High-Performance Task Scheduling for Heterogeneous Systems

Jingxuan Zhang, Peiran Hou

April 2026

URL: `https://rulihongran.github.io/15618-project-webpage/`

1 Project Overview

We propose to develop a task scheduler for directed acyclic graphs (DAGs) that optimizes execution across heterogeneous CPU and GPU resources. The core objective is to minimize the total completion time of complex task sets by balancing computational speedups against data transfer overheads. We will implement a custom heuristic cost model that estimates task duration based on computational workload, data size, CPU/GPU processing rates, and PCIe bandwidth to make informed scheduling decisions.

2 Progress Summary

Up to now, We have implemented a basic end-to-end DAG scheduler. The system has several core components: a DAG representation layer, a scheduler, an executor, and an execution engine, along with a profiling monitor for performance instrumentation.

On the scheduling side, we integrated five static scheduling algorithms alongside a greedy-based dynamic scheduler, and explored both static and online cost models to estimate task execution and data transfer times.

For evaluation, we designed five categories of synthetic task graphs and one application-specific benchmark (Cholesky decomposition) to test the scheduler under varied workloads. We measured four key metrics including makespan, Critical-path-to-Communication Ratio (CCR), cost estimation error, and CPU/GPU utilization to evaluate quality across all strategies.

3 Goals and Deliverables Update

3.1 Original Goals

Our project consists of three main components: system design, performance optimization, and evaluation.

- **System Design:**
  - Build a heuristic, list-based DAG scheduler capable of parsing a DAG and dispatching tasks to either a CPU thread pool or a CUDA stream based on a static cost model.
  - Implement a HEFT algorithm that assigns weights to nodes based on average execution time and edges based on data transfer costs.
  - Incorporate a static cost model to guide scheduling decisions, enabling the system to balance computation and data transfer overhead across CPU and GPU.
- **Performance Optimization:**
  - Develop and integrate the MemoryPool to eliminate cudaMalloc calls during the critical execution path.
  - Implement a logic where an idle CPU can steal tasks from the GPU queue if the estimated PCIe transfer time makes GPU execution a net loss in real-time, improving resource utilization.

- **Evaluation:**
  - Compare against multiple baselines including CPU-only execution, GPU-only execution, and static HEFT scheduling. We aim to achieve at least 1.5×–2.0× speedup over CPU-only execution, and 1.2×–1.5× improvement over GPU-only and static HEFT, especially in workloads with heterogeneous task sizes and non-trivial PCIe transfer costs.

3.2 Updated Goals for Poster Session

Based on our original goals and actual implementation, we made the following updates:

- **Revised Cost-modeling Strategies**
  We extended our scheduling models to Static analytical model, Static learned model and Online-calibrated model.
  - **Static analytical model**: This model includes HEFT scheduler, GPU-only, CPU-only, MCT scheduler and random scheduler, all of which are depending on a static estimated cost model.
  - **Static learned model**: A static learned cost model is an offline-trained performance model that predicts task execution and transfer costs before runtime. In our system, it is trained from profiling data and then used at runtime with fixed parameters. Unlike online calibration, it does not update itself from newly observed task timings within the current run.
  - **Online-calibrated model**: An online-calibrated model adapts cost estimates during execution using newly observed task runtimes. Instead of relying only on fixed offline predictions, it updates its estimates from runtime measurements and uses those refined values to guide later scheduling decisions.
- **Expanded Benchmark and Validation Support**
  - We extended the system with a Cholesky DAG benchmark and a profiling pipeline that records per-task execution, transfer, and utilization statistics.
- **Profiling and Model-Building Pipeline**
  - We built an end-to-end pipeline for collecting profiling data, exporting SQLite/CSV artifacts, and training offline learned cost models for both computation and transfer costs. This allows us to compare analytical estimates, learned estimates, and runtime-calibrated estimates under the same execution framework.
- **Refined Evaluation Focus**
  - Instead of evaluating only final speedup, we now analyze multiple dimensions of scheduler quality, including makespan, estimation error, measured communication-to-computation ratio (CCR), CPU/GPU utilization and gantt chart of task flows. This gives a more complete view of both scheduling quality and cost-model accuracy.

4 Poster Session Plan

For the poster session, we plan to present our project as a heterogeneous DAG scheduling system that evolves from static analytical estimation to learned cost model and adaptive runtime calibration on CPU-GPU platforms. The poster will focus on both the system design and the practical impact of different cost-modeling strategies.

- **System Overview**
  We will first introduce the overall architecture of our DAG runtime, including task graph generation, scheduling, execution on CPU threads and CUDA streams, memory management, and profiling support. This part will help the audience understand how scheduling decisions are connected to execution and measurement.

- **Scheduling Model Comparison**
  We will present the three scheduling settings used in our project: static analytical model, static learned model, and online-calibrated model. We will explain how these models differ in terms of when cost information is produced, whether it changes at runtime, and how it affects task-device assignment.

- **Benchmark and Profiling Pipeline**
  We will demonstrate our Cholesky DAG benchmark and profiling pipeline, which collect per-task execution time, transfer time, and utilization statistics. This part highlights how we generate training data, validate benchmark correctness, and build a reproducible evaluation workflow.

- **Evaluation Results**
  We will show the effectiveness of our system using multiple metrics, including makespan, estimation error, measured communication-to-computation ratio (CCR), CPU/GPU utilization, and Gantt-chart-style task flow visualizations. These results will illustrate both scheduling quality and cost-model accuracy.

5 Preliminary Results

5.1 Static Scheduling Algorithms

We apply static scheduling algorithims including HEFT,MCT, Random, Cpu-only and GPU-only. The results show a clear gap between how well the schedulers perform and how accurate the cost model is. In Fig1, HEFT consistently delivers the best speedups across most DAGs, with MCT and GPU-only also doing reasonably well.

However, looking at Fig2, the analytic cost model has very large estimation errors for all methods, often several hundred percent. This means the model is not reliably capturing the real execution cost, even when the scheduling outcome looks good.

Overall, these results suggest that our current analytic cost model still needs improvement. It does not accurately reflect real performance, especially when communication and heterogeneous resources are involved, so better modeling is needed to make scheduling decisions more reliable.

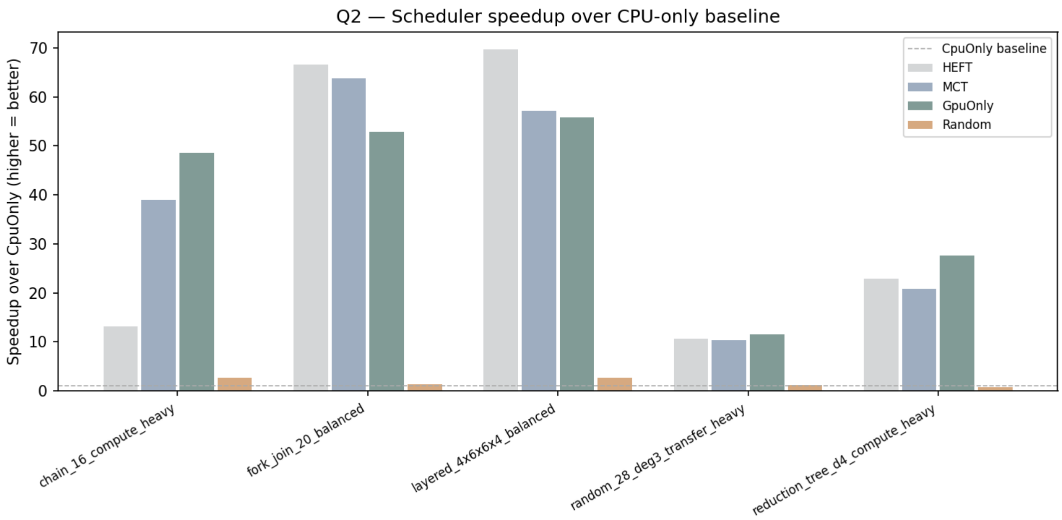

Figure 1: Static Scheduler Makespan Speedup

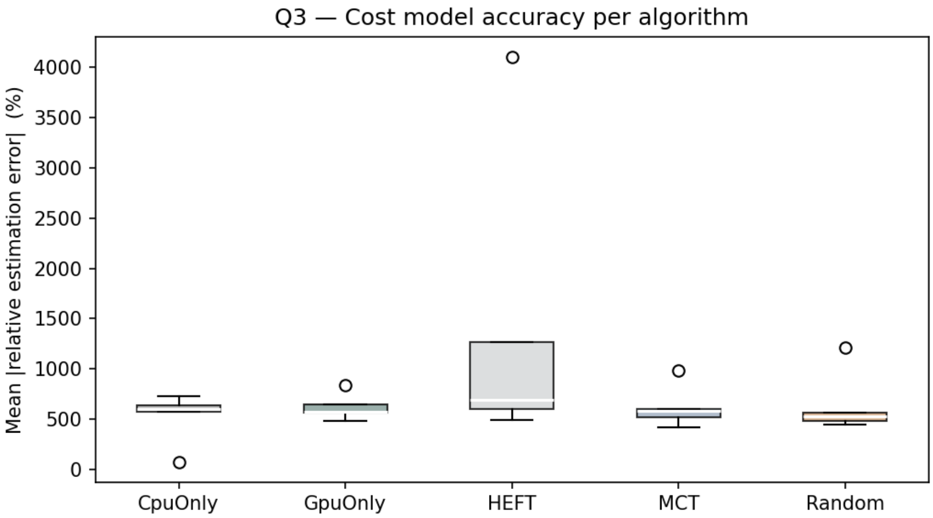

Figure 2: Static Scheduler Cost Estimation Error

5.2 Dynamic Scheduling With Online Calibration

Our online calibration method has three components:

- It builds a cross-case workload prior from task signatures (‘TaskType‘, FLOPs, input bytes, and output bytes), so repeated workload families do not restart from a cold analytical estimate.
- It refines this prior online within each case using per-task, per-device runtime measurements, blending global prior knowledge with case-local corrections after each completed task.
- It uses the calibrated estimate during dynamic CPU/GPU scheduling, while still exploring unseen device assignments when local data is unavailable and falling back to the analytical model when no runtime history exists.

Figure 3 reports the best makespan reduction achieved by online calibration for each case, ordered from left to right by decreasing improvement. The gains are clearly workload-dependent. The best case, `cholesky _nt4 _b64 _seed42`, shows an almost 90% reduction, while most other cases fall in a more moderate range of roughly 5–30%, and a small number of cases show little or no benefit. The largest reduction appears to come from cold-start correction: the first run has no case-local history, so online calibration can recover from initially poor estimates and produce an unusually large makespan improvement in this case.

This pattern is consistent with Figure 4: better runtime estimation generally helps dynamic scheduling, but the end-to-end makespan improvement depends on whether the corrected estimates actually change critical-path decisions, CPU/GPU assignment, or transfer-sensitive placements. In other words, online calibration can substantially improve performance when early measurements correct important scheduling mistakes, but estimation accuracy alone does not guarantee proportional speedup in every workload.

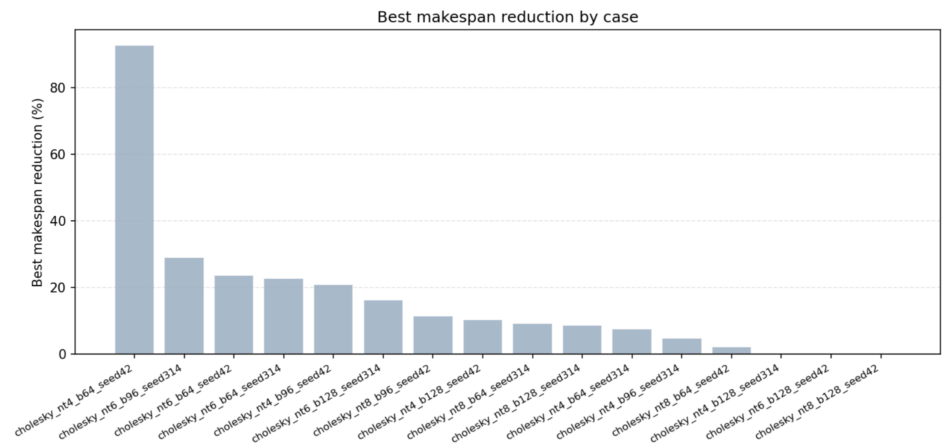

Figure 3: Best makespan reduction by case in order of execution

Figure 4: Run-5 estimation-error reduction relative to Run 1 for each case.

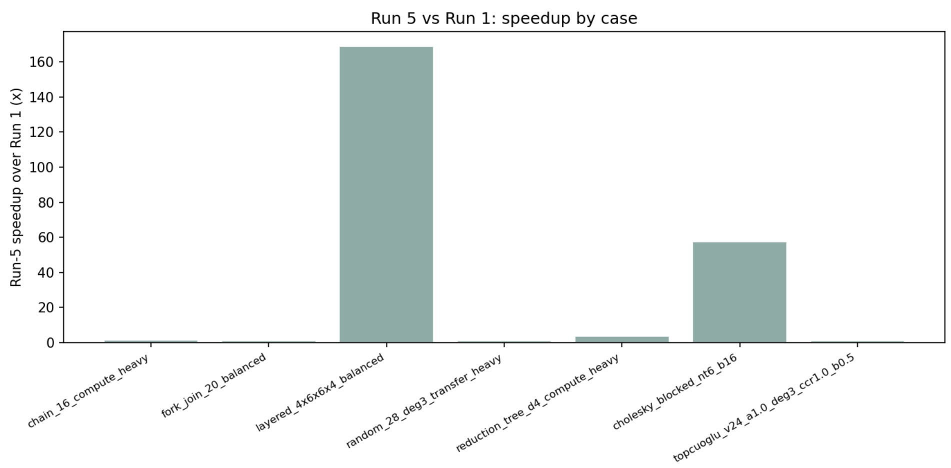

Figure 5: Run-5 speedup relative to Run 1 for each case.

6 Challenges and Concerns

6.1 GPU-only Execution May Outperform Hybrid Scheduling

While our heterogeneous scheduler aims to exploit both CPU and GPU resources simultaneously, assigning all tasks exclusively to the GPU may in practice yield lower makespan. When the GPU has sufficient parallelism to keep all cores occupied, the overhead of routing some tasks to CPU can outweigh any throughput gain from utilizing the CPU. Specifically, costs like scheduling latency, data synchronization, and memory transfers can cancel out the speed gains. The hybrid approach only becomes beneficial when the CPU can absorb tasks that would otherwise stall the GPU pipeline. So the specific test case, the task granularity and scheduling algorithms all impact the final optimization results.

6.2 Static Scheduling May Outperform Online Dynamic Scheduling

Static schedulers have access to the full DAG topology at planning time, allowing them to reason globally about critical-path lengths, load balance, and communication bottlenecks before any task executes. Online dynamic schedulers make greedy decisions with only local visibility, and may commit to suboptimal assignments that cannot be revised once downstream dependencies are revealed. Unless the runtime cost model is significantly more accurate than the static estimate, or the workload exhibits high task-time variance, static algorithms tend to produce tighter schedules. Therefore, our implementation strategy is critical in determining whether dynamic scheduling truly works.

6.3 Cost Model Accuracy Remains a Concern

The effectiveness of any DAG scheduler is tightly coupled to the fidelity of its cost model. In practice, GPU kernel execution times vary with occupancy, cache state, and memory-access patterns in ways that are difficult to predict analytically. Similarly, PCIe transfer latency is sensitive to contention and transfer size in a non-linear fashion. A poorly calibrated cost model can cause the scheduler to assign long tasks to bottleneck devices. In that case, we need to invest in building more robust cost models to improve execution time predictions.

7 Updated Schedule

Time Period	**Task**	**Owner**
Week 1 (First Half)	Refine the feature design of the learned cost model to reduce estimation error, and integrate the learned model into the static scheduling algorithms.	Peiran Hou
Week 1 (Second Half)	Improve the online calibration strategy and integrate it into the static scheduler.	Jingxuan Zhang
Week 2 (First Half)	Explore CUDA-side optimizations, such as using CUDA Graphs, to reduce kernel launch overhead and improve execution efficiency.	Jingxuan Zhang, Peiran Hou
Week 2 (Second Half)	Run benchmark experiments, tune system parameters, and compare static analytical, static learned, and online-calibrated scheduling models.	Jingxuan Zhang, Peiran Hou
Week 3 (First Half)	Conduct comprehensive testing, finalize evaluation results, and begin drafting the final report.	Jingxuan Zhang, Peiran Hou
Week 3 (Second Half)	Complete the final report and prepare the poster and presentation materials.	Jingxuan Zhang, Peiran Hou