);     // loop
```

Very difficult for existing DAG-based systems to express an efficient overlap between tasks and control flow …



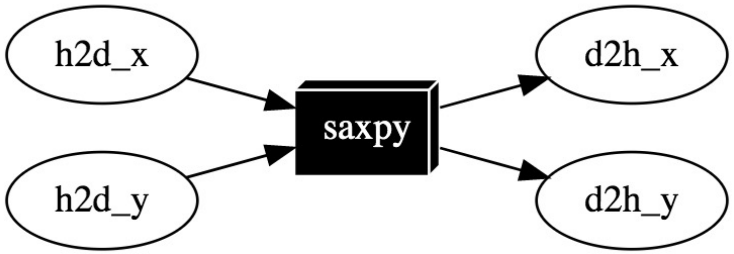
Heterogeneous Tasking

```
const unsigned N = 1<<20;
std::vector<float> hx(N, 1.0f), hy(N, 2.0f);
float *dx{nullptr}, *dy{nullptr};
auto allocate_x = taskflow.emplace([&](){ cudaMalloc(&dx, 4*N);});
auto allocate_y = taskflow.emplace([&](){ cudaMalloc(&dy, 4*N);});

auto cudaflow = taskflow.emplace([&](tf::cudaFlow& cf) {
  auto h2d_x = cf.copy(dx, hx.data(), N);  // CPU-GPU data transfer
  auto h2d_y = cf.copy(dy, hy.data(), N);
  auto d2h_x = cf.copy(hx.data(), dx, N);  // GPU-CPU data transfer
  auto d2h_y = cf.copy(hy.data(), dy, N);
  auto kernel = cf.kernel((N+255)/256, 256, 0, saxpy, N, 2.0f, dx, dy);
  kernel.succeed(h2d_x, h2d_y).precede(d2h_x, d2h_y);
});

cudaflow.succeed(allocate_x, allocate_y);
executor.run(taskflow).wait();
```

cudaFlow automatically transforms an application GPU task graph to an optimized **"CUDA graph"**



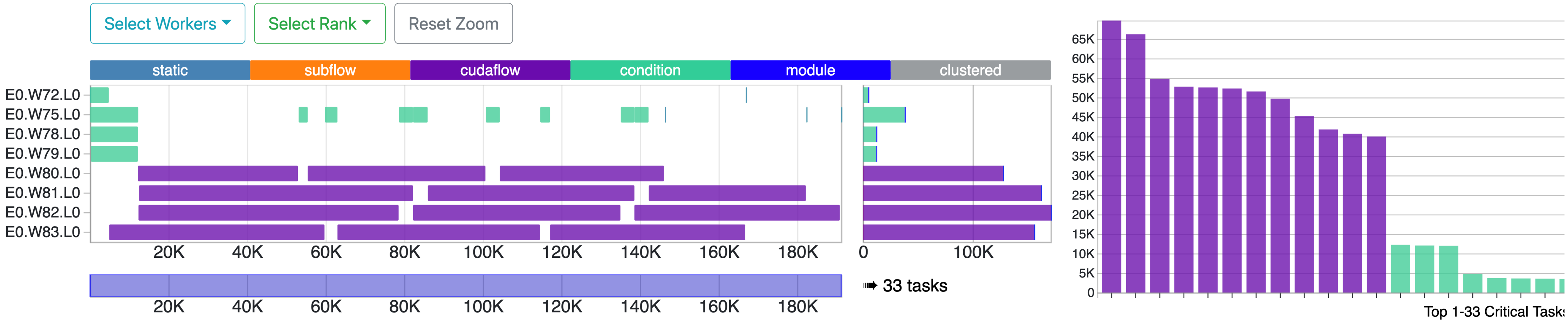
Drop-in Integration

- **Taskflow is header-only – *no wrangle with installation***
  - Include Taskflow to your project and tell your compiler where to find it

```
# Compile your program with Taskflow
~$ git clone https://github.com/taskflow/taskflow.git
~$ g++ -std=c++17 simple.cpp –I taskflow/ –O2 –pthread –o simple
~$ ./simple
TaskA
TaskC
TaskB
TaskD
```

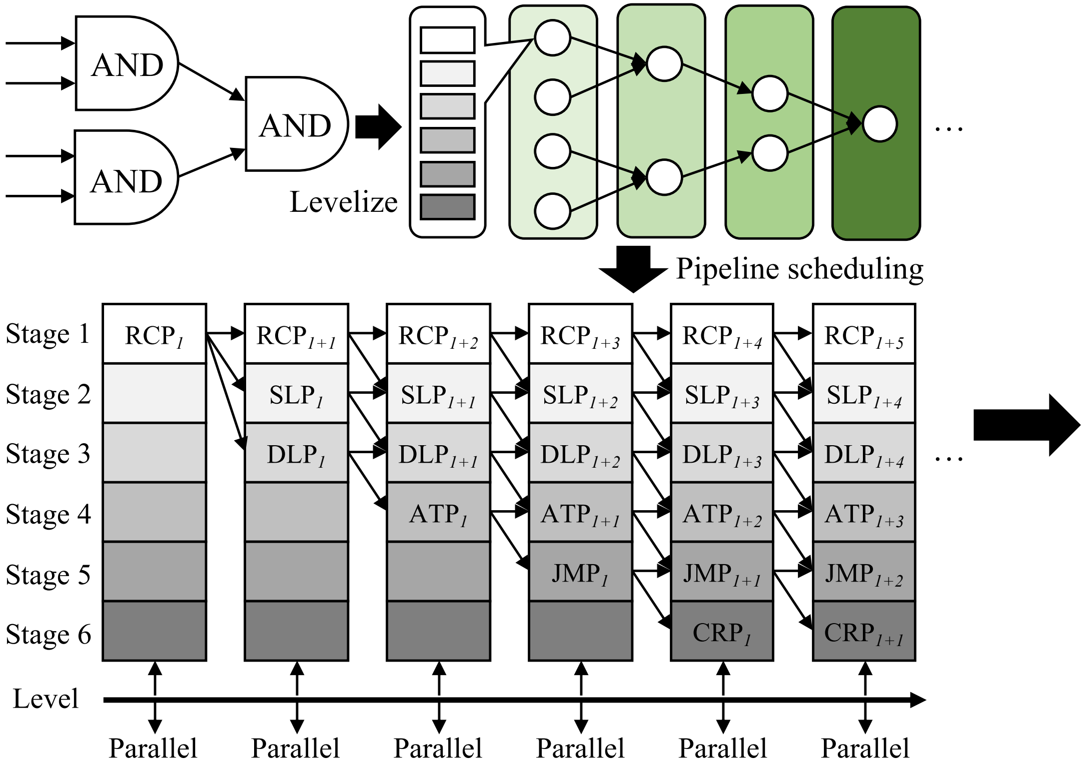
Built-in Visualizer using a Browser

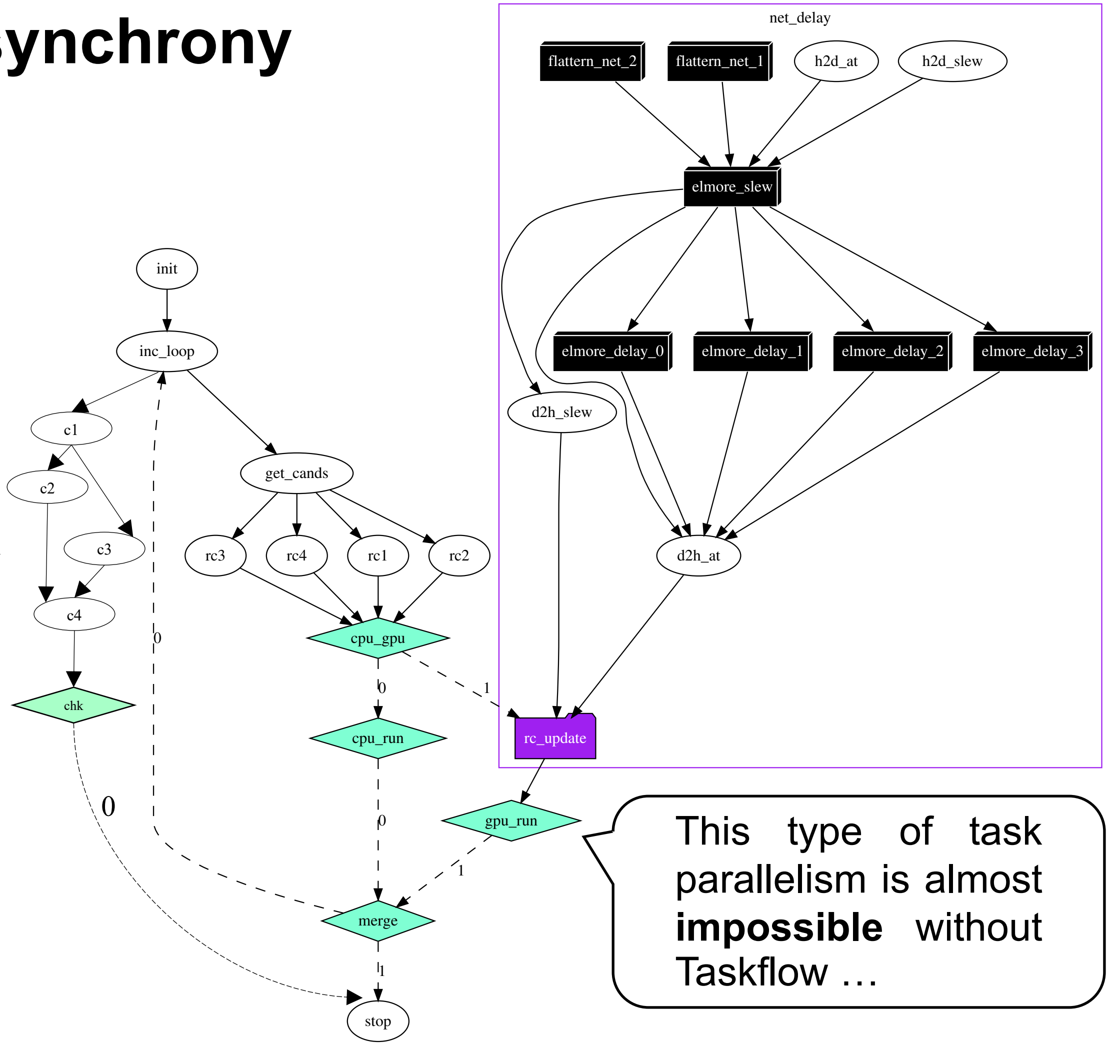
```
# Enable the environment variable TF_ENABLE_PROFILER for visualizer
~$ TF_ENABLE_PROFILER=simple.json ./simple
~$ cat simple.json
[
{"executor":"0", "data":[{"worker":0, "level":0, "data": …}]}
]
# Paste the JSON to https://taskflow.github.io/tfprof/
```



Application: Timing Analysis (TCAD'21)

- **Taskflow largely improves task asynchrony**





This type of task parallelism is almost **impossible** without Taskflow …

Tsung-Wei Huang, et al, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," *IEEE TCAD*, 2021

Application: Timing Analysis (DAC'21)

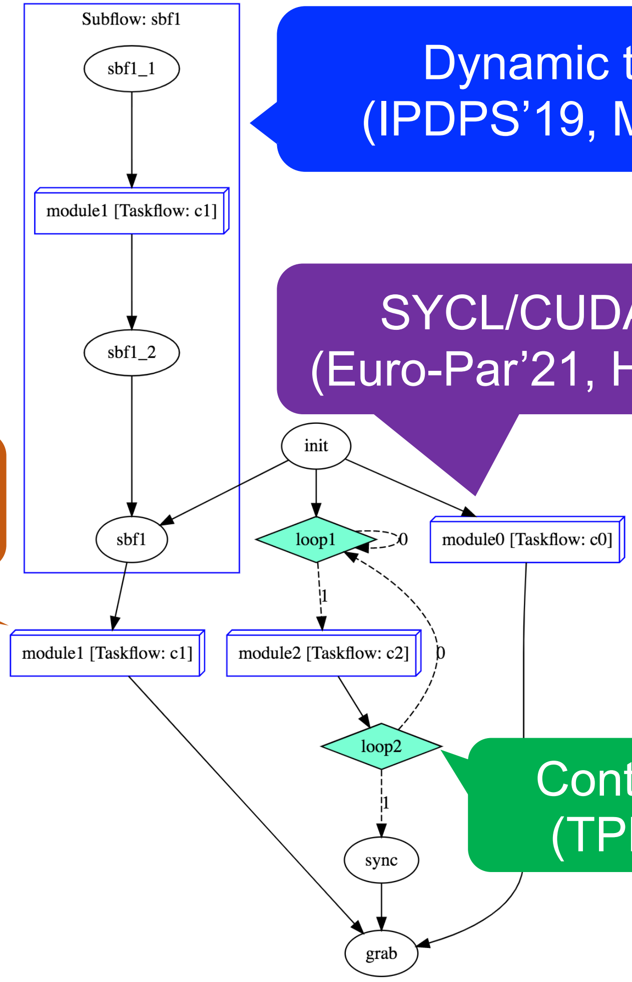
- **Applied Taskflow to accelerate path-based analysis on GPU**
  - Ex: leon3mp (1.6M gates): **611x speed-up** over 1 CPU (**44x** over 40 CPUs)
  - **Best paper award in TAU 2021**

Benchmark	#Pins	#Gates	#Arcs	OpenTimer Runtime	Our Algorithm #MDL=10		Our Algorithm #MDL=15		Our Algorithm #MDL=20	
					Runtime	Speed-up	Runtime	Speed-up	Runtime	Speed-up
leon2	4328255	1616399	7984262	2875783	4708.36	611×	5295.49ms	543×	5413.84	531×
leon3mp	3376821	1247725	6277562	1217886	5520.85	221×	7091.79ms	172×	8182.84	149×
netcard	3999174	1496719	7404006	752188	2050.60	367×	2475.90ms	304×	2484.08	303×
vga_lcd	397809	139529	756631	53204	682.94	77.9×	683.04ms	77.9×	706.16	75.3×
vga_lcd_iccad	679258	259067	1243041	66582	720.40	92.4×	754.35ms	88.3×	766.29	86.9×
b19_iccad	782914	255278	1576198	402645	2144.67	188×	2948.94ms	137×	3483.05	116×
des_perf_ispd	371587	138878	697145	24120	763.79	31.6×	766.31ms	31.5×	780.56	30.9×
edit_dist_ispd	416609	147650	799167	614043	1818.49	338×	2475.12ms	248×	2900.14	212×
mgc_edit_dist	450354	161692	852615	694014	1463.61	474×	1485.65ms	467×	1493.90	465×
mgc_matric_mult	492568	171282	948154	214980	994.67	216×	1075.90ms	200×	1113.26	193×

Guannan Guo, Tsung-Wei Huang, Yibo Lin, and Martin Wong, "GPU-accelerated Path-based Timing Analysis," *IEEE/ACM Design Automation Conference (DAC)*, CA, 2021

Everything is Composable in Taskflow

- **End-to-end parallelism in one graph**
  - Task, dependency, control flow all together
  - Scheduling with whole-graph optimization
  - Efficient overlap among heterogeneous tasks
- **Largely improved productivity!**



Dynamic task (IPDPS'19, MM'19)

SYCL/CUDA task (Euro-Par'21, HPEC'20)

Composition (HPDC'22, ICPP'22, HPEC'19)

Control flow (TPDS'22)

Industrial use-case of productivity improvement using Taskflow

jcelerier
ossia score

Reddit: https://www.reddit.com/r/cpp/ [under taskflow]

I've migrated https://ossia.io from TBB flow graph to taskflow a couple weeks ago. Net +8% of throughput on the graph processing itself, and took only a couple hours to do the change. Also don't have to fight with building the TBB libraries for 30 different platforms and configurations since it's header only.

8 Reply Share Report Save Follow

ossia score

We Value Research Impacts for Sustainability

- **Taskflow**[1] has been downloaded thousands of times

[1]: Tsung-Wei Huang, et al., "Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System," IEEE TPDS, vol. 33, no. 6, pp. 1303-1320, June 2022

Use the right tool for the right job

Taskflow: https://taskflow.github.io

Thank You

Dr. Tsung-Wei Huang

tsung-wei.huang@utah.edu