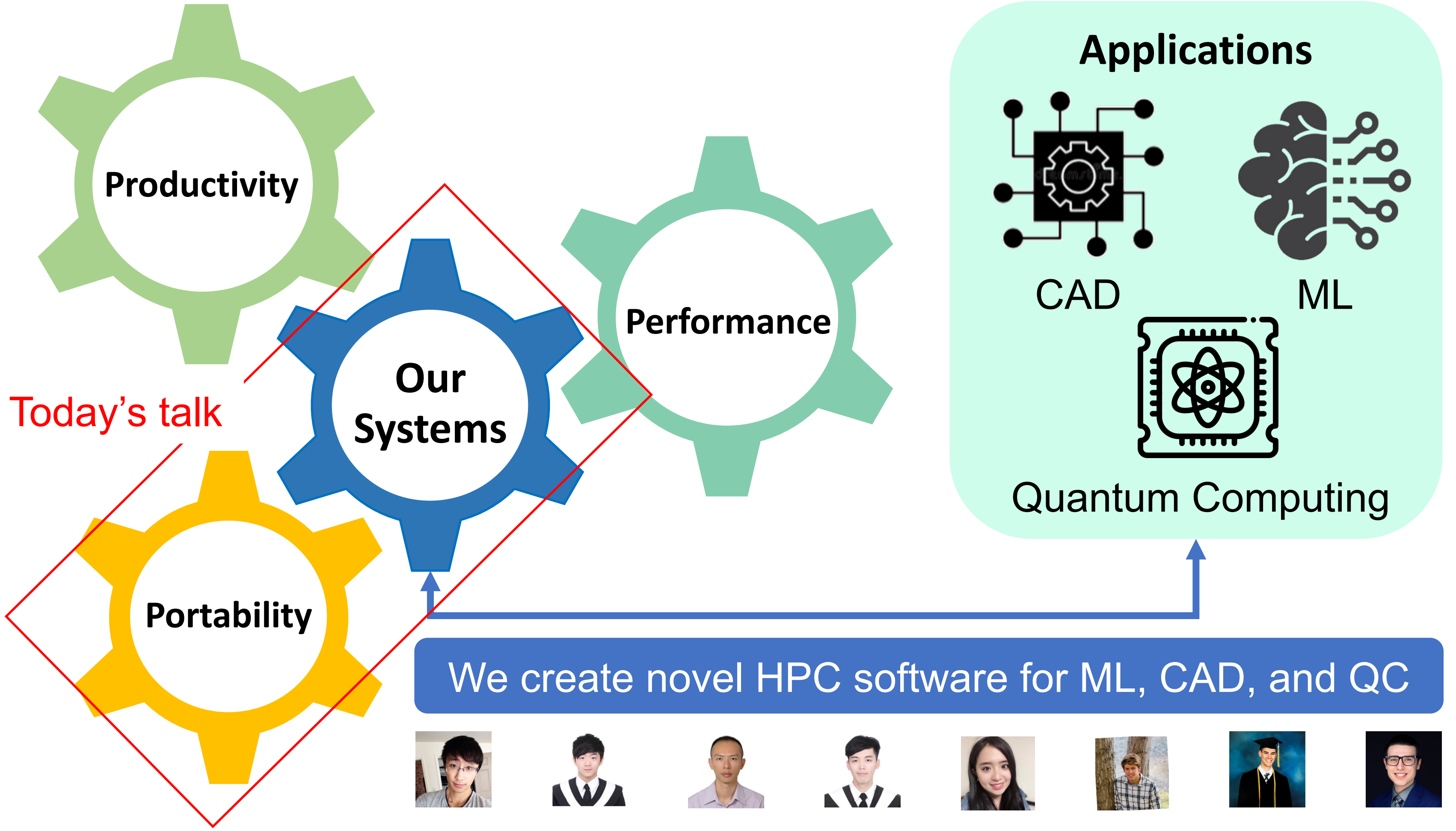
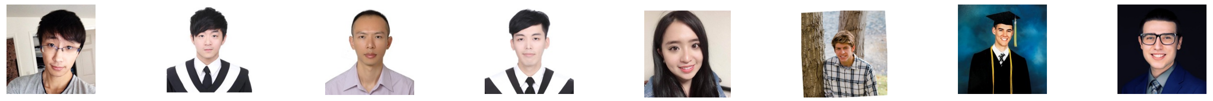
Performance Portability and Optimization using Machine Learning

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/

Overarching View of My Research



We create novel HPC software for ML, CAD, and QC



My research group: https://tsung-wei-huang.github.io/

>$2M since 2019

DARPA

NSF

NUMFOCUS
OPEN CODE = BETTER SCIENCE

Google Summer of Code

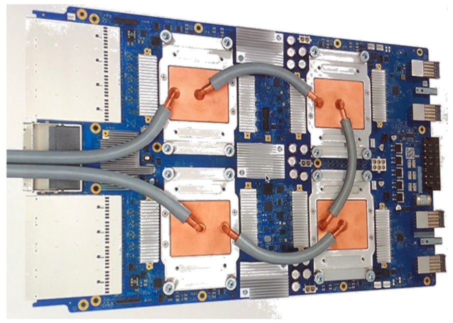
NVIDIA

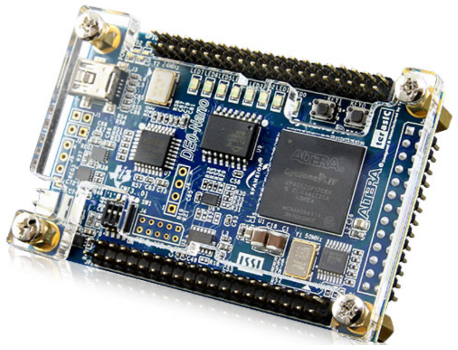
THE UNIVERSITY OF UTAH

Modern Computing Systems are Heterogeneous

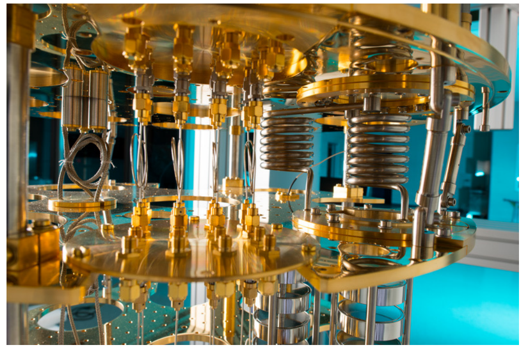
Central Processing Unit (CPU)

Graphics Processing Unit (GPU)


Tensor Processing Unit (TPU)


FPGA

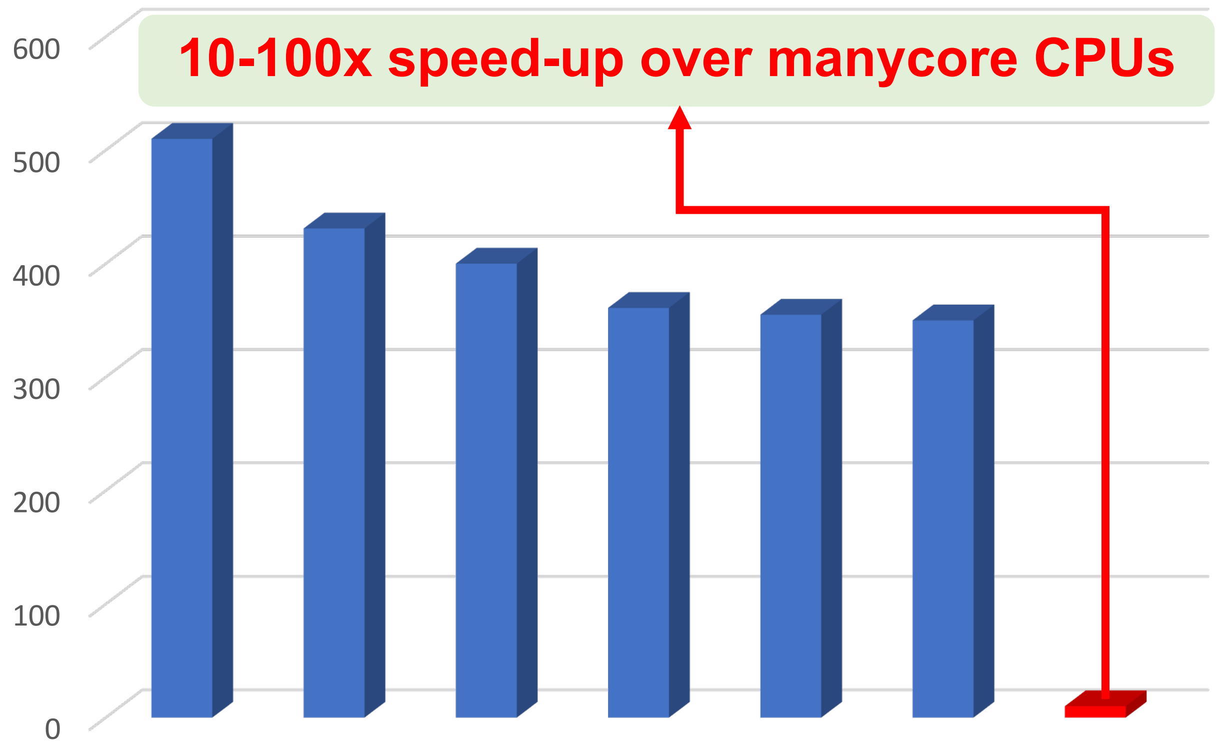
Neuromorphic Devices


Quantum Accelerator

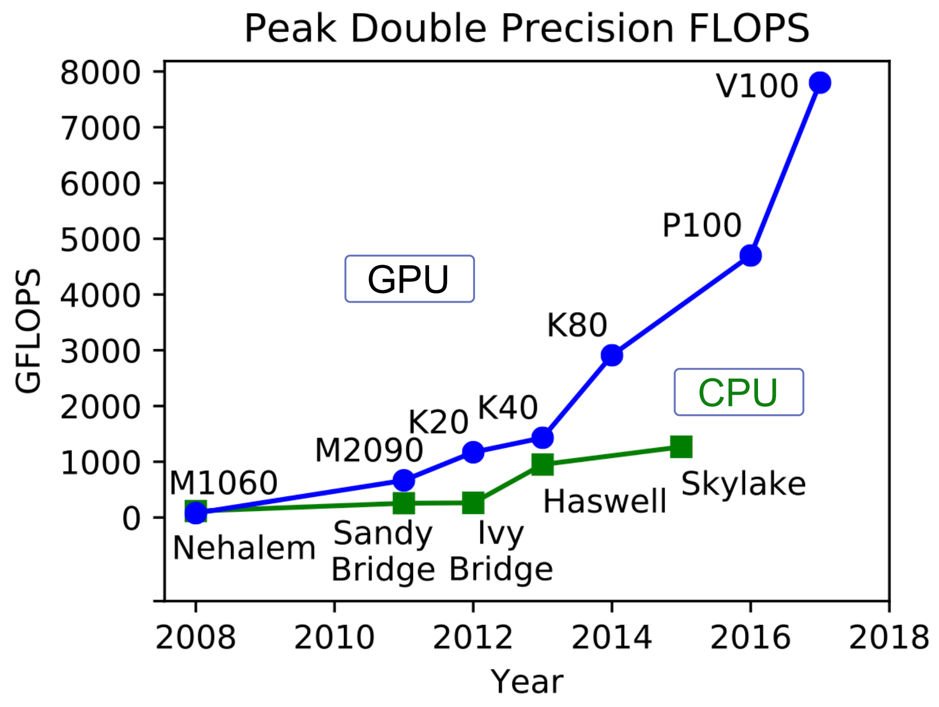
The future of computing is heterogeneous – *DARPA ERI, DOE ASCR, NSF PPoSS, SRC Jump 2.0, etc.*

Why Heterogeneous Computing (HC)?

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

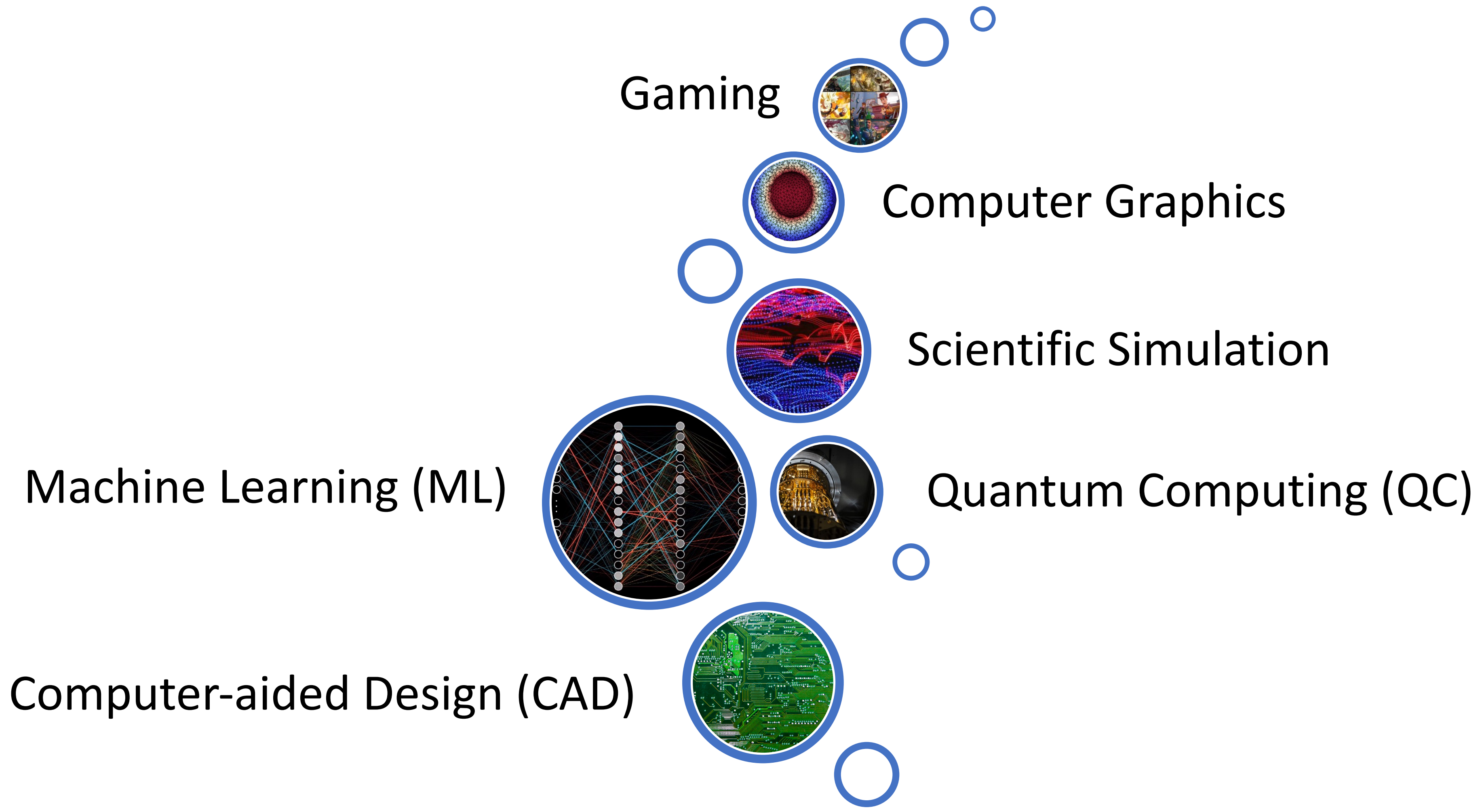


Time (minutes) to solve a machine learning workload



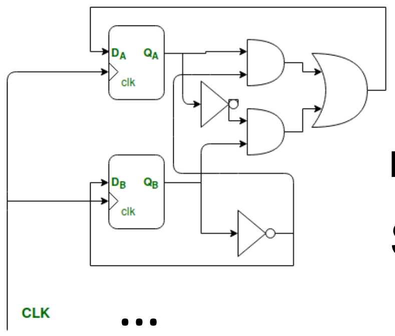
Over **60x** speedup in neural network training since 2013

HC Enabled Vast Success in Computing



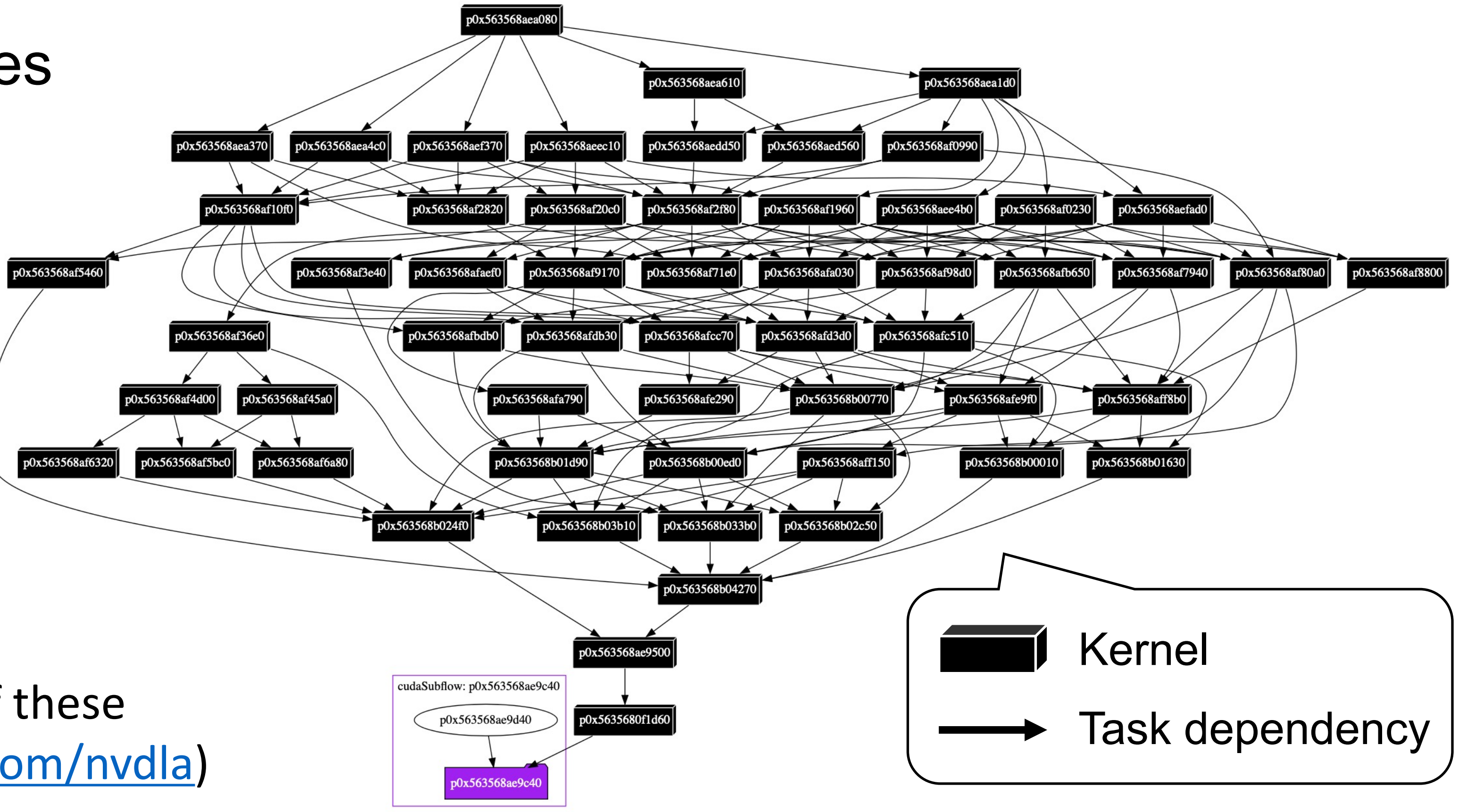
New HC Workloads are Very Complex …

- **GPU-accelerated circuit analysis on a design of <u>500M</u> gates**
  - >100 kernels
  - >100 dependencies
  - >500s to finish
  - >10 hrs turnaround



…

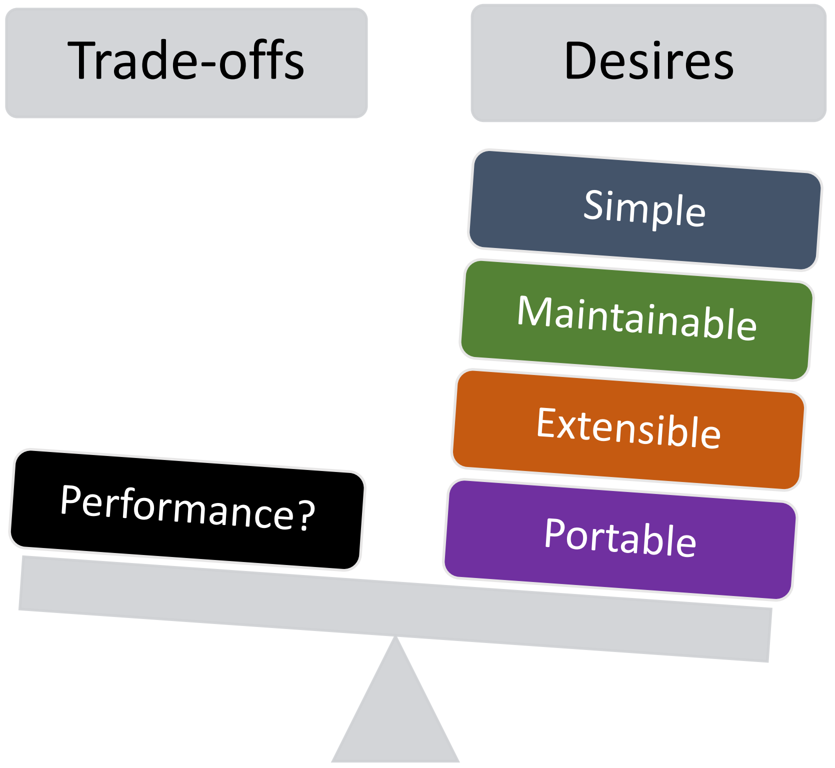
Simulation



What are the output values of these 500M gates? (https://github.com/nvdla)

Programming is a "Big" Challenge

- **You need to deal with A LOT OF technical 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



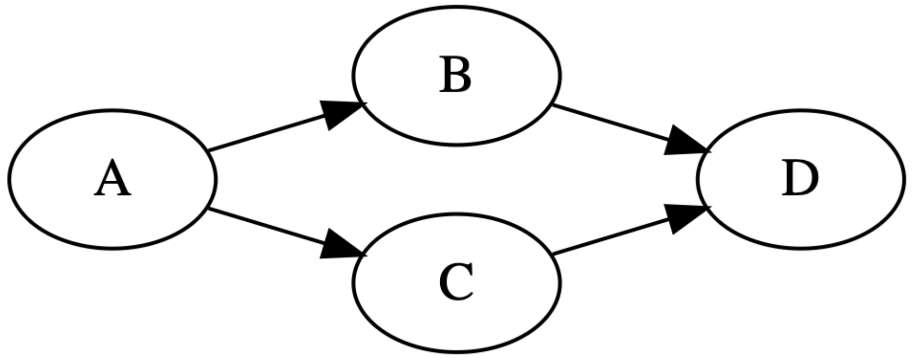
Need Help from Programming Systems

- **The hurdle for widespread adoption is programming difficulty**
  - Prioritize ease of use – *want expressive, transparent programming models*
  - Maintain a single code base – *write once and run everywhere*
  - Adapt performance to different architectures – *optimize intelligently*
- **Task-based programming fits best the need of HC systems**
  - Enable top-down optimization that scales to many processing units
  - Standards are evolving towards task parallelism
- **Plenty of challenges remain unsolved …**
  - New applications are driving new tasking models
  - We must value performance portability
  - Sustainability over hardware generations

StarPU PaRSEC TBB ECP OpenMP kokkos

Our DARPA ERI Project[1,2]: Taskflow

```
#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;
}
```


$$\begin{tikzcd} & B \arrow[dr] & \\ A \arrow[ur] \arrow[dr] & & D \\ & C \arrow[ur] & \end{tikzcd}$$

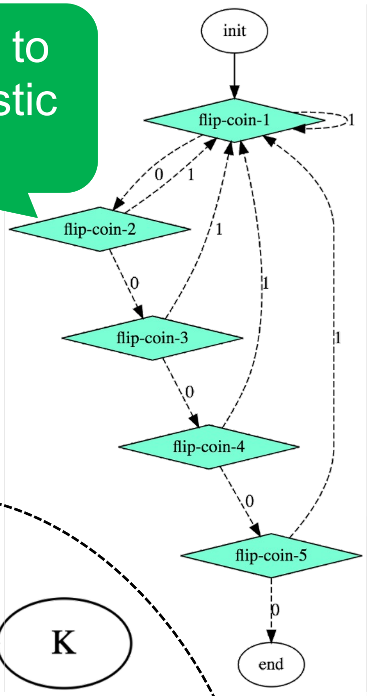
[1]: "OpenTimer and DtCraft," $427K, 06/2018-07/2019, DARPA Intelligent Design of Electronic Assets (IDEA) Program, FA 8650-18-2-7843

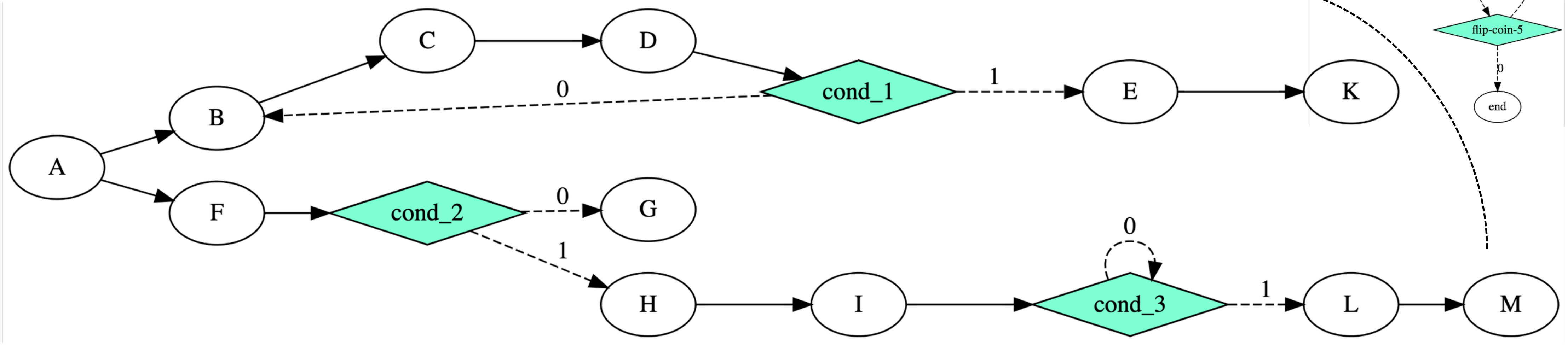
[2]: "A General-purpose Parallel and Heterogeneous Task Graph Computing System for VLSI CAD," $403K, 10/2021—10/2024, NSF CISE, CCF-2126672

Control Taskflow Graph (CTFG) Programming

```
auto cond_1 = taskflow.emplace([](){ return decision1(); });
auto cond_2 = taskflow.emplace([](){ return decision2(); });
auto cond_3 = taskflow.emplace([](){ return decision3(); });
// embed in-graph control flow through task dependencies
cond_1.precede(B, E);          // return 0 to B or 1 to E
cond_2.precede(G, H);          // return 0 to G or 1 to H
cond_3.precede(cond_3, L);  // return 0 to cond_3 or 1 to L
```

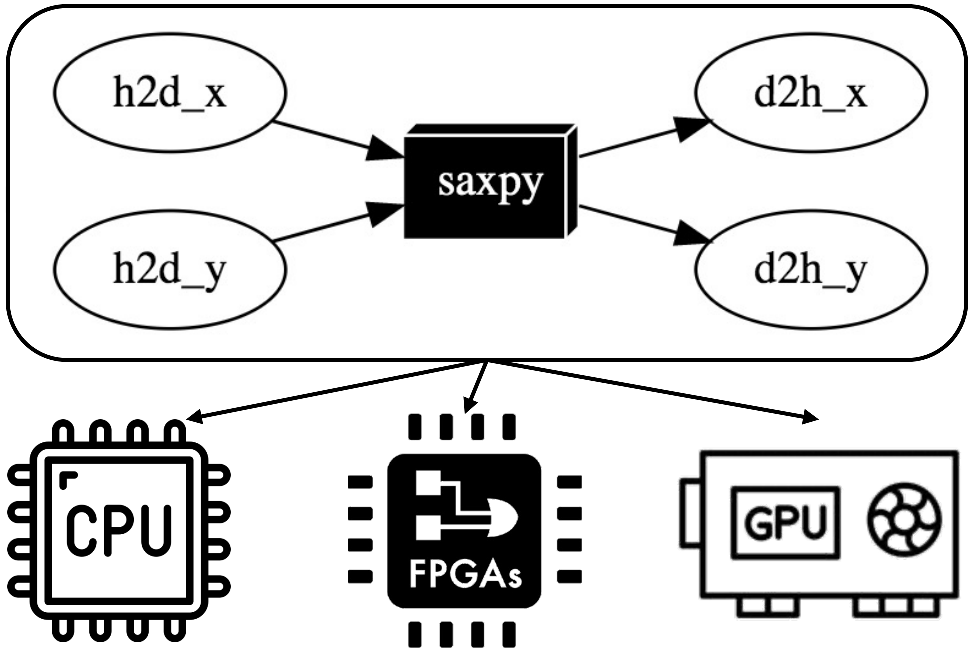
Generalizable to non-deterministic control flow





Single-Source Heterogeneous Tasking

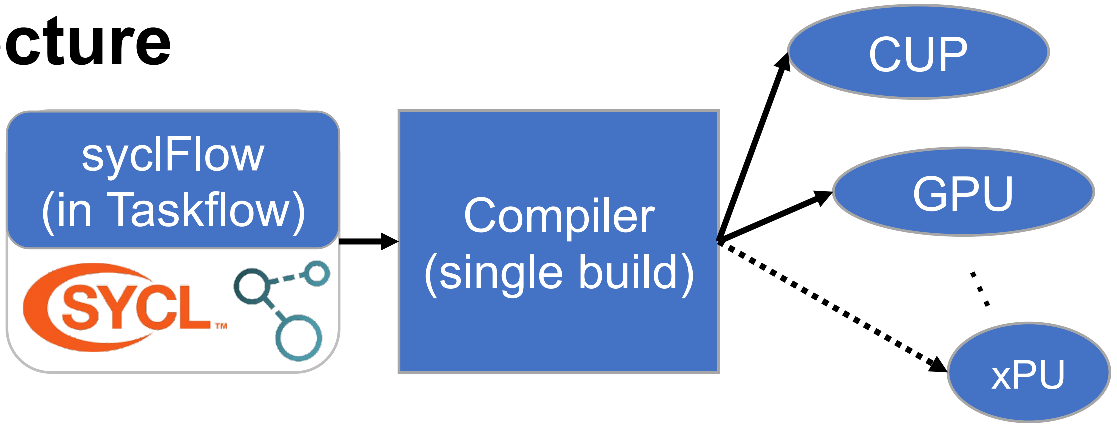
```
taskflow.emplace_on([&](tf::syclFlow& sf) {
  auto h2d_x = cf.copy(dx, hx.data(), N);
  auto h2d_y = cf.copy(dy, hy.data(), N);
  auto d2h_x = cf.copy(hx.data(), dx, N);
  auto d2h_y = cf.copy(hy.data(), dy, N);
  auto kernel = cf.parallel_for(
   sycl::range<1>(N), [=](sycl::id<1> id){
    dx[id] = 2.0f * dx[id] + dy[id];
   }
  );
  kernel.succeed(h2d_x, h2d_y)
        .precede(d2h_x, d2h_y);
}, queue);
```



Single-source program to offload to any SYCL device (e.g., CPU, GPU, FPGA) through an SYCL queue

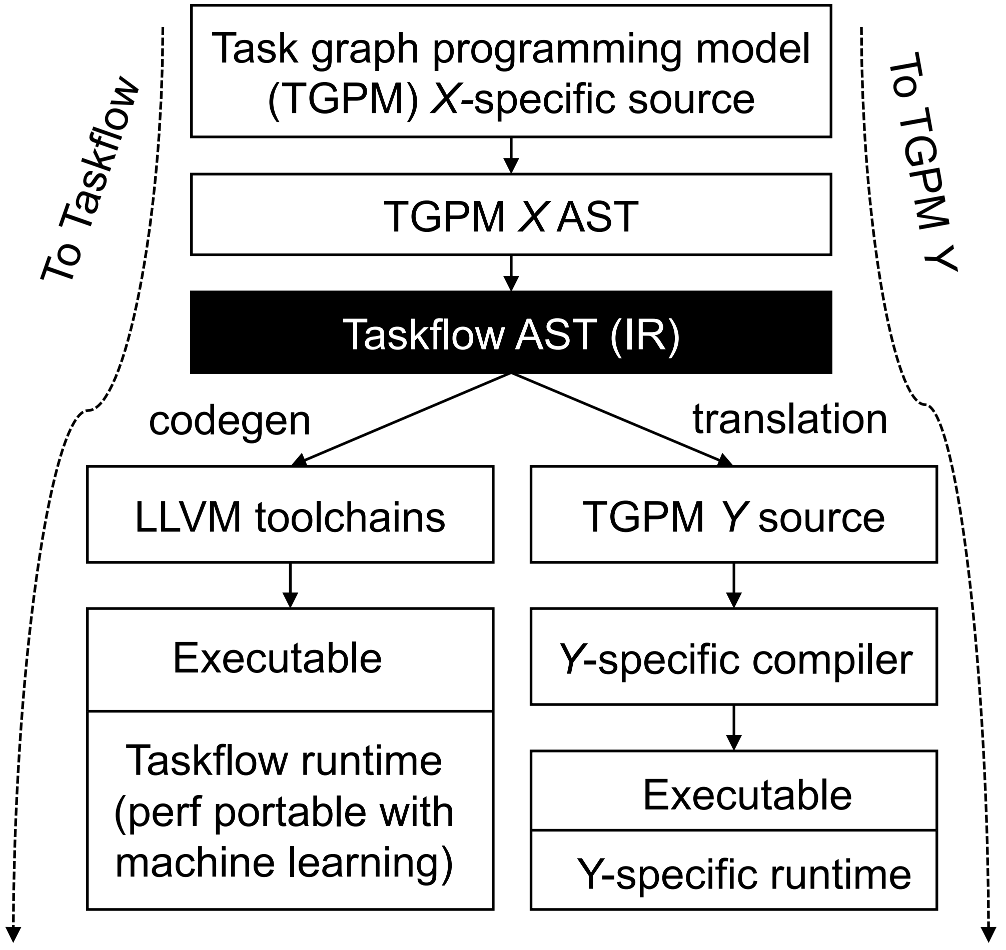
Why SYCL in Taskflow?

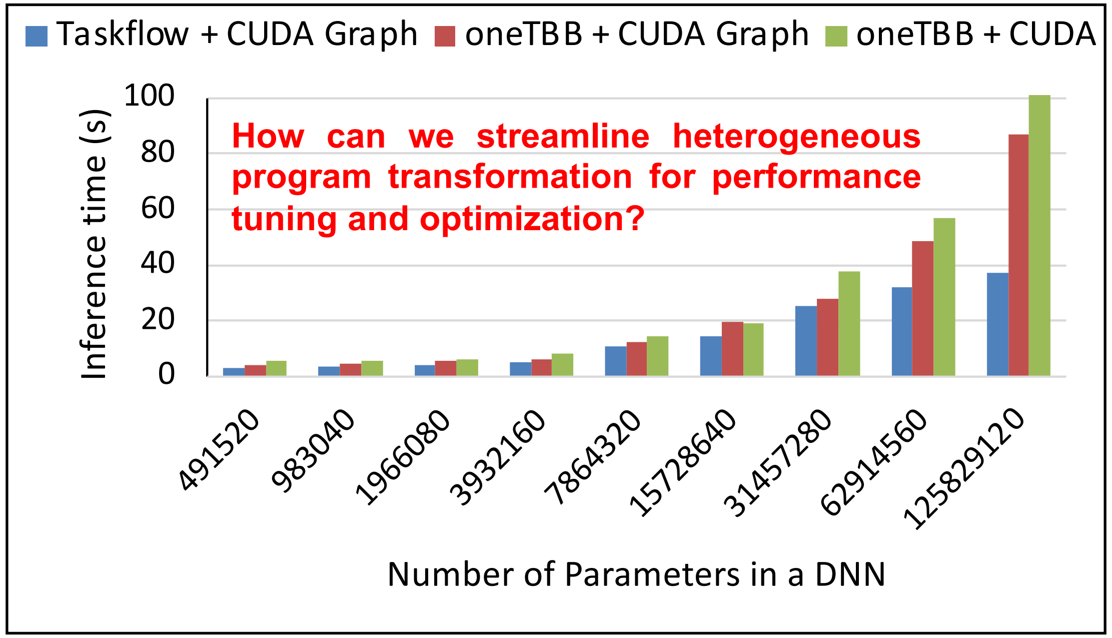
- **Software programming *cannot* be too heterogeneous!**
  - Cost(Software) >>> Cost(Hardware)
- **Single-source heterogeneous programming is the way forward**
  - SYCL enables full heterogeneous computing using *completely standard C++*
  - SYCL-aware compilers create executables for arbitrary architectures
  - New optimization opportunities for performance portability
    - Ex: machine learning to learn complex parameters
- **Open, Multivendor, Multiarchitecture**
  - Standardize "SYCL Graph"
    - Inspired by CUDA Graph
  - Serve on SYCL Advisory Panel
    - Chaired by Michael Wong



Our NSF OAC Project[1]: Taskflow Compiler

- **Single source streamlines performance optimization**



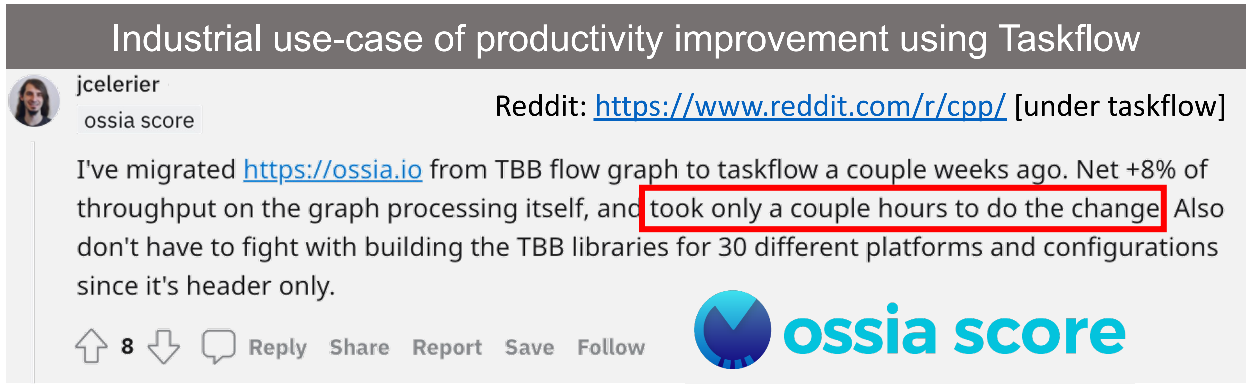


How can we streamline heterogeneous program transformation for performance tuning and optimization?

[1]: "Transpass: Transpiling Parallel Task Graph Programming Models for Scientific Software," $488K, 07/2022—07/2025, NSF CISE Core, OAC-2209957

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!**



Industrial use-case of productivity improvement using Taskflow

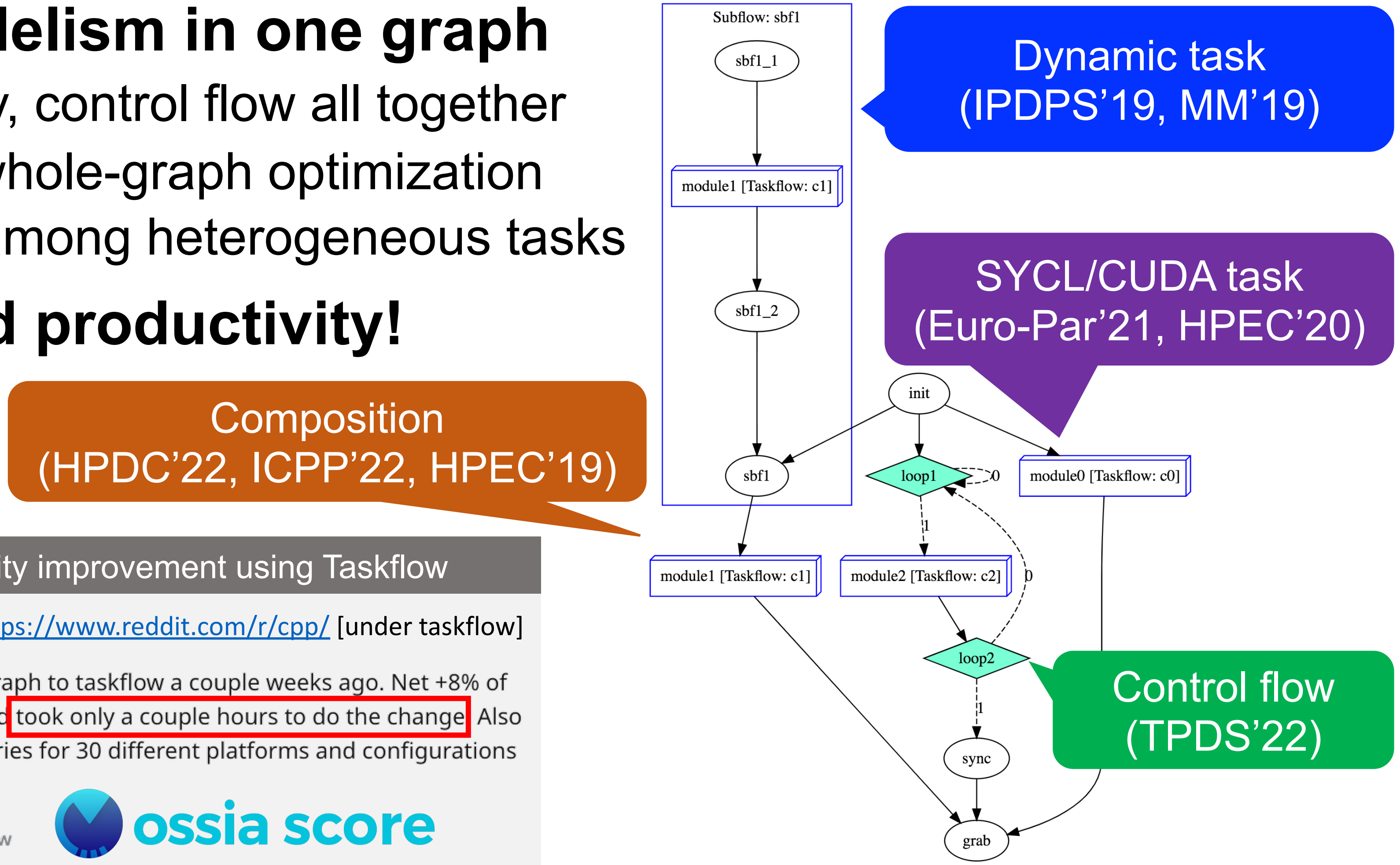
jcelerier
ossia score

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

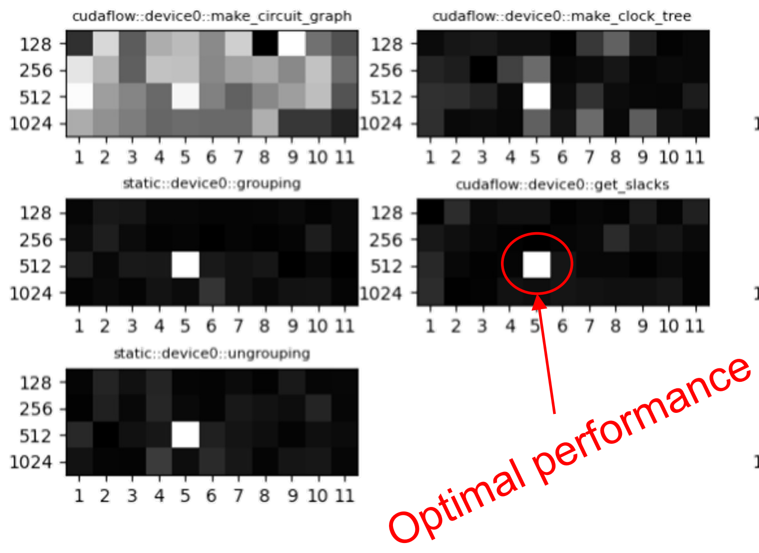
ossia score



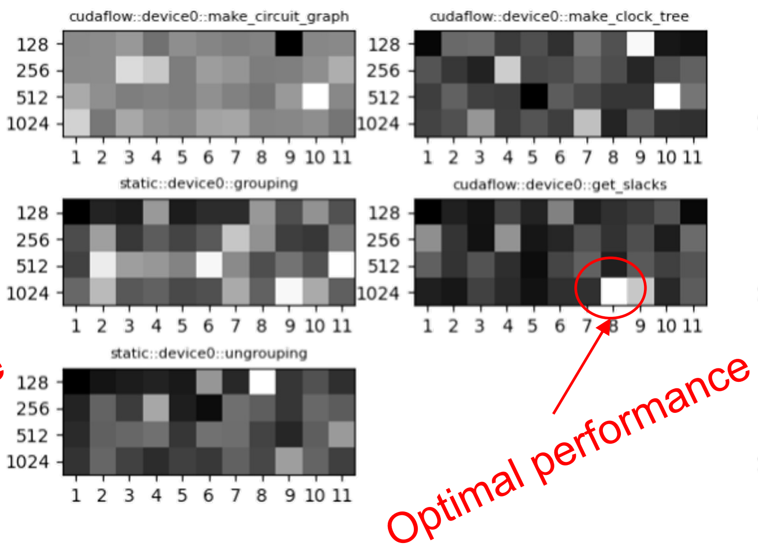
Performance Portability (HPEC'22)

- **Single-source programming enables "code" portability**
  - Same C++ kernel code runs on different architectures
- **But, the performance on different architectures varies a lot …**
  - Ex: up to 41% runtime difference for the same circuit analysis program

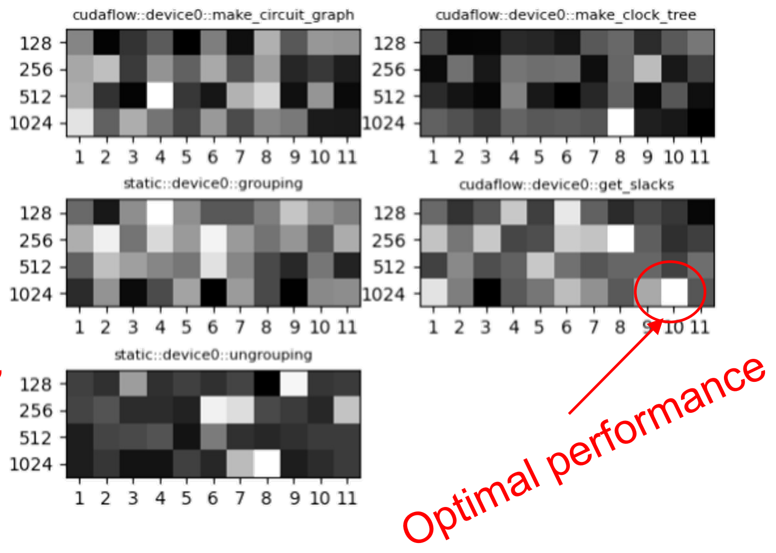
Intel CPU + Nvidia GPU (311s)



AMD CPU + AMD GPU (438s)

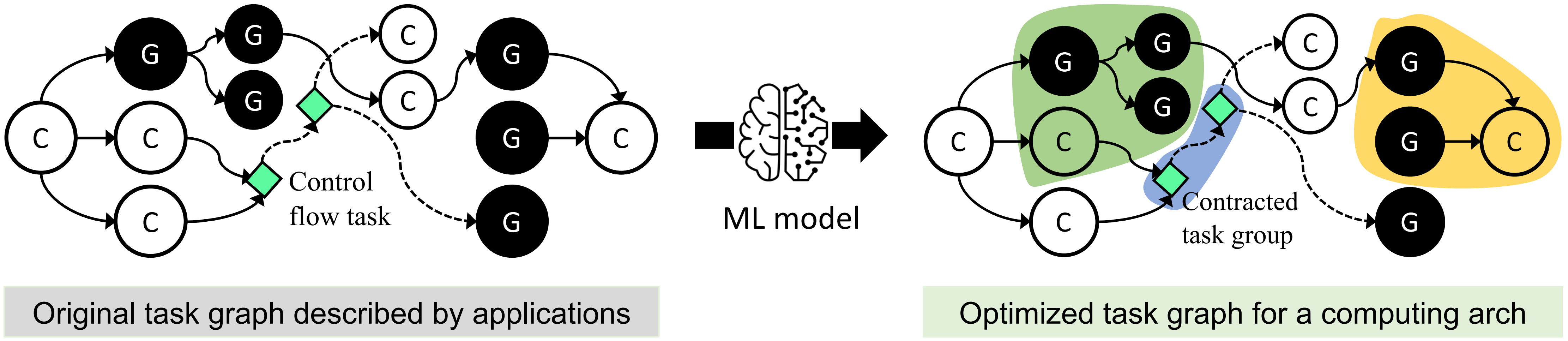


Intel CPU + Intel GPU (397s)



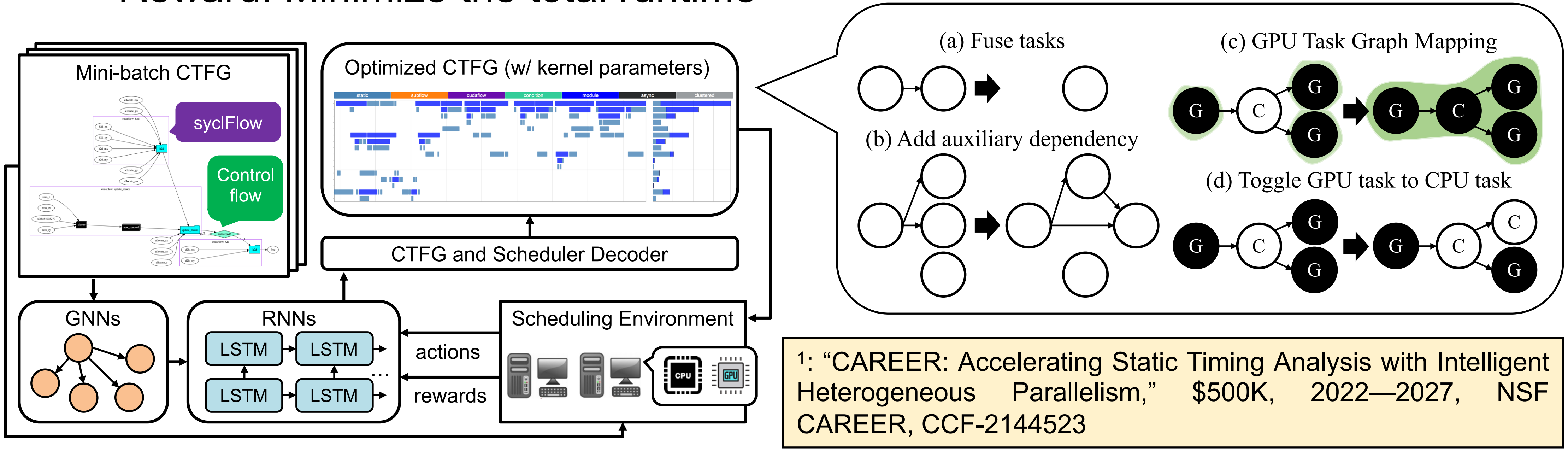
Is Performance Portability Impossible?

- **No! Achieving performance portability is *highly parameterizable***
  - Massive parameter space (block/thread size, task graph structures, etc.)
- **Highlights the need for novel learning-based methods**
  - Learn to optimize a task graph to boost scheduling performance
  - Adapt performance optimization to any computing environment


G
C
Control flow task
ML model
Contracted task group
Original task graph described by applications
Optimized task graph for a computing arch

Our NSF CAREER Project[1]: RL-based Runtime

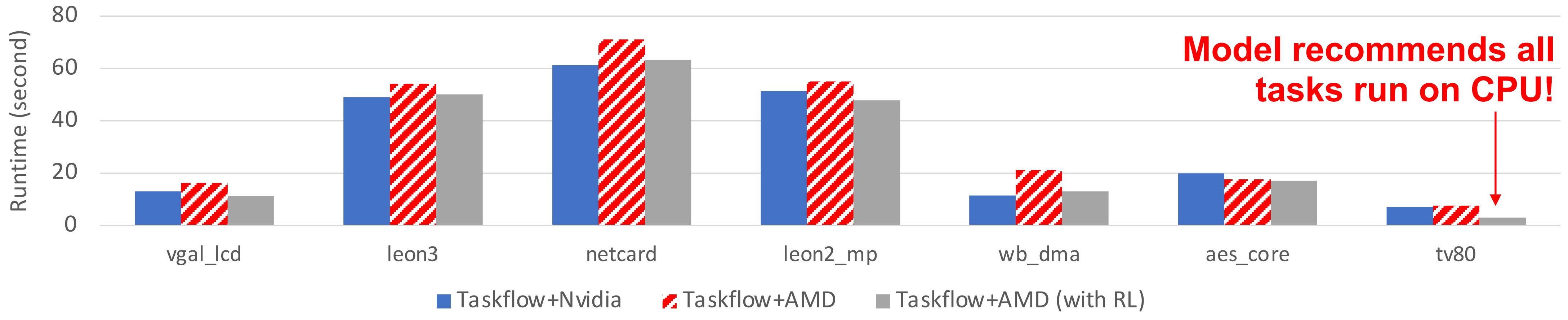
- **Leverage reinforcement learning for performance optimization**
  - Environment: Taskflow's scheduler running on a user computing platform
  - Action: Control taskflow graph (CTFG) modifiers to optimize graph structure
  - Network: GNN to learn CTFG structure and RNN to learn scheduling impact
  - Reward: Minimize the total runtime


Mini-batch CTFG
syclFlow
Control flow
Optimized CTFG (w/ kernel parameters)
CTFG and Scheduler Decoder
GNNs
RNNs
LSTM
LSTM
LSTM
LSTM
actions
rewards
Scheduling Environment
CPU
GPU
(a) Fuse tasks
(b) Add auxiliary dependency
(c) GPU Task Graph Mapping
(d) Toggle GPU task to CPU task
G
C

[1]: "CAREER: Accelerating Static Timing Analysis with Intelligent Heterogeneous Parallelism," $500K, 2022—2027, NSF CAREER, CCF-2144523

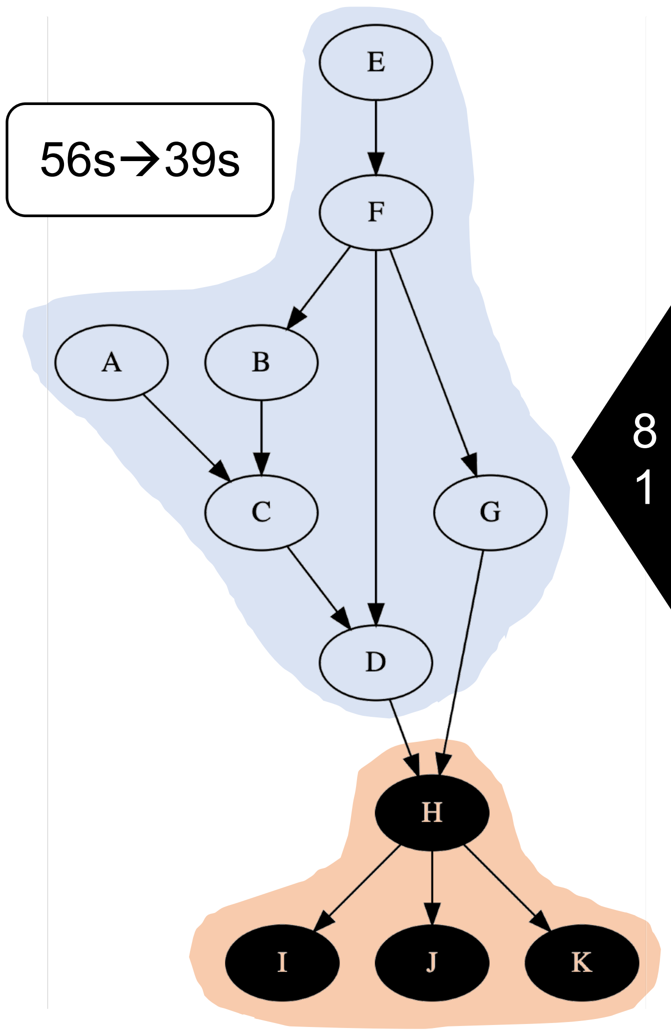
Result on Circuit Timing Analysis

- **Measured on two GPU architectures: Nvidia RTX vs AMD RX**
  - Baseline written with SYCL and tested on Nvidia GPU
  - The same code/program incurs **7-41%** performance variation
- **RL-based adaptor infers the best graph parameters for AMD RX**
  - 7-36% toggled tasks (action #4) – *small circuits*
  - 10-21% reduced CTFG (action #1) – *clock trees and linear segments*

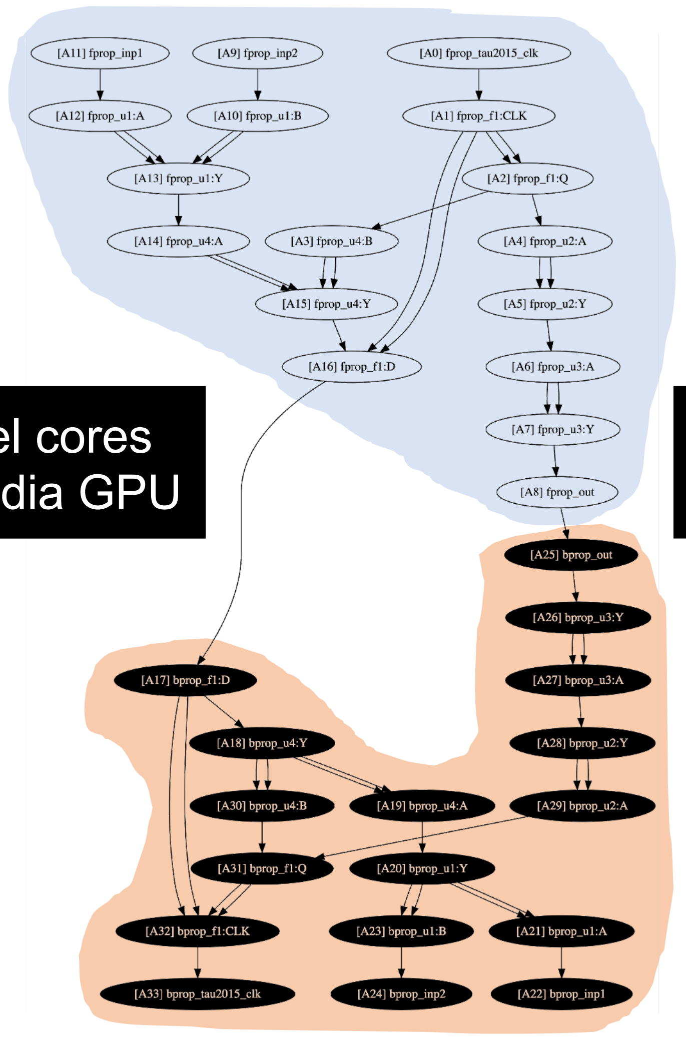


Result on Circuit Timing Analysis (cont'd)

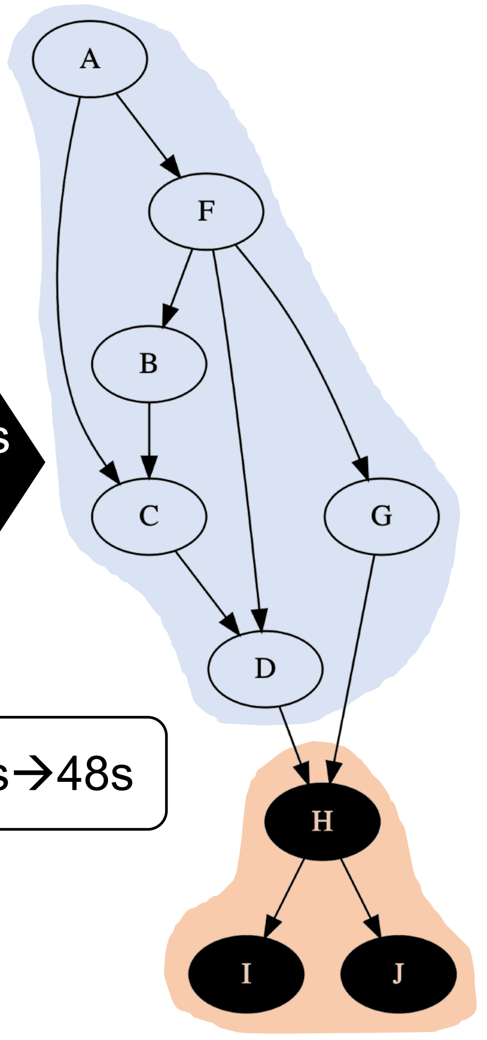
56s→39s



8 Intel cores
1 Nvidia GPU



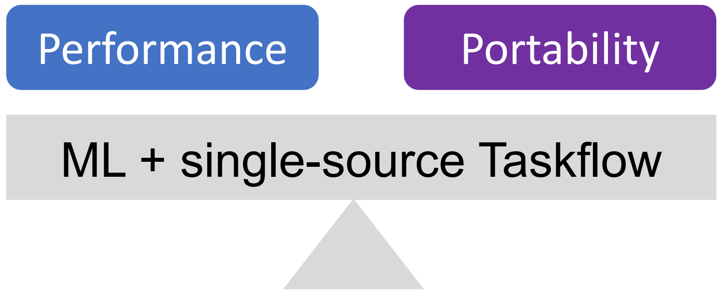
4 AMD cores
1 AMD GPU



56s→48s

Insight from RL-based Optimization

- **Performance portability is possible with ML and single source**
  - Restructure task graphs for the right granularity and data locality
  - Infer the right performance parameters
- **Outperform general-purpose heuristics!**
- **However, the cost of ML is non-negligible**
  - 7-11% performance eaten by ML itself
    - Feature vector generation (GNN, RNN)
    - Inference
- **Plenty of research opportunities**
  - Discover efficient neural network architectures
  - Improve the sample/action space efficiency



U.S. DEPARTMENT OF ENERGY | Office of Science

FY 2022 ASCR Open Call and CAREER programs (computer science): Programming Models, Environments, and Portability

Taskflow Open-Source User Community

- https://taskflow.github.io/ (5-8K downloads / week)

Star 7,360

OT ossia deal.II MYDATAMODELS XANADU LiveHD

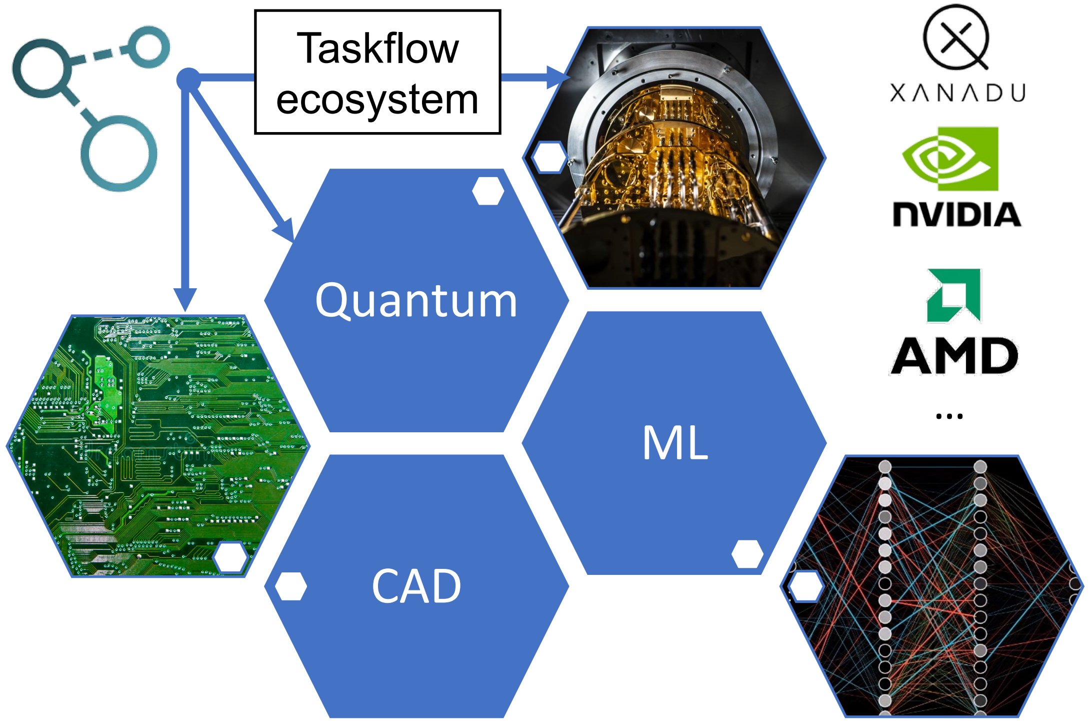
ROS industrial REVEAL SNIG Inference Engine 24h OPERON

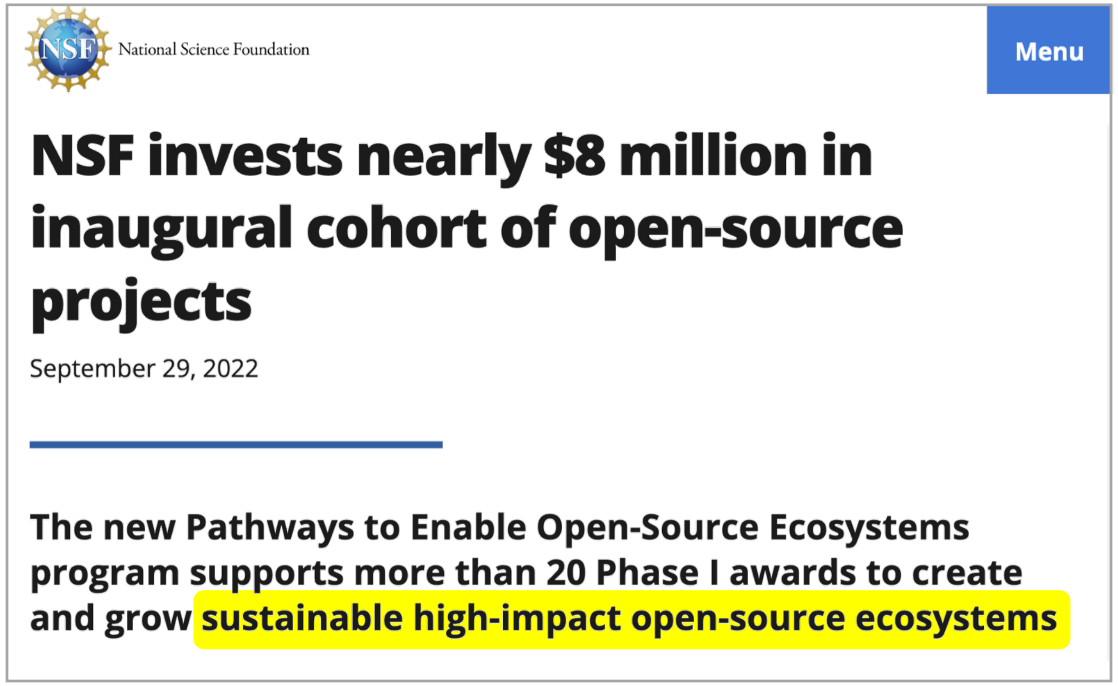
R NVIDIA. Explosion METHANE powered graphics Chainblocks Rapid Fuzz JET BRAINS RESEARCH

atomicDEX ReAgent RAPIDS XILINX

Our NSF POSE Project[1]: Sustainability

- Applications expect programming systems to last for 10 years




National Science Foundation
Menu

NSF invests nearly $8 million in inaugural cohort of open-source projects

September 29, 2022

The new Pathways to Enable Open-Source Ecosystems program supports more than 20 Phase I awards to create and grow sustainable high-impact open-source ecosystems

https://beta.nsf.gov/tip/updates/nsf-invests-nearly-8-million-inaugural-cohort-open

1: "POSE: Phase I: Toward a Task-Parallel Programming Ecosystem for Modern Scientific Computing," $298K, 09/15/2022—08/31/2023, NSF POSE, TI-2229304

Conclusion

- **We have presented our Taskflow programming system**
  - Simple, expressive, and transparent
  - Single-source heterogeneous tasking using SYCL
- **We have presented our learning-based runtime**
  - Adaptive performance optimization
  - Performance portability using reinforcement learning
- **We are very open to collaboration!**

Use the right tool for the right job

Taskflow: https://taskflow.github.io

Thank You

Dr. Tsung-Wei Huang

tsung-wei.huang@utah.edu