A General-purpose Parallel and Heterogeneous Task Programming System at Scale

Dr. Tsung-Wei Huang

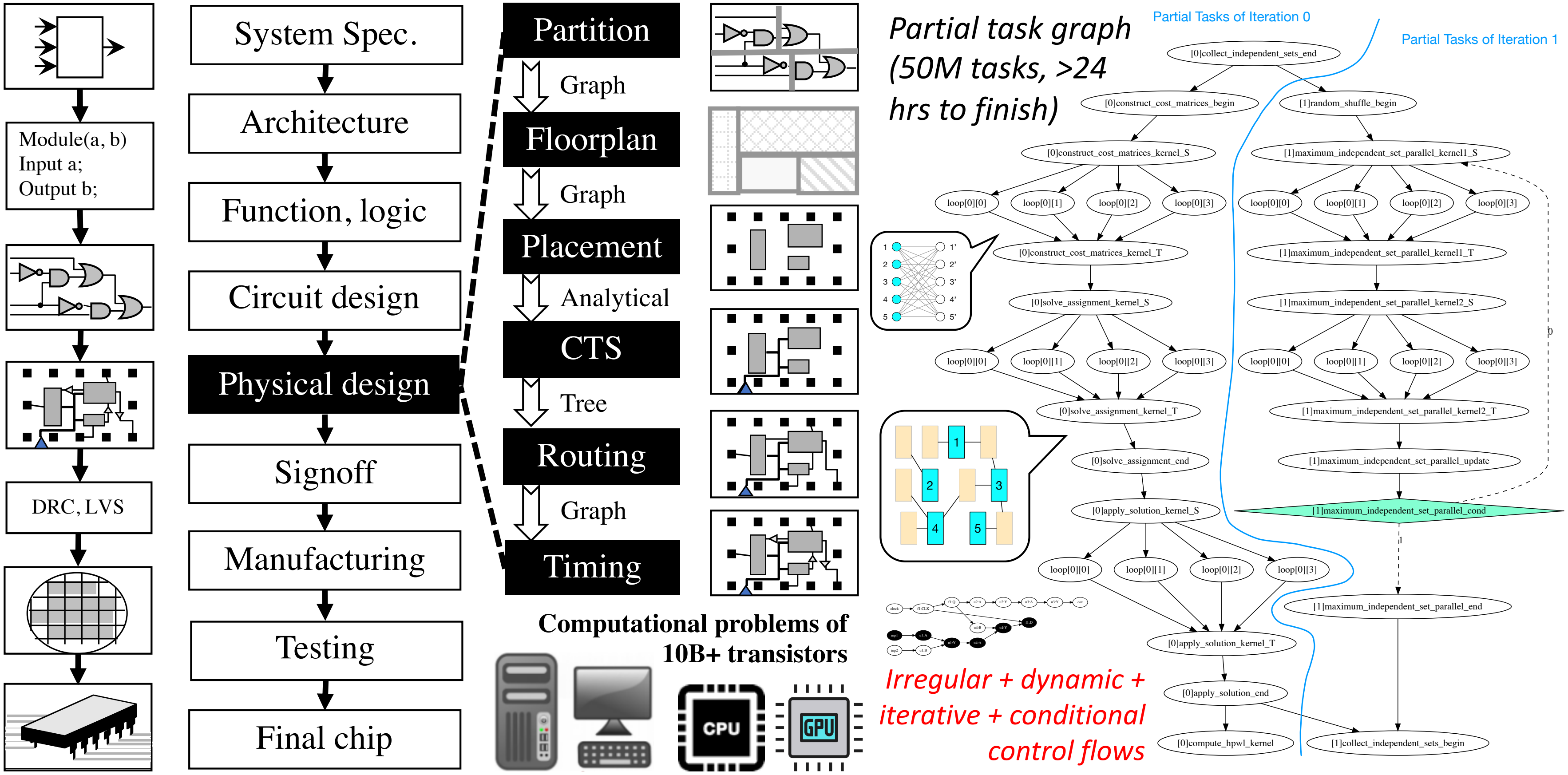
Department of Electrical and Computer Engineering

University of Utah, Salt Lake City, UT

How can we make it easier for scientific software developers to program large parallel and heterogeneous resources with high performance scalability and simultaneous high productivity?

Real Experience: Parallelizing VLSI CAD Software

- ❑ **CAD has many of the most difficult CS problems**
  - ❑ multidisciplinary, irregular, dynamic, billions of tasks, etc



Module(a, b)
Input a;
Output b;

DRC, LVS

System Spec. → Architecture → Function, logic → Circuit design → Physical design → Signoff → Manufacturing → Testing → Final chip

Partition → Graph → Floorplan → Graph → Placement → Analytical → CTS → Tree → Routing → Graph → Timing

Computational problems of 10B+ transistors

Partial task graph (50M tasks, >24 hrs to finish)

Irregular + dynamic + iterative + conditional control flows

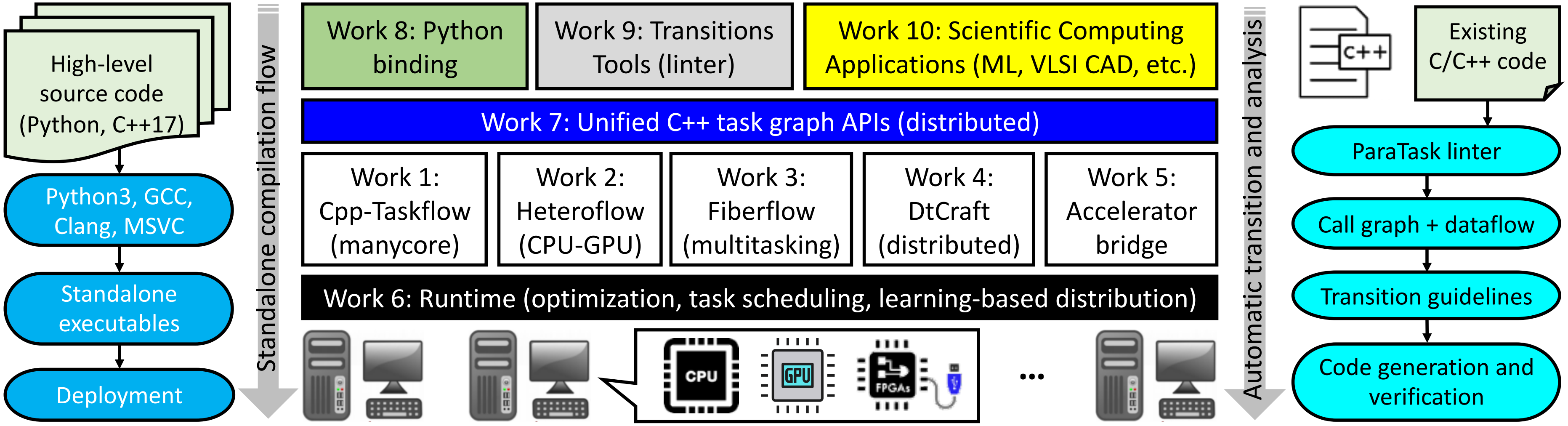
CPU
GPU
Partial Tasks of Iteration 0
Partial Tasks of Iteration 1
[0]collect_independent_sets_end
[0]construct_cost_matrices_begin
[1]random_shuffle_begin
[0]construct_cost_matrices_kernel_S
[1]maximum_independent_set_parallel_kernel1_S
loop[0][0]
loop[0][1]
loop[0][2]
loop[0][3]
[0]construct_cost_matrices_kernel_T
[1]maximum_independent_set_parallel_kernel1_T
[0]solve_assignment_kernel_S
[1]maximum_independent_set_parallel_kernel2_S
[0]solve_assignment_kernel_T
[1]maximum_independent_set_parallel_kernel2_T
[0]solve_assignment_end
[1]maximum_independent_set_parallel_update
[0]apply_solution_kernel_S
[1]maximum_independent_set_parallel_cond
[0]apply_solution_kernel_T
[1]maximum_independent_set_parallel_end
[0]apply_solution_end
[0]compute_hpwl_kernel
[1]collect_independent_sets_begin
1
2
3
4
5
1'
2'
3'
4'
5'

CAD is Demanding New Parallelism!

- ❑ **Yet, most existing solutions are incremental …**
  - ❑ Augment codebase with OpenMP, Pthreads, MPI, Intel TBB, and CUDA to introduce incremental parallelism
  - ❑ Solutions are heavily hard-coded by resorting everything to "heroic programmers"
- ❑ **Why are we sluggishly changing this?**
  - ❑ Existing programming tools (HPX, SYCL, Kokkos, RAJA, PaRSEC, StarPU, etc.) are perfect in data-parallel programs but short in:
    - Steep learning curve of new models (syntax + semantic)
    - Dynamic/conditional/cyclic control flows
    - Composition to handle complex heterogeneous workflows
  - ❑ Lack of automatic transition and verification tools

A General-purpose Parallel Task Programming System

- ❑ **Task-based approach scales best with parallel arch**
- ❑ **We need to handle various computational needs**
  - ❑ Dynamic controls, cyclic flows, composition, irregularity
- ❑ **Transition tools to ease the adoption of new models**
  - ❑ Never acceptable if everything is done manually



High-level source code (Python, C++17)

Python3, GCC, Clang, MSVC

Standalone executables

Deployment

Standalone compilation flow

Work 8: Python binding

Work 9: Transitions Tools (linter)

Work 10: Scientific Computing Applications (ML, VLSI CAD, etc.)

Work 7: Unified C++ task graph APIs (distributed)

Work 1: Cpp-Taskflow (manycore)

Work 2: Heteroflow (CPU-GPU)

Work 3: Fiberflow (multitasking)

Work 4: DtCraft (distributed)

Work 5: Accelerator bridge

Work 6: Runtime (optimization, task scheduling, learning-based distribution)

Automatic transition and analysis

Existing C/C++ code

ParaTask linter

Call graph + dataflow

Transition guidelines

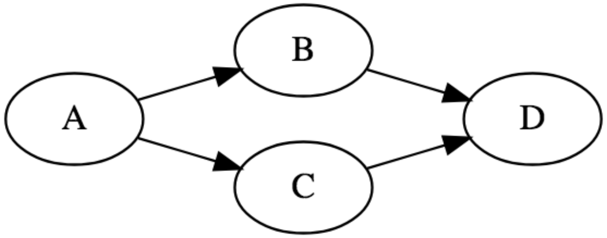
Code generation and verification

This is an on-going large system project with many components under construction!

“Hello World” in Cpp-Taskflow [IPDPS19]

```
#include <taskflow/taskflow.hpp>   // Cpp-Taskflow is header-only
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);            // 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
   executor.run(taskflow);  // create an executor to run the taskflow
   return 0;
}
```

*Only **17 lines** of code to get a parallel task execution!*



“Hello World” in OpenMP

```
#include <omp.h>  // OpenMP is a lang ext to describe parallelism in compiler directives
int main(){
   #omp parallel num_threads(std::thread::hardware_concurrency())
   {
      int A_B, A_C, B_D, C_D;
      #pragma omp task depend(out: A_B, A_C)
      {
         std::cout << ”TaskA\n” ;
      }
      #pragma omp task depend(in: A_B; out: B_D)
      {
         std::cout << ” TaskB\n” ;
      }
      #pragma omp task depend(in: A_C; out: C_D)
      {
         std::cout << ” TaskC\n” ;
      }
      #pragma omp task depend(in: B_D, C_D)
      {
         std::cout << ”TaskD\n” ;
      }
   }
   return 0;
}
```

Task dependency clauses

Task dependency clauses

Task dependency clauses

Task dependency clauses

OpenMP task clauses are static and explicit; Programmers are responsible for a proper order of writing tasks consistent with sequential execution

“Hello World” in Intel’s TBB Library

```
#include <tbb.h>  // Intel’s TBB is a general-purpose parallel programming library in C++
int main(){
    using namespace tbb;
    using namespace tbb:flow;
    int n = task_scheduler init::default_num_threads () ;
    task scheduler_init init(n);
    graph g;
    continue_node<continue_msg> A(g, [] (const continue msg &) {
        std::cout << “TaskA” ;
    }) ;
    continue_node<continue_msg> B(g, [] (const continue msg &) {
        std::cout << “TaskB” ;
    }) ;
    continue_node<continue_msg> C(g, [] (const continue msg &) {
        std::cout << “TaskC” ;
    }) ;
    continue_node<continue_msg> C(g, [] (const continue msg &) {
        std::cout << “TaskD” ;
    }) ;
    make_edge(A, B);
    make_edge(A, C);
    make_edge(B, D);
    make_edge(C, D);
    A.try_put(continue_msg());
    g.wait_for_all();
}
```

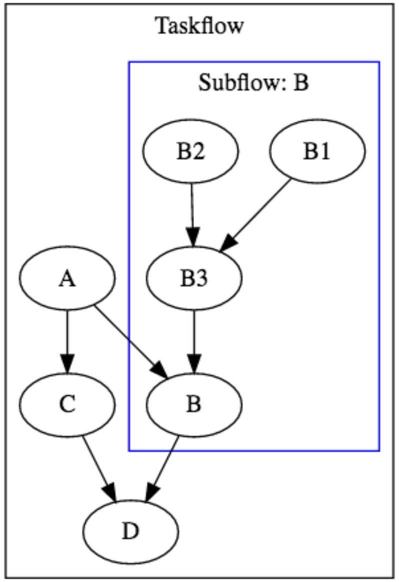
Use TBB’s FlowGraph for task parallelism

Declare a task as a continue_node

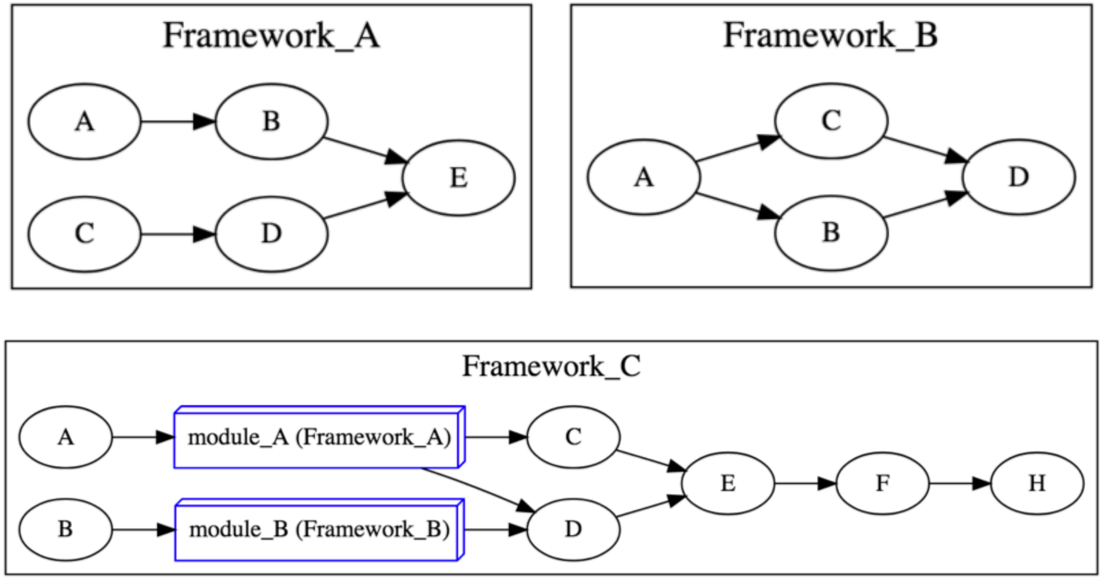
TBB has excellent performance in generic parallel computing. Its drawback is mostly in the ease-of-use standpoint (simplicity, expressivity, and programmability).

"Hello Universe" in Cpp-Taskflow

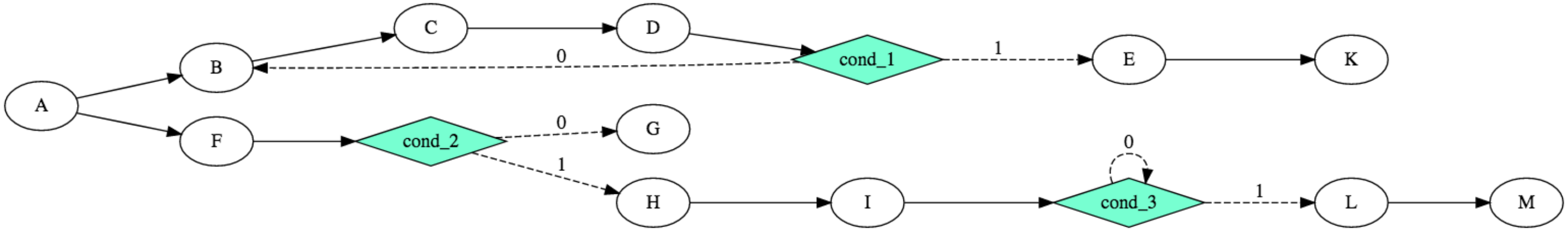
Dynamic tasking



Composable tasking for *complex workflows*



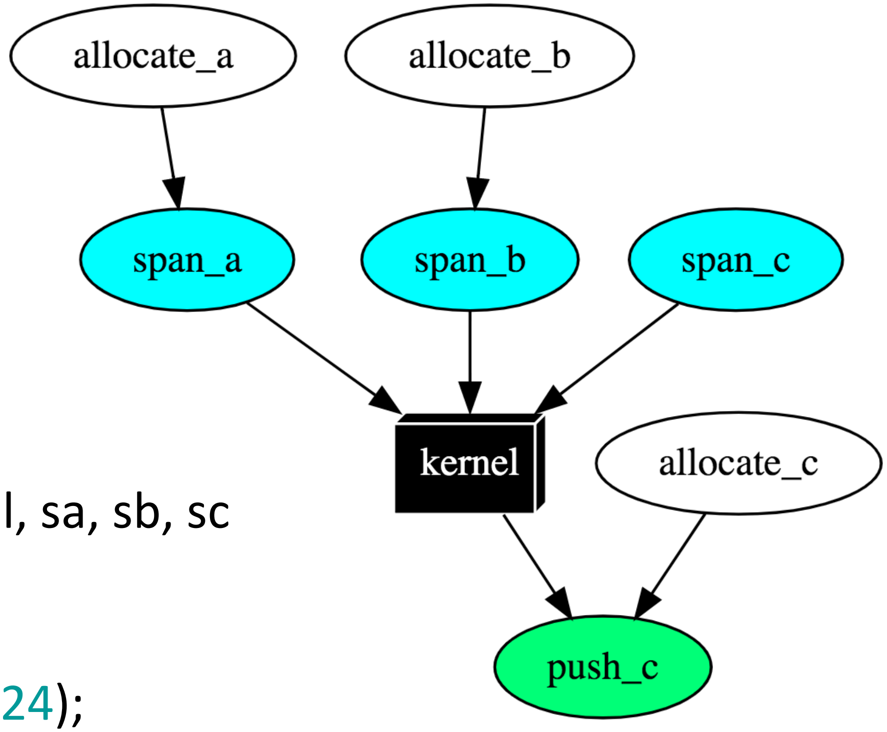
Conditional tasking for *cyclic* and *dynamic* control flows



More on: https://github.com/cpp-taskflow/cpp-taskflow

Concurrent CPU-GPU Task Programming

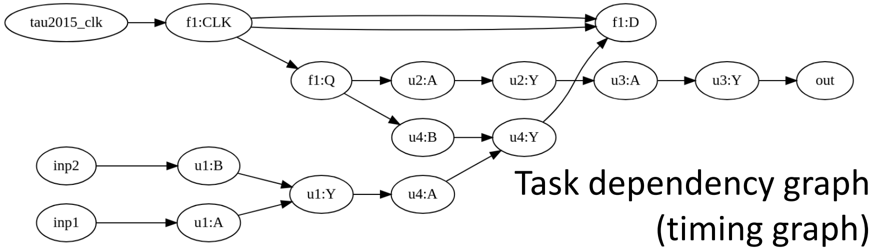
```
auto ha = hf.host([](){});
auto hb = hf.host([](){});
auto hc = hf.host([](){});
auto sa = hf.span(1024);
auto sb = hf.span(1024);
auto sc = hf.span(1024);

auto op = hf.kernel(
  {(1024+32-1)/32}, 32, 0, fn_kernel, sa, sb, sc
);

auto cc = hf.copy(host_data, sc, 1024);

ha.precede(sa);
hb.precede(sb);
op.succeed(sa, sb, sc).precede(cc);
cc.succeed(hc);
```



kernel is non-trivial, but what makes heterogeneous programming difficult is the "surrounding tasks"

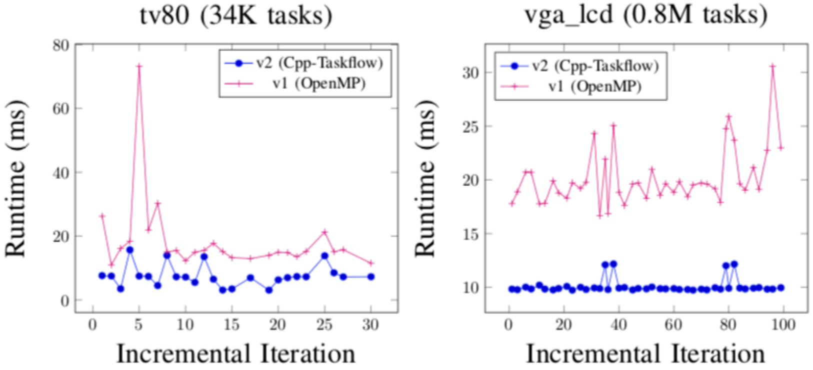
Real Use Case: VLSI Timing Analysis

- □ **OpenTimer v1: A VLSI Static Timing Analysis Tool**
  - □ v1 first released in 2015 (open-source under GPL)
  - □ Loop-based parallelisms using OpenMP 4.0
- □ **OpenTimer v2: A New Parallel Incremental Timer**
  - □ v2 first released in 2018 (open-source under MIT)
  - □ Task-based parallel decomposition using Cpp-Taskflow



Task dependency graph
(timing graph)

Saved 4K lines of parallel code!
Cost to develop is $275K with OpenMP vs $130K with Cpp-Taskflow! (measured by sloccount)



v2 (Cpp-Taskflow) is 1.4-2x faster than v1 (OpenMP)

Current and Future Work

- **A general-purpose parallel task programming system**
  - Simple, efficient, and transparent using modern C++
  - Multithreading and CPU-GPU tasking
  - Real case use in VLSI timing analysis with billion-tasking
- **On-going items**
  - Developing new task models using cudaGraph
  - Developing accelerator bridge to handle other devices
  - Developing transition tools to ease the adoption
- **Open to collaboration for more use cases!!!**
  - https://github.com/cpp-taskflow/cpp-taskflow
  - https://github.com/tsung-wei-huang/DtCraft