Taskflow: A General-purpose Task-parallel Programming System

Dr. Tsung-Wei (TW) Huang
Department of Electrical and Computer Engineering
University of Wisconsin at Madison, Madison, WI

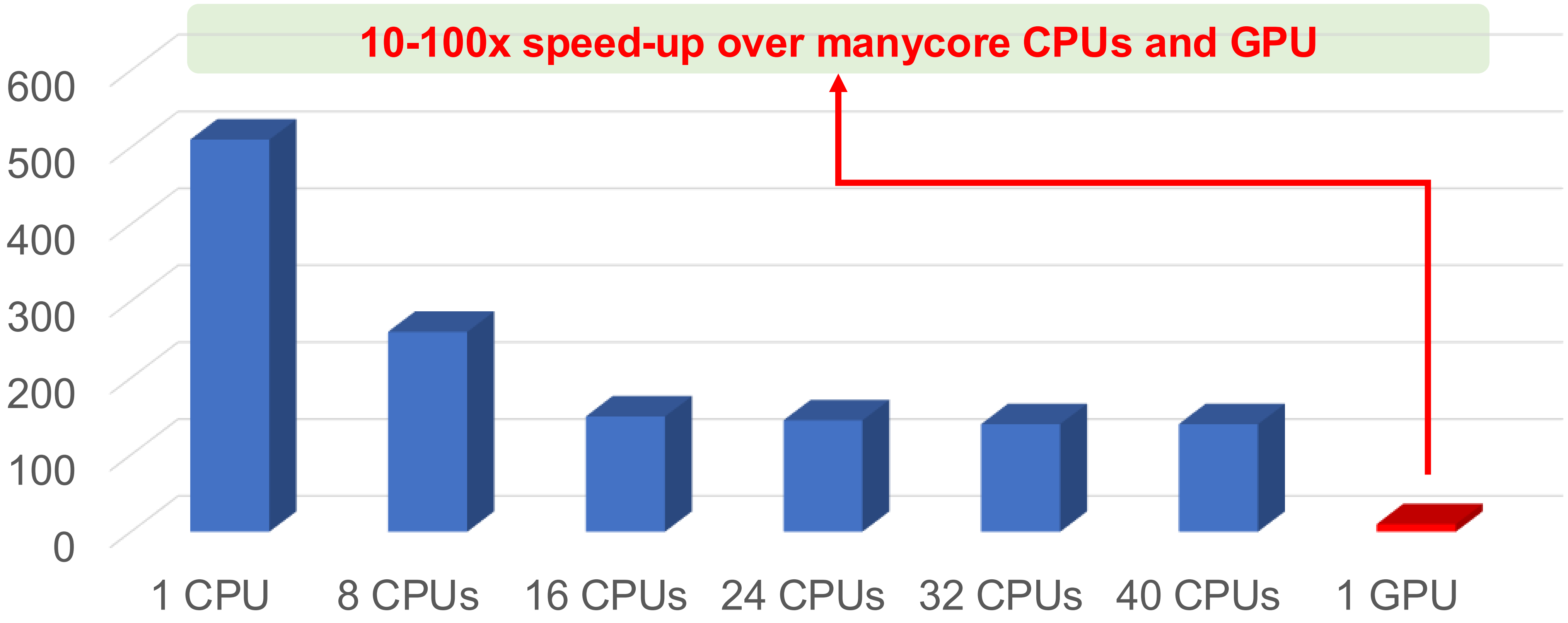
https://taskflow.github.io/

Takeaways

- Understand the importance of task-parallel programming
- Program static task graph parallelism using Taskflow
- Program dynamic task graph parallelism using Taskflow
- Showcase real-world applications of Taskflow
- Conclude the talk

Why Parallel Computing?

- **Advances performance to a new level previously out of reach**

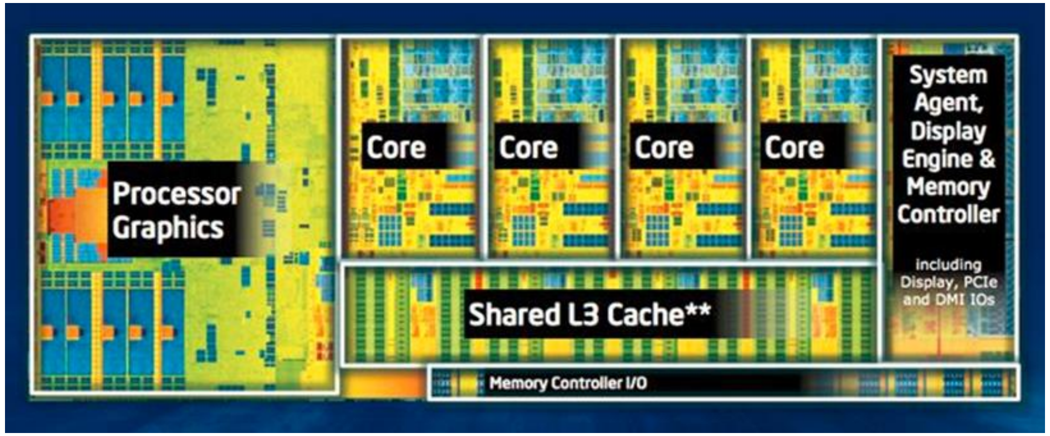


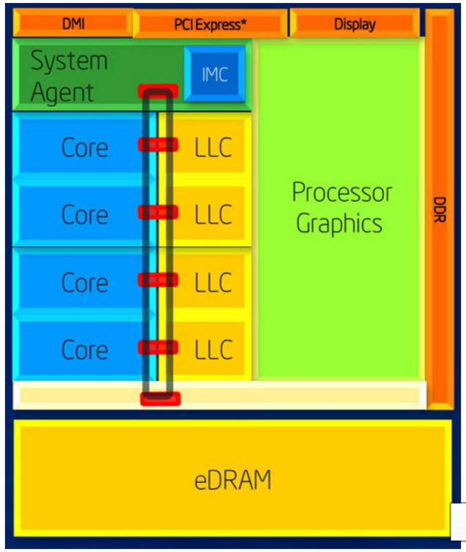
Time (minutes) to train a machine learning model

Modern Hardware is Designed to Run in Parallel

- **Intel Haswell microarchitecture**
  - Released in June 2013
  - Typically comes with four cores
  - Has an integrated GPU
  - 1.4 B transistors with 22 nm technology
  - Sophisticated design for ILP acceleration
  - Deep pipeline – 16 stages
- **Superscalar architecture**
  - Can issue and complete multiple independent instructions per cycle
- **Supports hyper-threading tech (HTT)**
  - Allows a single physical CPU core to appear as two logical processors to the OS

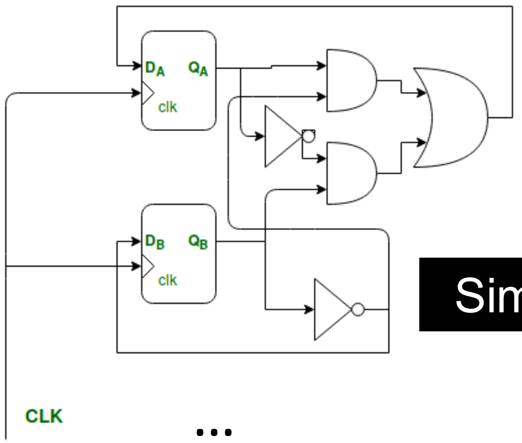
If you don't do parallel programming, you are not utilizing your hardware efficiently …

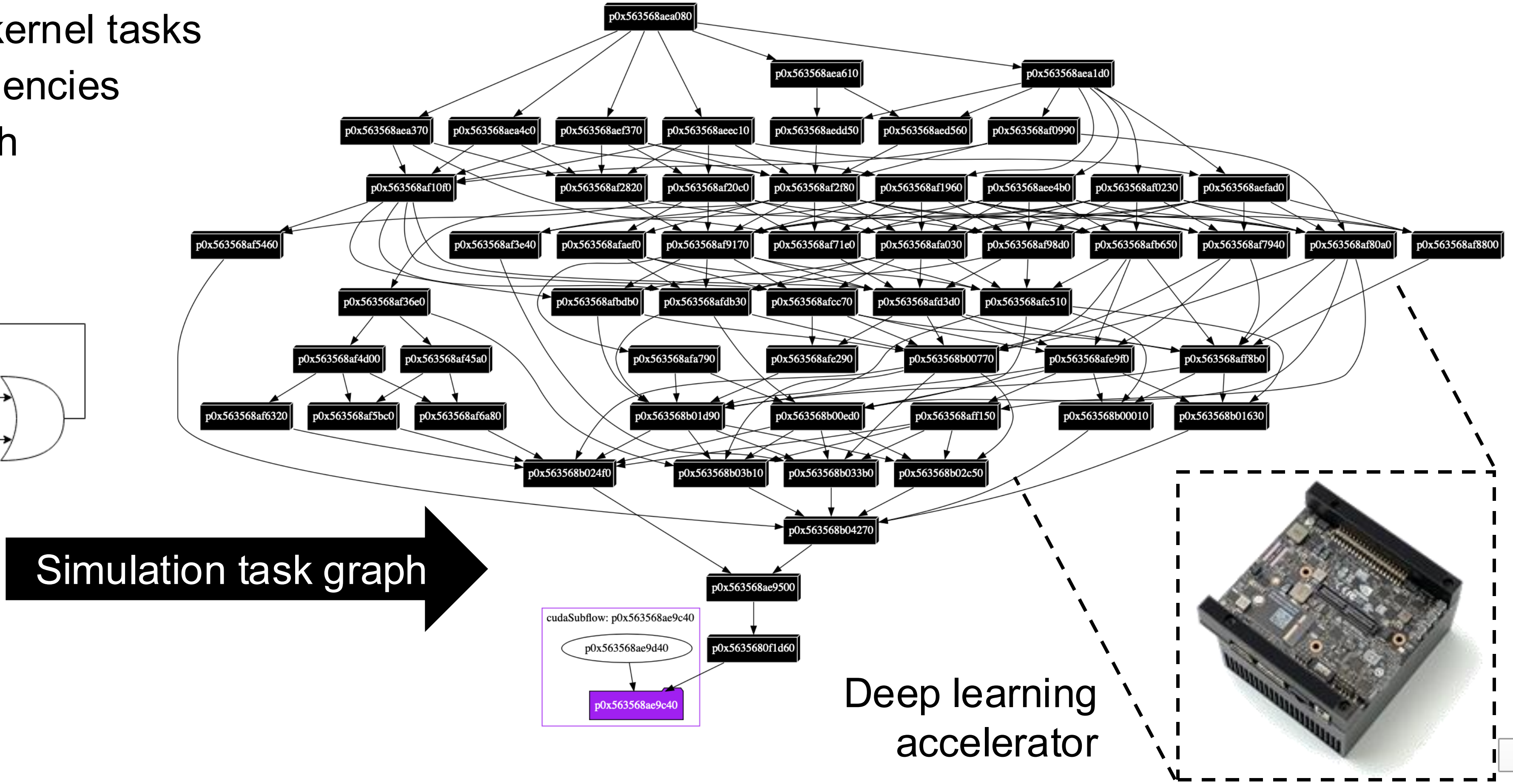




Today’s Parallel Computing Problem is Very Irregular

- **Computational task graph of a GPU-parallel circuit simulation workload[1]**
  - > 500M gates and nets
  - > 1000 async kernel tasks
  - > 1000 dependencies
  - > hours to finish

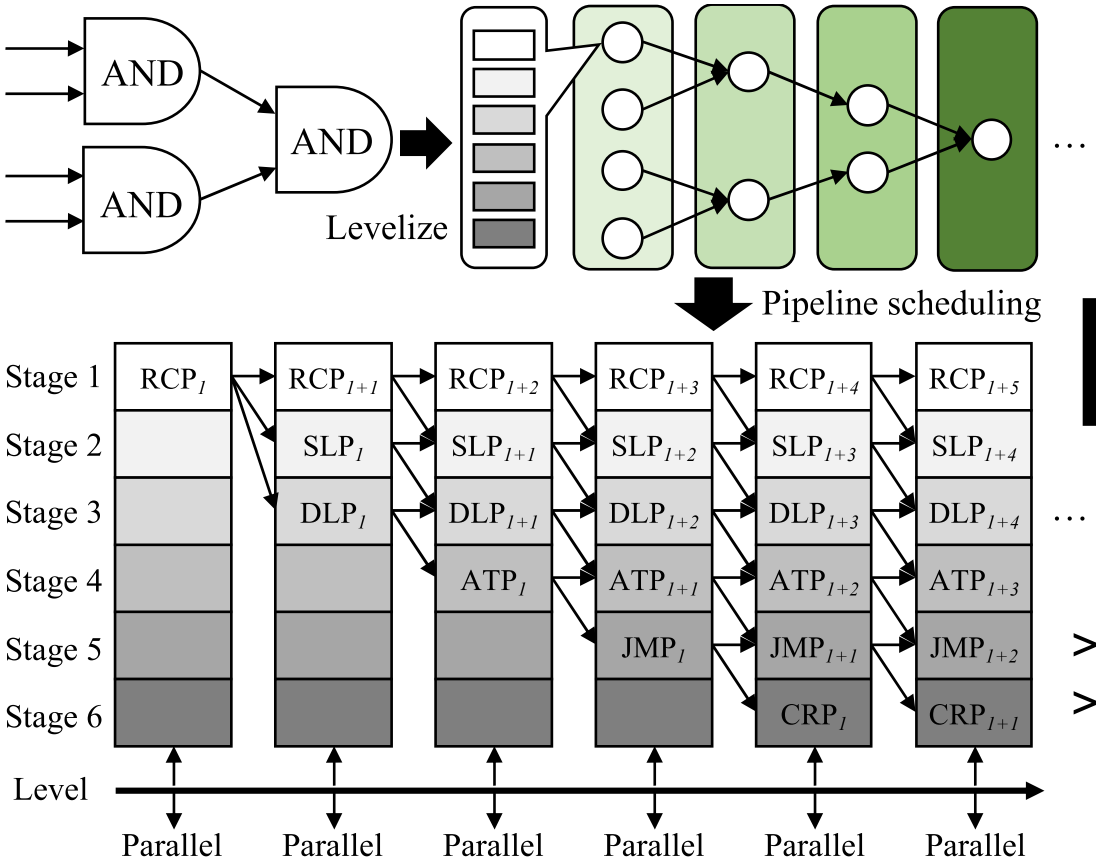




[1]: Dian-Lun Lin, et al, “From RTL to CUDA: A GPU Acceleration Flow for RTL Simulation with Batch Stimulus,” *ACM ICPP*, 2022

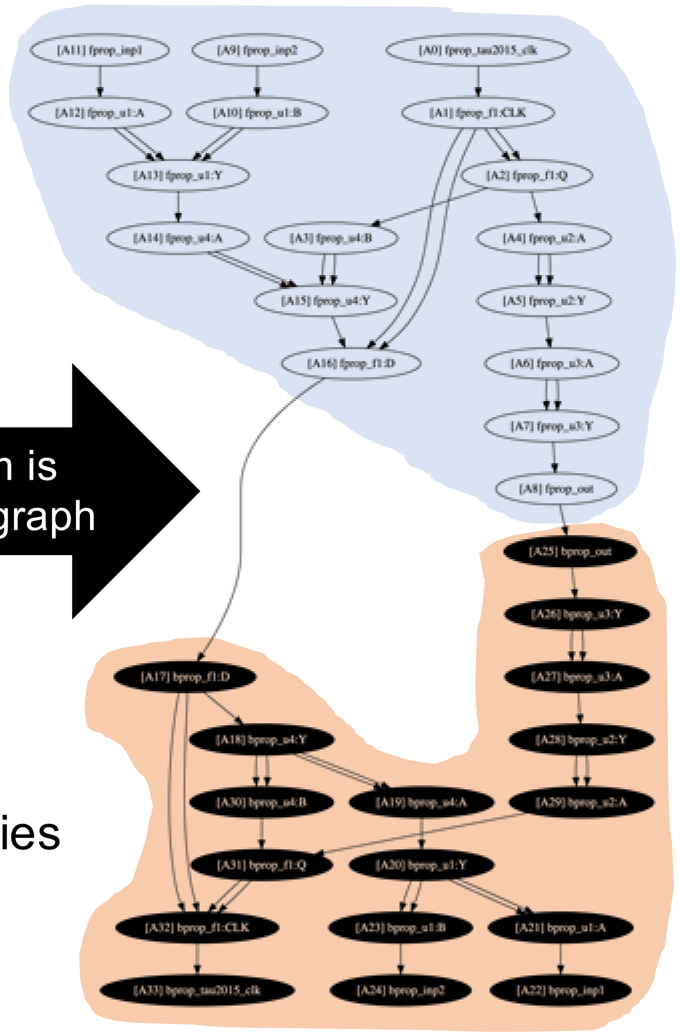
Another Example of Irregular Parallel Workload

- **CPU-parallel VLSI static timing analysis algorithm**



Analysis algorithm is modeled as a task graph

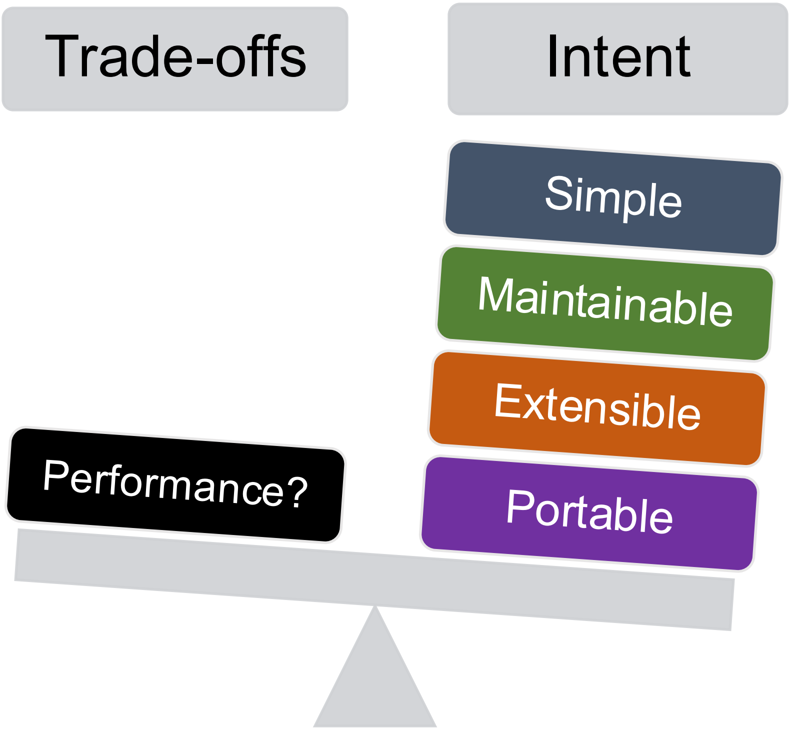
> 100M tasks
> 150M dependencies



[1]: Tsung-Wei Huang, et al, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," *IEEE TCAD*, 2022

Parallelizing such Irregular Workloads is Not Easy …

- **You need to deal with A LOT OF technical details**
    - Parallelism abstraction (software + hardware)
    - Concurrency control
    - Synchronization
    - Task and data race avoidance
    - Dependency constraints
    - Scheduling efficiencies (load balancing)
    - Programming productivity
    - Performance portability
    - …
- **And, don't forget about trade-offs**
    - Performance vs Developer's intent

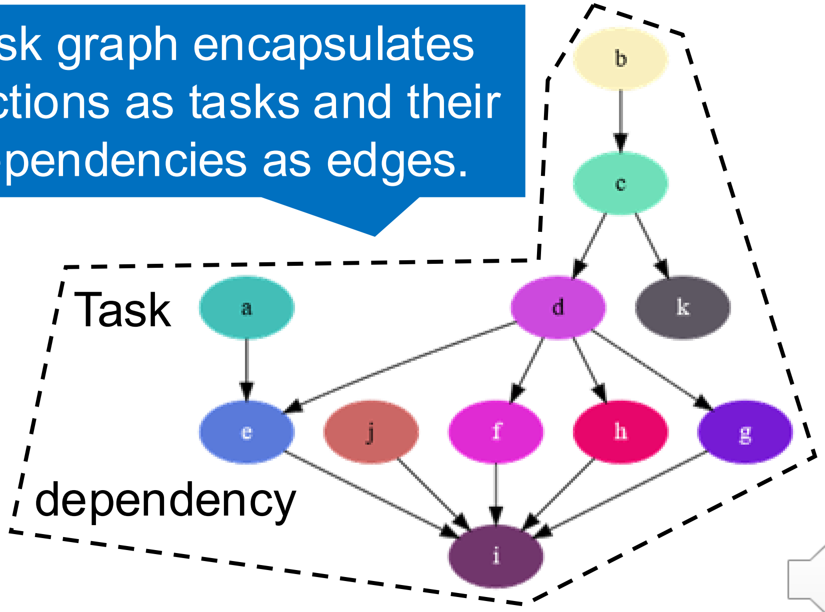


We want a solution that can sit on top to help programmers manage these details as much as possible because programmers care how fast (performance + productivity) they can get things done!

Why Task-parallel Programming (TPP)?

- **TPP is an effective solution for parallelizing irregular workloads**
  - Captures developers' intention in decomposing an algorithm into a *top-down* task graph
  - Delegates difficult scheduling details (e.g., load balancing) to an optimized runtime
- **Modern parallel programming libraries are moving towards task parallelism**
  - OpenMP 4.0 task dependency clauses (`omp depend`)
  - C++26 execution control library (`std::exec`)
  - TBB flow graph (`tbb::flow::graph`)
  - Taskflow control Taskflow graph (CTFG) model
  - … (many others)

Task graph encapsulates functions as tasks and their dependencies as edges.



OpenMP StarPU kokkos TBB PaRSEC

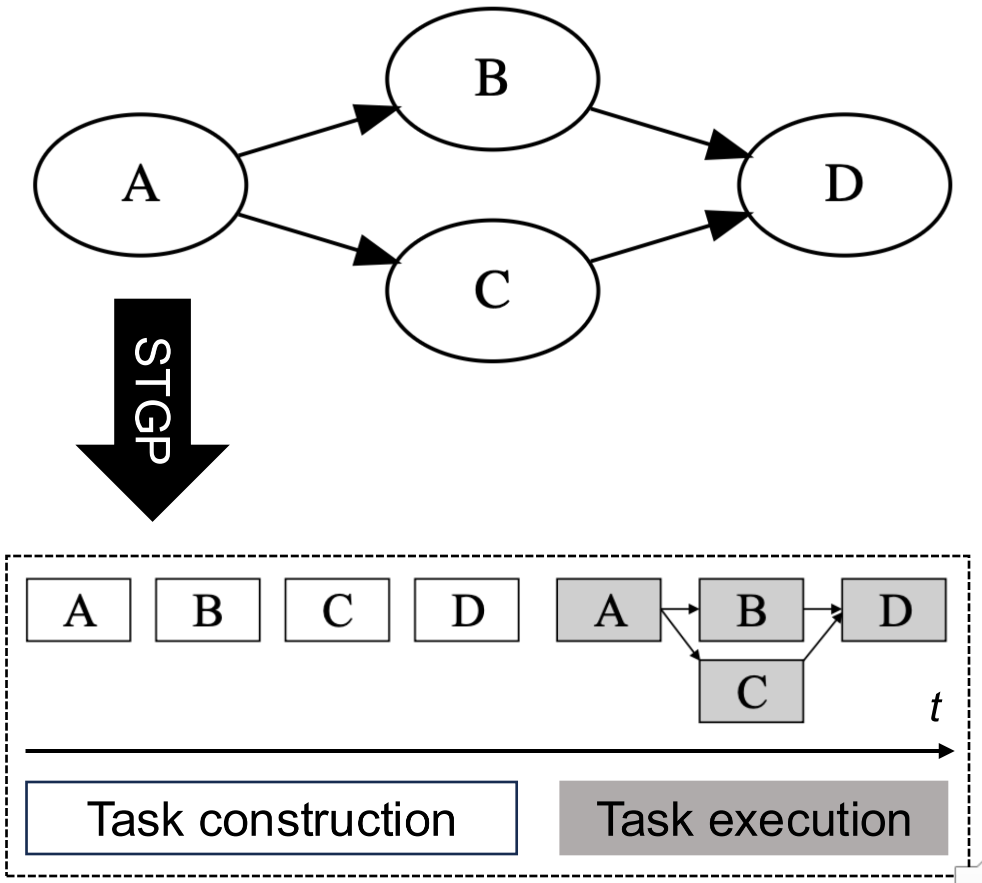
Takeaways

- Understand the importance of task-parallel programming
- Program static task graph parallelism using Taskflow
- Program dynamic task graph parallelism using Taskflow
- Showcase real-world applications of Taskflow
- Conclude the talk

Static Task Graph Programming in Taskflow[1]

```
#include <taskflow/taskflow.hpp>  // Live: https://godbolt.org/z/j8hx3xnnx

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);
  D.succeed(B, C);
  executor.run(taskflow).wait();
  return 0;
}
```



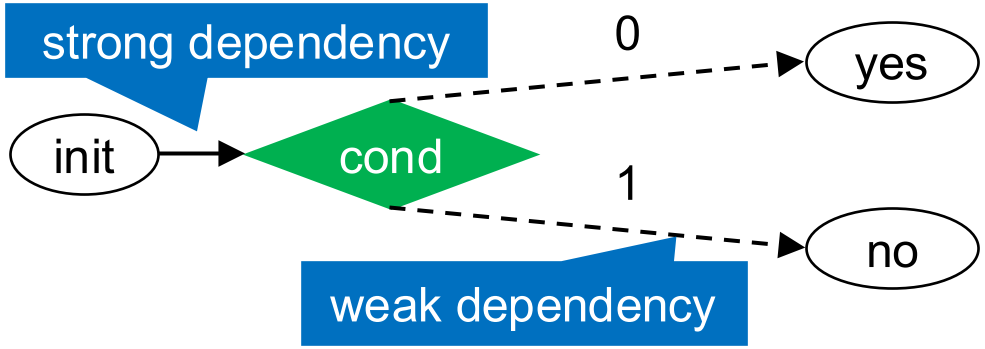
[1]: T.-W. Huang, et. al, "Taskflow: A Lightweight Parallel and Heterogeneous Task Graph Computing System," *IEEE TPDS*, 2022

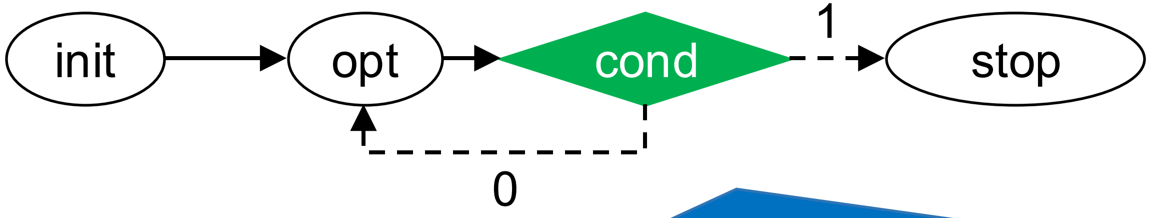
Control Taskflow Graph (CTFG) Programming Model

- **A key innovation that distinguishes Taskflow from existing libraries**

```
auto [init, cond, yes, no] =
taskflow.emplace(
  [] () { },
  [] () { return 0; },
  [] () { std::cout << "yes"; },
  [] () { std::cout << "no"; }
);
cond.succeed(init)
    .precede(yes, no);
```

```
auto [init, opt, cond, stop] =
taskflow.emplace(
  [&](){ initialize_data_structure(); },
  [&](){ some_optimizer(); },
  [&](){ return converged() ? 1 : 0; },
  [&](){ std::cout << ”done!\n"; }
);
opt.succeed(init).precede(cond);
converged.precede(opt, stop);
```





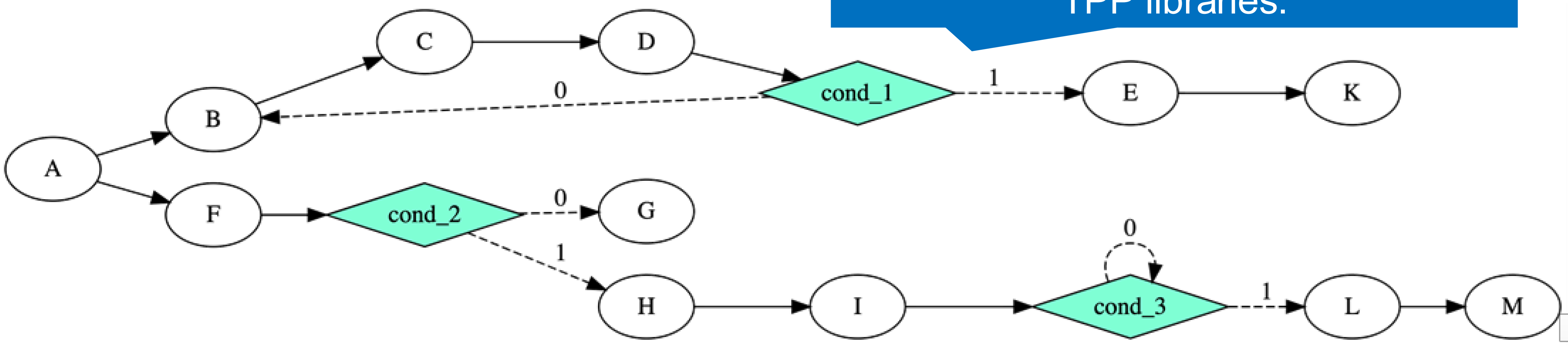
CTFG goes beyond the limitation of traditional DAG-based frameworks (no in-graph control flow).

Power of CTFG Programming Model

- **Enables very efficient overlap among tasks alongside control flow**

```
auto cond_1 = taskflow.emplace([](){ return run_B() ? 0 : 1; });
auto cond_2 = taskflow.emplace([](){ return run_G() ? 0 : 1; });
auto cond_3 = taskflow.emplace([](){ return loop() ? 0 : 1; });
cond_1.precede(B, E);
cond_2.precede(G, H);
cond_3.precede(cond_3, L);
```

This type of parallelism is almost impossible to achieve using existing TPP libraries.



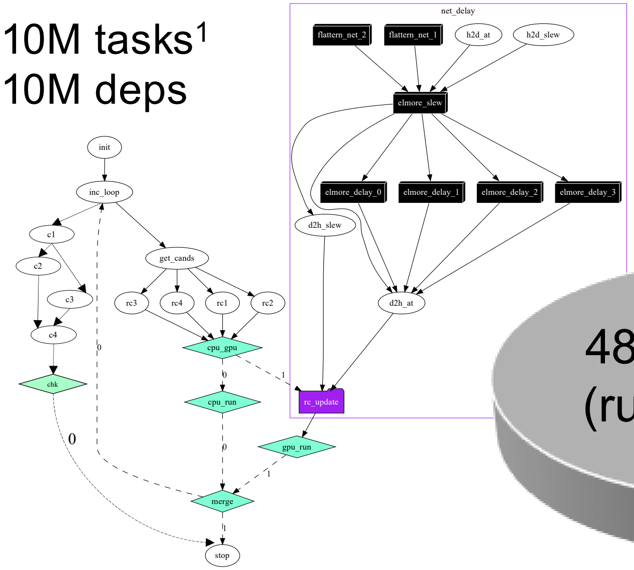
Takeaways

- Understand the importance of task-parallel programming
- Program static task graph parallelism using Taskflow
- Program dynamic task graph parallelism using Taskflow
- Showcase real-world applications of Taskflow
- Conclude the talk

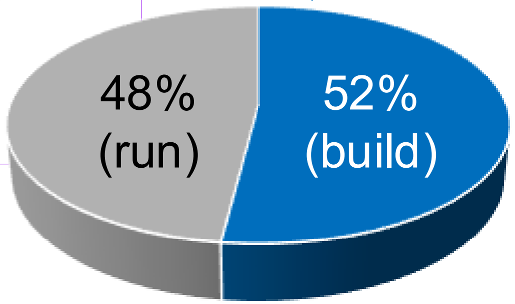
Why STGP Alone is Not Sufficient?

- **In STGP, the graph structure must be known up front before execution**
  - Execution of STGP is based on the *construct-and-run* model
- **Lack of overlap between task graph construction and task graph execution**
  - For large task graphs, this overlap can sometimes bring a significant performance advantage
- **Lack of flexibility for dynamically expressing task graph parallelism**
  - Task graph structure cannot depend on runtime values or control-flow results

>10M tasks[1]
>10M deps



Building graph is typically sequential (single thread) …




```
if (a == true) {
  G1 = build_task_graph1();
  if (b == true) {
    G2 = build_task_graph2();
    G1.precede(G2);
  }
  else {
    G3 = build_task_graph3();
    G3.precede(G1);
  } ...
```

STGP cannot handle graph structure that depends on control-flow results …

[1]: Tsung-Wei Huang, et al, “OpenTimer v2: A New Parallel Incremental Timing Analysis Engine,” *IEEE TCAD*, 2022

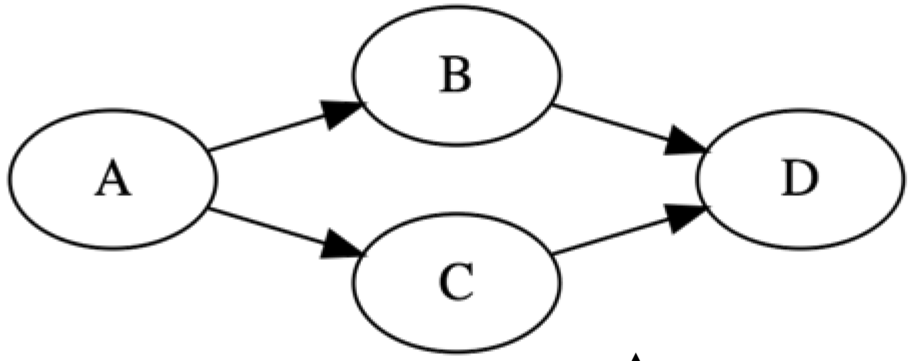
Dynamic Task Graph Programming in Taskflow

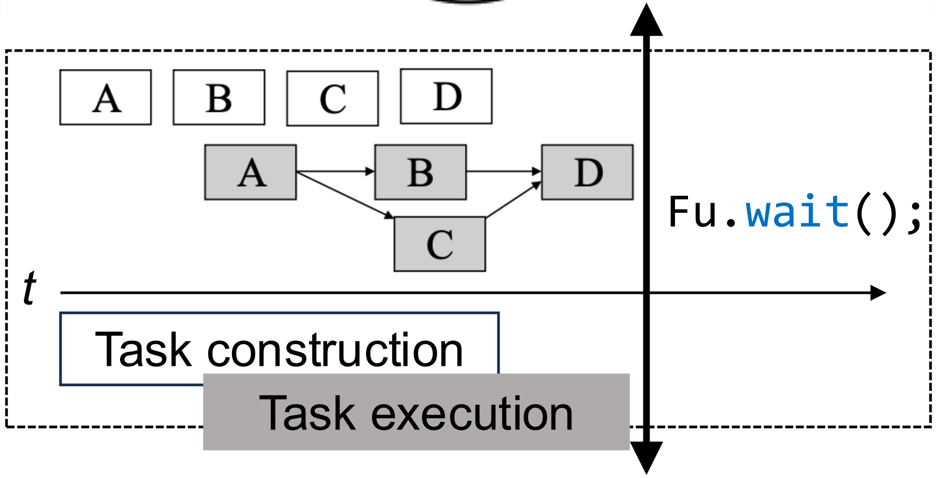
```cpp
// Live: https://godbolt.org/z/j76ThGbWK

tf::Executor executor;

auto A = executor.silent_dependent_async([](){
    std::cout << "TaskA\n";
});
auto B = executor.silent_dependent_async([](){
    std::cout << "TaskB\n";
}, A);
auto C = executor.silent_dependent_async([](){
    std::cout << "TaskC\n";
}, A);
auto [D, Fu] = executor.dependent_async([](){
    std::cout << "TaskD\n";
}, B, C);

Fu.wait();
```





Specify arbitrary task dependencies on previously created tasks

DTGP is Flexible for Runtime-driven Execution

- **Assemble task graphs driven by runtime variables and control-flow results**

```
if (a == true) {
  G1 = build_task_graph1();
  if (b == true) {
    G2 = build_task_graph2();
    G1.precede(G2);
    if (c == true) {
      … // defined other TGPs
    }
  }
  else {
    G3 = build_task_graph3();
    G1.precede(G3);
  }
}
```

```
G1 = build_task_graph1();
G2 = build_task_graph2();
if (G1.num_tasks() == 100) {
  G1.precede(G2);
}
else {
  G3 = build_task_graph3();
  G1.precede(G2, G3);
  if(G2.num_dependencies()>=10){
    … // define another TGP
  } else {
    … // define another TGP
  }
}
```

This type of dynamic task graph is very difficult to achieve using static task graph programming …

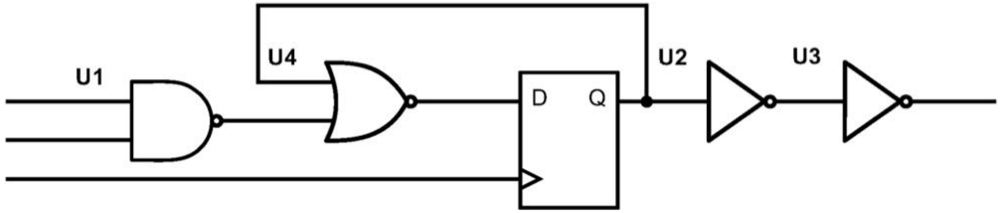
Takeaways

- Understand the importance of task-parallel programming
- Program static task graph parallelism using Taskflow
- Program dynamic task graph parallelism using Taskflow
- Showcase real-world applications of Taskflow
- Conclude the talk

VLSI Static Timing Analysis

- **Task-parallel timing propagation[1]**
  - Task: per-pin propagation function
    - Ex: cell delay, net delay calculator
  - Edge: pin-to-pin dependency
    - Ex: intra-/inter-gate dependencies

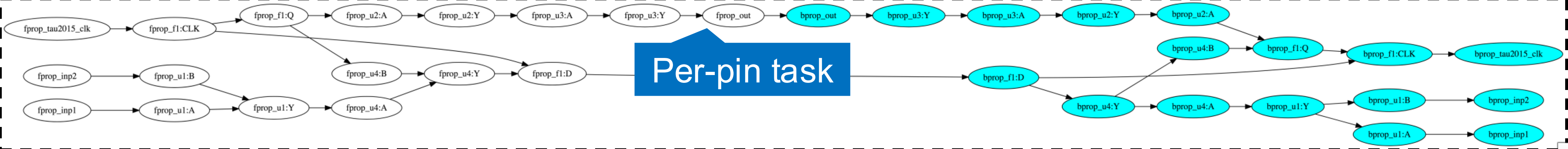
A circuit of four comb gates and one seq gate




```
ot> report_timing # report the most critical path
Startpoint    : inp1
Endpoint      : f1:D
Analysis type : min
----------------------------------------------------
Type Delay Time Dir Description
----------------------------------------------------
port 0.000 0.000 fall inp1
pin  0.000 0.000 fall u1:A (NAND2X1)
…
pin  0.000 2.967 fall f1:D (DFFNEGX1)
…
----------------------------------------------------
slack -23.551 VIOLATED
```

Derive a timing propagation task graph

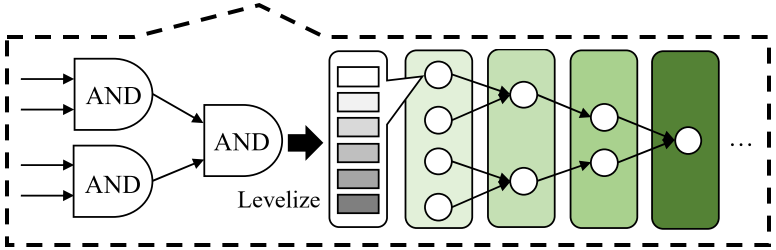
Evaluate and report violated data paths

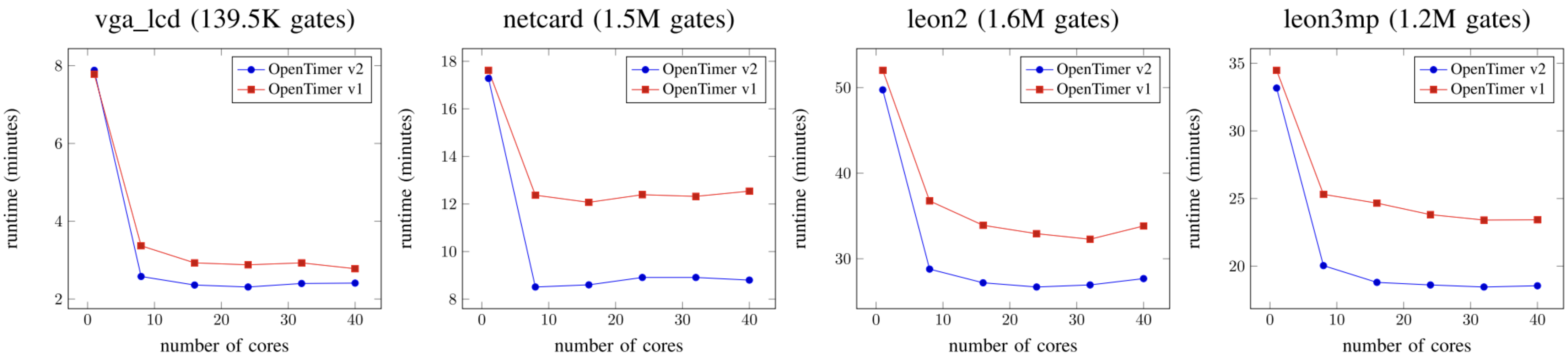


[1]: Tsung-Wei Huang, et al, “OpenTimer v2: A New Parallel Incremental Timing Analysis Engine,” *IEEE TCAD*, 2022

How Good is Task-parallel STA?

- **OpenTimer v1: levelization-based (or loop-parallel) timing propagation[1]**
  - Implemented using OpenMP "`parallel_for`" primitive
- **OpenTimer v2: task-parallel timing propagation[2]**
  - Implemented using Taskflow STGP



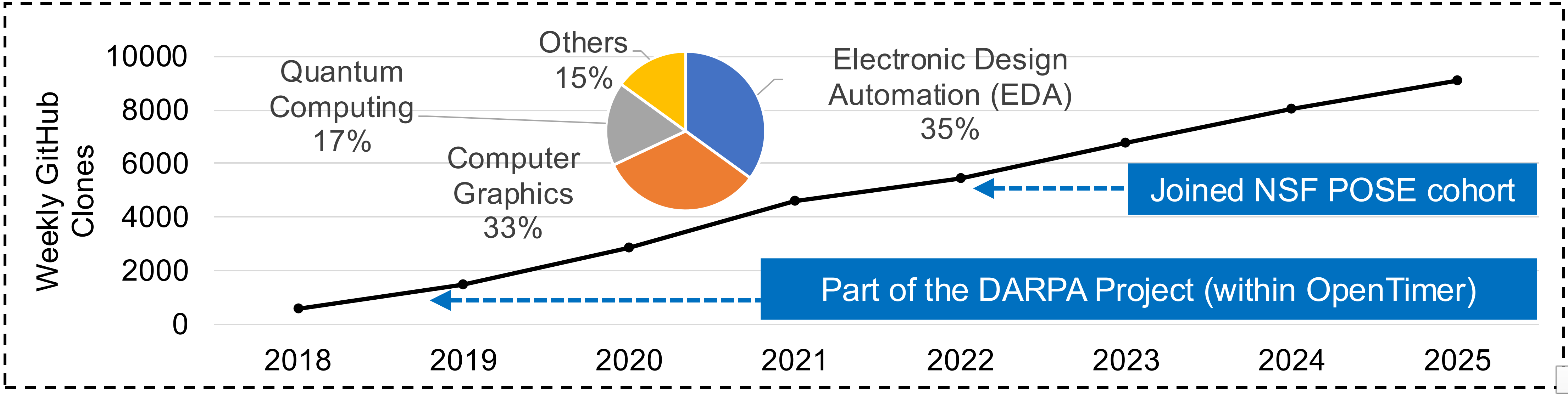


💡Task-parallelism allows us to more asynchronously parallelize the timing propagation

[1]: Tsung-Wei Huang and Martin Wong, "OpenTimer: A High-Performance Timing Analysis Tool," *IEEE/ACM ICCAD*, 2015
[2]: Tsung-Wei Huang, et al, "OpenTimer v2: A New Parallel Incremental Timing Analysis Engine," *IEEE TCAD*, 2022

Other Industrial Applications of Taskflow

- **At the time of this talk, Taskflow has weekly downloads of 7-9K**
  - Quantum computing (e.g., Xanadu uses Taskflow in JET to simulate quantum tensor network)
  - Bioinformatics (e.g., Roche uses Taskflow to parallelize its DNA sequencing software)
  - Computer graphics (e.g., Vulkan recommends Taskflow for parallelizing rendering engines)
  - Embedded systems (e.g., Tesseract uses Taskflow to design robotic planning software)
  - Scientific computing (e.g., Deal.II uses Taskflow to parallelize finite element analysis)
  - …



[1]: Taskflow: A General-purpose Task-parallel Programming Library: https://taskflow.github.io/

Takeaways

- Understand the importance of task-parallel programming
- Program static task graph parallelism using Taskflow
- Program dynamic task graph parallelism using Taskflow
- Showcase real-world applications of Taskflow
- Conclude the talk

Using Taskflow is EXTREMELY EASY

- **Taskflow is a header-only library built entirely with standard C++ libraries**
  - No wrangling with installation – just copy the headers and tell your compiler where to find them

```
# clone the Taskflow project
~$ git clone https://github.com/taskflow/taskflow.git
~$ cd taskflow
# compile your program and tell your compiler where to find Taskflow header files
~$ g++ -std=c++20 examples/simple.cpp –I ./ -O2 -pthread -o simple
~$ ./simple
TaskA
TaskC
TaskB
TaskD
```

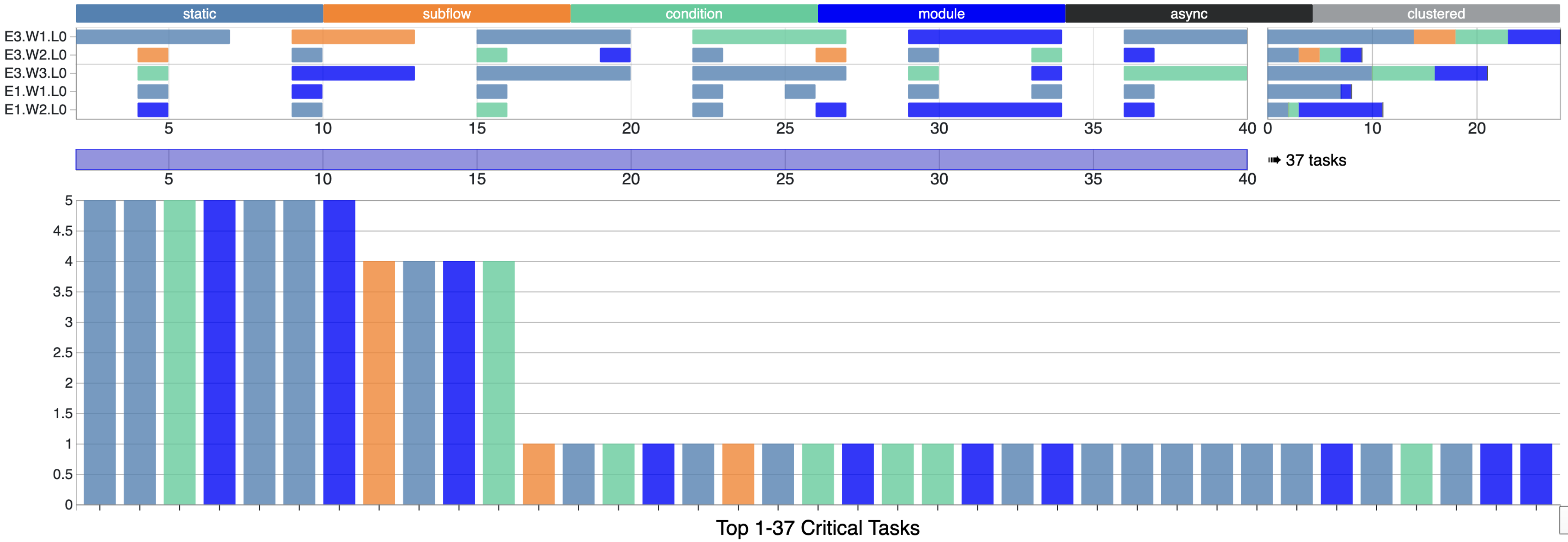
- **Taskflow has been evolving over the years to a stable programming system**
  - Started in 2018 as a DARPA-sponsored research project to parallelize critical EDA applications

[1]: Taskflow: A General-purpose Task-parallel Programming system in Modern C++: https://taskflow.github.io/

Visualize the Execution of Your Taskflow Programs

run you program with the env variable TF_ENABLE_PROFILER enabled and
paste the output JSON content to https://taskflow.github.io/tfprof/

~$ TF_ENABLE_PROFILER=simple.json ./simple



Thank you for using Taskflow! https://taskflow.github.io/

…

Question?

Taskflow: https://taskflow.github.io

Static Task Graph Programming (STGP)

```
// Live: https://godbolt.org/z/j8hx3xnnx

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);
D.succeed(B, C);
executor.run(taskflow).wait();
```

Dynamic Task Graph Programming (DTGP)

```
// Live: https://godbolt.org/z/T87PrTarx

tf::Executor executor;
auto A = executor.silent_dependent_async([]{
  std::cout << "TaskA\n";
});
auto B = executor.silent_dependent_async([]{
  std::cout << "TaskB\n";
}, A);
auto C = executor.silent_dependent_async([]{
  std::cout << "TaskC\n";
}, A);
auto D = executor.silent_dependent_async([]{
  std::cout << "TaskD\n";
}, B, C);
executor.wait_for_all();
```

NSF DARPA NUMFOCUS OPEN CODE = BETTER SCIENCE Google Summer of Code