High-Performance Task Scheduling for Heterogeneous Systems

Jingxuan Zhang, Peiran Hou

March 2026

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

1 Summary

We propose to develop an intelligent heuristic, list-based 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 Background

DAGs are the standard abstraction for managing task dependencies in high-performance computing. They are essential in Data Analytics (e.g., Spark), Model Training Pipeline, and Scientific Computing for managing large-scale simulation workflows. By modeling execution as a DAG, systems can identify independent tasks that can be executed in parallel while respecting synchronization constraints.

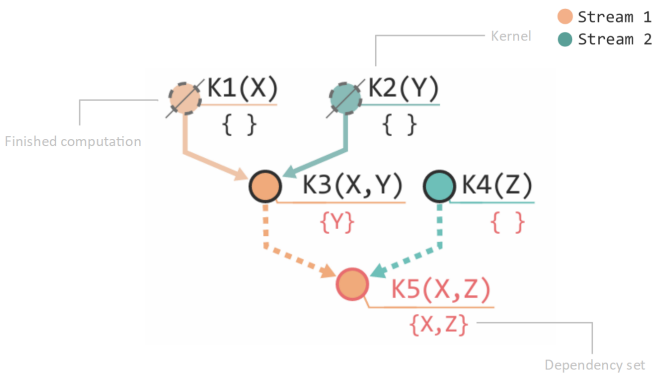

Figure 1: Computation DAG

Some tasks are "control-heavy" (better suited for the CPU), while others are "data-parallel" (ideal for the GPU), which introduce a "communication vs. computation" trade-off. A good task scheduler is required to decide when the acceleration on a GPU outweighs the transfer latency of moving data from CPU memory. There are two aspects in which the problem can benefit from parallelism. One is task parallelism, which indicates that independent nodes in the DAG can be executed simultaneously. Another is data parallelism. Within an individual node, the GPU can process massive arrays in parallel.

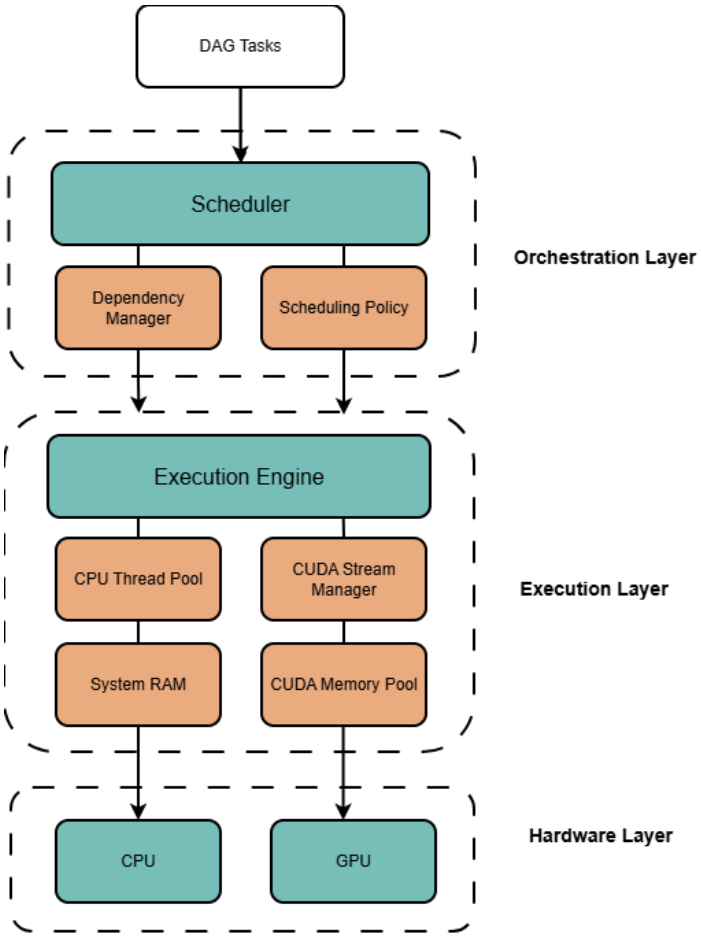

Figure 2: System Architecture

3 The Challenge

This problem is challenging because efficiently mapping a DAG-based workload onto heterogeneous CPU–GPU systems requires balancing computation, data movement, and resource constraints.

3.1 Workload

- **Computation Granularity:** When a DAG consists of hundreds of fast-executing nodes, the time required for the CPU to prepare a command and trigger the GPU via the driver can exceed the task's actual execution time. This launch overhead makes it difficult to parallelize fine-grained tasks.
- **Memory Access and Locality:** Tasks might use entirely different data sets, forcing the VRAM to constantly swap data in and out.
- **Dependencies:** Some workloads have long sequences of serial tasks that limit the theoretical speedup regardless of how many GPUs are available.

3.2 Constraints

The constraints below make it difficult to map workloads efficiently, as task placement decisions must consider both memory availability and communication bottlenecks in addition to computation.

- **Limited VRAM Capacity:** Since the CPU and GPU do not share a physical memory space, the program is constrained by VRAM capacity. If the MemoryPool runs out of space, the entire pipeline stalls.
- **Shared PCIe Bandwidth:** The PCIe bus is shared between multiple tasks. If two CUDA streams try to copy data at the same time, they compete for bandwidth, slowing down the entire system.

3.3 Cost Model

Predicting the best device for a task is difficult. The total execution cost depends not only on computation time but also on data transfer overhead between CPU and GPU. Additionally, resource contention, such as competition for VRAM bandwidth, PCIe bandwidth, and compute units, introduces variability that is difficult to model statically.

3.4 Scheduling Algorithm

Making the scheduling algorithm efficient is challenging. A standard scheduling algorithm like HEFT (Heterogeneous Earliest Finish Time) often struggles because it assumes hardware is static. In reality, the GPU might be busy but actually have idle SMs that could take more work.

4 Resources

We will develop our system using a heterogeneous computing environment consisting of multi-core CPUs and NVIDIA GPUs. Development and testing will primarily be conducted on CMU GHC machines.

We plan to implement the scheduler largely from scratch to maintain flexibility in designing and evaluating different scheduling strategies. However, we will build on several existing components and frameworks to accelerate development. For GPU execution, we will use

CUDA (including CUDA Streams and potentially CUDA Graphs) as the primary programming model. For CPU parallelism, we will use C++ threading libraries. We may also reuse basic DAG representations and task execution structures from existing open-source projects such as the heterogeneous task scheduler repository[6] as a reference, but not as a full codebase.

Our scheduling algorithms will be inspired by prior work in heterogeneous DAG scheduling, particularly the HEFT (Heterogeneous Earliest Finish Time) algorithm[1]. We will use the following references as guidance for both algorithm design and evaluation methodology: [1, 2, 3, 4, 5, 6, 7]

5 Goals and Deliverables

5.1 Plan to Achieve

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.

5.2 Hope to Achieve

- Apply CUDA Graph Integration for Fine-Grained Tasks. The scheduler can automatically detect clusters of small nodes and make them into a CUDA Graph to reduce the CPU-side launch overhead.

- Use learned rather than static cost model to predict the runtime of the workflow execution. Refine the cost model to account for PCIe bus contention when multiple streams are performing data transfers simultaneously.

6 Platform Choice

We use C++ with CUDA on GHC machine to support efficient heterogeneous computing. C++ provides low-level control for implementing the DAG scheduler and CPU parallelism, while CUDA enables GPU acceleration and concurrency through streams. The GHC machines support multi-core CPU and GPU execution.

This platform is well-suited for our workload because DAG tasks exhibit diverse characteristics: compute-intensive tasks benefit from GPU execution, while smaller or latency-sensitive tasks are better handled by CPUs. The heterogeneous system allows us to balance computation and data transfer costs, making it ideal for studying scheduling strategies that minimize overall execution time.

7 Schedule

Week	**Planned Tasks / Goals**	**Notes / Dependencies**
Mar 23 – Mar 29	Submit proposal and build initial framework	Implement dependency manager, ready queue, and basic load balancing
Mar 30 – Apr 5	CPU task parallelization	Implement thread pool and task execution on CPU
Apr 6 – Apr 12	GPU task optimization	Develop memory pool and optimize CUDA execution
Apr 13 – Apr 19	Cost model design and integration	Integrate scheduling policy with runtime cost estimation
Apr 20 – Apr 26	Comprehensive testing and optimization	Benchmarking, profiling, and performance tuning
Apr 27 – Apr 30	Finalize report and poster	Prepare final deliverables

References

[1] H. Topcuoglu, S. Hariri and Min-You Wu, *Task scheduling algorithms for heterogeneous processors*, Proceedings. Eighth Heterogeneous Computing Workshop (HCW'99), San Juan, PR, USA, 1999. Available: `https://doi.org/10.1109/71.993206`

[2] H. Arabnejad and J. G. Barbosa, *List Scheduling Algorithm for Heterogeneous Systems by an Optimistic Cost Table*, IEEE Transactions on Parallel and Distributed Systems, vol. 25, no. 3, 2014. Avaliable: `https://doi.org/10.1109/TPDS.2013.57`

[3] A. Parravicini, A. Delamare, M. Arnaboldi, and M. D. Santambrogio, *DAG-based Scheduling with Resource Sharing for Multi-task Applications in a Polyglot GPU Runtime,* 2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS), 2021, pp. 111-120. Avaliable: https://doi.ieeecomputersociety.org/10.1109/IPDPS49936.2021.00020

[4] Sirisha, D., *Complexity versus quality: a trade-off for scheduling workflows in heterogeneous computing environments.*, 2023. Available: https://doi.org/10.1007/s11227-022-04687-x

[5] Ding Pan, Zhuangzhuang Zhou, Long Qian, Binhang Yuan, *Trident: Adaptive Scheduling for Heterogeneous Multimodal Data Pipelines*, 2026. Available: https://doi.org/10.48550/arXiv.2603.02075

[6] Lessup, *CPU/GPU Heterogeneous Task Scheduling Framework (CUDA + C++17): DAG Dependencies, Strategies & Profiling.* Avaliable: https://github.com/LessUp/heterogeneous-task-scheduler

[7] *Pegasus (Makes the Work Flow).* Avaliable: https://pegasus.isi.edu/