digraph Taskflow { "A" -> "B" "A" -> "C" "A" -> "E" "B" -> "D" "B" -> "E" "C" -> "D" }

Cpp-Taskflow: Fast Parallel Programming with Task Dependency Graphs

Tsung-Wei Huang, Chun-Xun Lin, Guannan Guo, Martin D. F. Wong

Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign

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

ILLINOIS

Introduction

Cpp-Taskflow is a **zero-dependency & header-only** 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.

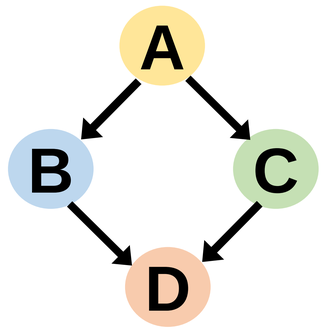
Parallel Programming with Cpp-Taskflow in 3 steps

1. Include taskflow.hpp header.
2. Describe your parallelism as a task dependency graph.
3. Execute the Tasks!

```cpp
// Cpp-Taskflow is header-only
#include <taskflow/taskflow.hpp>

int main(){

  tf::Taskflow tf(4);  // taskflow with 4 workers

  // Add 4 tasks
  auto [A, B, C, D] = tf.silent_emplace(
    [] () { std::cout << "TaskA\n"; },
    [] () { std::cout << "TaskB\n"; },
    [] () { std::cout << "TaskC\n"; },
    [] () { std::cout << "TaskD\n"; }
  );

  A.precede(B);  // B runs after A
  A.precede(C);  // C runs after A
  B.precede(D);  // D runs after B
  C.precede(D);  // D runs after C

  tf.wait_for_all();  // block until finish

  return 0;
}
```

1 2 3

Task dependency graph



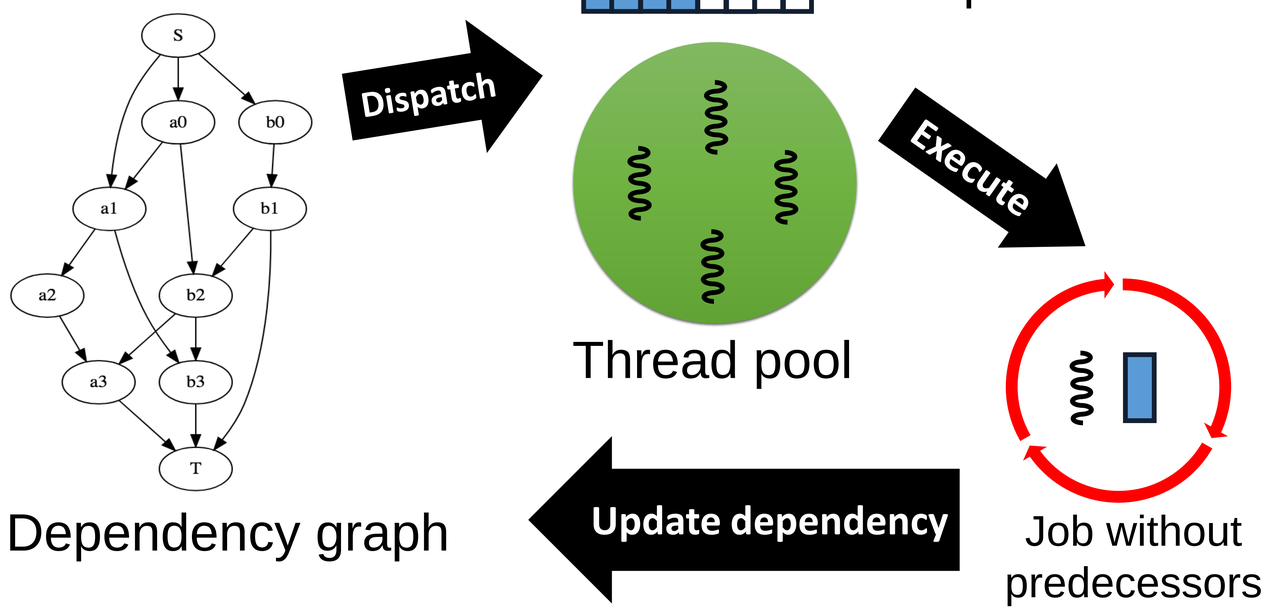
Output

```
Task A
Task B
Task C
Task D
```

or

```
Task A
Task C
Task B
Task D
```

How it works

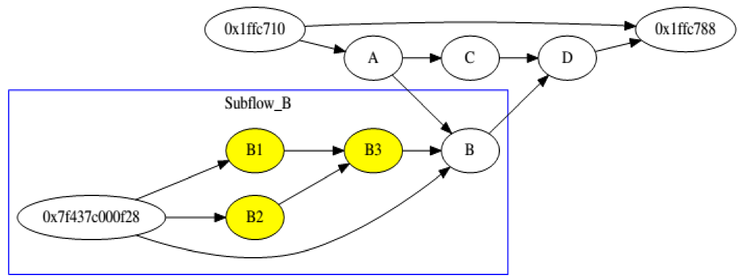


Dynamic Tasking

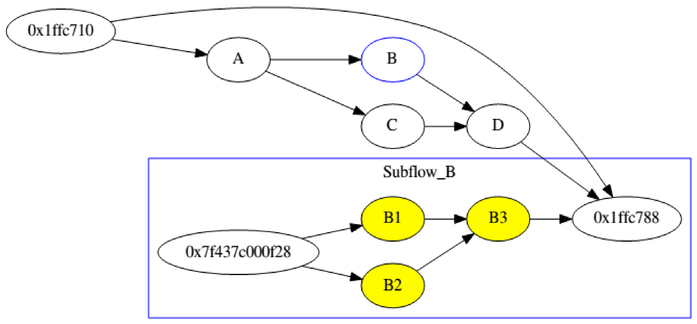
Spawn new tasks during the execution of a dispatched taskflow graph!

```cpp
#include <taskflow/taskflow.hpp>

int main(){

  tf::Taskflow tf(8);

  // create three regular tasks
  auto A = tf.silent_emplace([](){}).name("A");
  auto C = tf.silent_emplace([](){}).name("C");
  auto D = tf.silent_emplace([](){}).name("D");

  // create a subflow graph (dynamic tasking)
  auto B = tf.silent_emplace([] (auto& subflow) {
    auto B1 = subflow.silent_emplace([](){}).name("B1");
    auto B2 = subflow.silent_emplace([](){}).name("B2");
    auto B3 = subflow.silent_emplace([](){}).name("B3");
    B1.precede(B3);
    B2.precede(B3);
  }).name("B");

  A.precede(B);  // B runs after A
  A.precede(C);  // C runs after A
  B.precede(D);  // D runs after B
  C.precede(D);  // D runs after C

  // execute the graph without cleanning up topologies
  tf.wait_for_all();

  return 0;
}
```

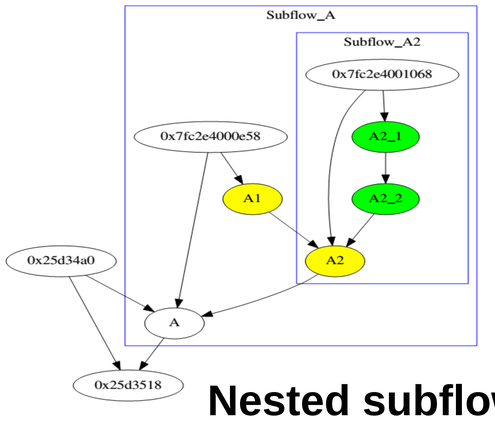
1. Emplace a callable with one input argument of type **tf::SubflowBuilder**
2. Insert new tasks into the subflow
3. Set the subflow to be joined or detached with parent node.
4. Subflow can be nested.



Joined subflow (default)



Detached subflow



Nested subflows

Graph API

Name	Description
parallel_for	apply the callable in parallel and group-by-group to the result of dereferencing every iterator in the range
linearize	create a linear dependency in the given task list
gather	enable this task to run after the given tasks
broadcast	enable this task to run before the given tasks
reduce	reduce a range of elements to a single result through a binary operator
transform_reduce	apply a unary operator to each element in the range and then reduce them to a single result through a binary operator