A Modern C++ Parallel Task Programming Library

GitHub: https://github.com/cpp-taskflow
Docs: https://cpp-taskflow.github.io/cpp-taskflow/

C.-X. Lin, Tsung-Wei Huang, G. Guo, and M. Wong
University of Utah, Salt Lake City, UT, USA
University of Illinois at Urbana-Champaign, IL, USA
Cpp-Taskflow’s Project Mantra

A programming library helps developers quickly write efficient parallel programs on a manycore architecture using task-based models in modern C++

- Parallel computing is important in modern software
  - Multimedia, machine learning, scientific computing, etc.
- Task-based approach scales best with manycore arch
  - We should write tasks NOT threads
  - Not trivial due to dependencies (race, lock, bugs, etc.)
- We want developers to write parallel code that is:
  - Simple, expressive, and transparent
- We don’t want developers to manage:
  - Threads, concurrency controls, scheduling
Hello-World in Cpp-Taskflow

#include <taskflow/taskflow.hpp>  // Cpp-Taskflow is header-only
int main(){
    tf::Taskflow tf;
    auto [A, B, C, D] = tf.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
    tf::Executor().run(tf); // create an executor to run the taskflow
    return 0;
}
Hello-World in OpenMP

```c
#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;
}
```

OpenMP task clauses are static and explicit; Programmers are responsible a proper order of writing tasks consistent with sequential execution.
Hello-World in Intel’s TBB Library

```cpp
#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> D(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).
“We want to let users easily express their parallel computing workload without taking away the control over system details to achieve high performance, using our expressive API in modern C++”
Accelerating DNN Training

- 3-layer DNN and 5-layer DNN image classifier

<table>
<thead>
<tr>
<th>Cpp-Taskflow</th>
<th>OpenMP</th>
<th>TBB</th>
<th>Sequential</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
<td>CC</td>
<td>T</td>
<td>LOC</td>
</tr>
<tr>
<td>59</td>
<td>11</td>
<td>3</td>
<td>162</td>
</tr>
</tbody>
</table>

CC: cyclomatic complexity of the implementation
T: development time (in hours) by an experienced programmer

Dev time (hrs): 3 (Cpp-Taskflow) vs 9 (OpenMP)

Propagations Pipeline

- **F**: Forward prop task
- **G_i**: i-th layer gradient calc task
- **U_i**: i-th layer weight update task

Cpp-Taskflow is about 10%-17% faster than OpenMP and Intel TBB in avg, using the least amount of source code
Cpp-Taskflow is Composable

- Large parallel graphs from small parallel patterns
- Key to improving programming productivity

Describes end-to-end parallelisms both inside and outside a machine learning workflow -> less code, more powerful, and better runtime

22% less coding complexity and up to 40% faster than Intel TBB in Neural Architecture Search (NAS) applications
Large-Scale Graph Analytics

- **OpenTimer v1**: A VLSI Static Timing Analysis Tool
  - v1 first released in 2015 (open-source under GPL)
  - Loop-based parallelism 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)


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

- GitHub: [https://github.com/cpp-taskflow](https://github.com/cpp-taskflow) (MIT)
- README to start with Cpp-Taskflow in just a few mins
- Doxygen-based C++ API and step-by-step tutorials
- Showcase presentation: [https://cpp-taskflow.github.io/](https://cpp-taskflow.github.io/)

“Cpp-Taskflow has the cleanest C++ Task API I have ever seen,” Damien Hocking

“Best poster award for open-source parallel programming library,” 2018 Cpp-Conference (voted by 1K+ professional developers)
Beyond Cpp-Taskflow: Heteroflow

Concurrent CPU-GPU task programming library

```
#include <heteroflow/heteroflow.hpp>
__global__ void saxpy(int n, float a, float *x, float *y) {
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n) y[i] = a*x[i] + y[i];
}
int main(void) {
    const int N = 1<<20;
    std::vector<float> x, y;
    hf::Heteroflow hf; // create a heteroflow object
    auto host_x = hf.host([&](){ x.resize(N, 1.0f); });
    auto host_y = hf.host([&](){ y.resize(N, 2.0f); });
    auto pull_x = hf.pull(x);
    auto pull_y = hf.pull(y);
    auto kernel = hf.kernel(saxpy, N, 2.0f, pull_x, pull_y)
        .shape((N+255)/256, 256);
    auto push_x = hf.push(pull_x, x);
    auto push_y = hf.push(pull_y, y);
    host_x.precede(pull_x); // host_x to run before pull_x
    host_y.precede(pull_y); // host_y to run before pull_y
    kernel.precede(push_x, push_y).succeed(push_x, push_y);
    hf::Executor().run(hf).wait(); // create an executor to run the graph
} Only 20 lines of code to enable parallel CPU-GPU task execution!
✓ No device memory controls
✓ No manual device offloading
✓ No explicit CPU-GPU synchronization
✓ No hardcoded scheduling
```
Thank You All (Users + Sponsors) 😊

Cpp-Taskflow integration with LGraph (master thesis by R. Ganpati @ UCSC)

Cpp-learning’s highlight (written by Hayabusa)

Cpp-Taskflow Modern C++ Parallel Task Programming
- Simple
- Expressive
- Transparent
- Performant
- Productive

NovusCore’s World of Warcraft emulator

IDEA grant

Golden timer in ACM TAU contests

Purdue’s gds2Para

Qflow Placement & Route

VSD open-source flow

Parallel Graph Processing Systems

LNSOracle