Impact of GPU Multistreaming on Task-Based DAG Execution in Dagger.jl

Víctor Câmara*
University of Brasilia
Brasilia, Federal District, Brazil
victor.camara@aluno.unb.br

Felipe Tomé (advisor)
Massachusetts Institute of Technology
Cambridge, MA, USA
felipe0@mit.edu

Julian Samaroo (advisor)
Massachusetts Institute of Technology
Cambridge, MA, USA
jpsamaroo@mit.edu

Abstract

Task-graph runtimes like Dagger.jl typically drive each GPU through a single execution stream, serializing independent tasks even when the dependency graph permits overlap. We extend Dagger.jl with N concurrent CUDA/HIP streams per device and evaluate four stream-assignment policies (round-robin, random, shortest device queue (SDQ), and locality-aware) across four DAG topologies (linear, diamond, chainlink, tangled) and four stream counts (1, 2, 8, 32) on an NVIDIA H200 (CUDA) and an AMD MI300A (ROCm). At 32 streams, round-robin cuts median wall-clock time by 4.5× on average on H200 and 2.5× on MI300A, and is the fastest policy in 15 of 16 configurations with 8 or more streams. SDQ is a close second on H200 (4.0×) but only matches random on MI300A (1.75× vs. 1.73×). At 32 streams, locality-aware assignment issues 1.6–8× fewer synchronization events than round-robin but stops scaling beyond 8 streams, reaching only 2.5× on H200 and 1.5× on MI300A.

CCS Concepts

• **Computing methodologies** → **Massively parallel algorithms**; **Concurrent algorithms**; • **Theory of computation** → *Parallel computing models.*

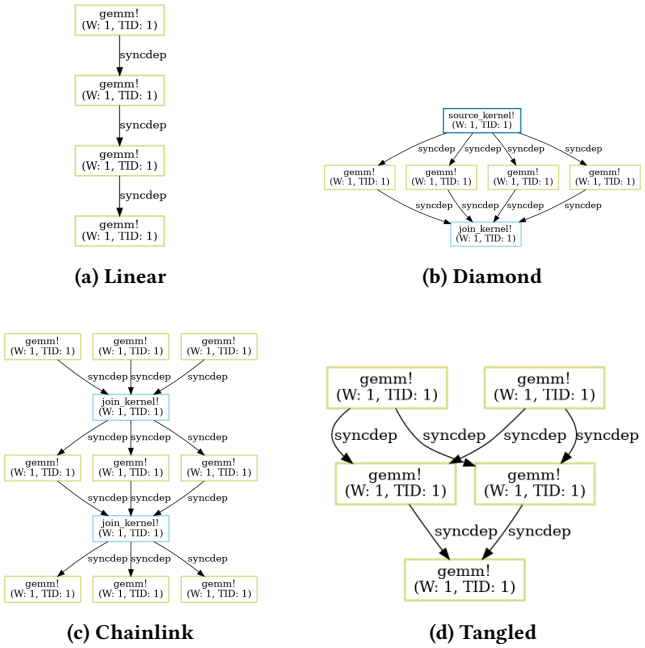
Keywords

HPC, Julia, Dagger, CUDA, ROCm, Streams

ACM Reference Format:
Víctor Câmara, Felipe Tomé (advisor), and Julian Samaroo (advisor). 2026. Impact of GPU Multistreaming on Task-Based DAG Execution in Dagger.jl. In *Proceedings of The International Conference for High Performance Computing, Networking, Storage, and Analysis. (SC '26)*. ACM, New York, NY, USA, 2 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

High-performance computing (HPC) and scientific research depend on task-based runtime systems such as PaRSEC [2], IRIS [3], and Dagger.jl [1, 4] to fully exploit modern heterogeneous hardware. Dagger.jl, a Julia framework for parallel and distributed computing, now supports multi-stream execution on GPUs, allowing independent tasks to run concurrently on separate hardware queues instead of serializing through a single stream. This raises GPU utilization but opens a scheduling question: which stream should each task use? Blind policies maximize overlap, dependency-aware ones avoid cross-stream synchronization, and the right balance may shift with DAG shape, stream count, and hardware. We implement and evaluate four GPU stream-scheduling strategies: round-robin, random assignment, shortest device queue (SDQ), and locality-aware assignment, adapting the DAG-topology taxonomy and policies of a cross-device scheduler for heterogeneous runtimes [3] to a single GPU's concurrent streams. Strategies are benchmarked across four DAG topologies (linear chain, diamond, chainlink and tangled) and a range of stream counts (1, 2, 8, 32), on both AMD and NVIDIA GPUs to assess performance across different hardware backends.



(a) Linear (b) Diamond

(c) Chainlink (d) Tangled

Figure 1: DAG Topologies

2 Methodology

We evaluate multi-stream GPU scheduling in Dagger.jl using a unified benchmark harness that runs identically on CUDA and ROCm backends. Each configuration executes a synthetic task DAG built from Dagger's `spawn_datadeps` region, where each DAG node is a 2×2-tiled 4096^2 GEMM (8 tile tasks), giving 24 to 96 tasks per

*Main author contributed to this research.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SC '26, Chicago, IL
© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-XXXX-X/2026/11
https://doi.org/XXXXXXX.XXXXXXX

DAG. We vary the number of per-device streams (1, 2, 8, 32) and the stream-selection strategy. Round-robin cycles through streams; random picks one uniformly; SDQ picks the stream with the fewest in-flight tasks; locality-aware picks the stream that produced most of a task's inputs, falling back to SDQ when that stream is more than two tasks deeper than the shortest queue. Each tile task runs a calibrated occupancy-controlled "spin" kernel, sized to 8 ms and 1/32 of the GPU's SMs/CUs, so that the benchmark isolates cross-task stream overlap from any saturation effects of a particular numerical library. Only the datadeps region is timed (allocation and `collect` excluded); each configuration runs 5 warmup and 20 measured samples, device-synchronized between samples, reporting median/min/IQR wall-clock time, a busy fraction (summed standalone kernel time over median wall time; values above 1 indicate overlap), and cross-stream synchronization-event counts.

3 Results

Figures 2 and 3 show median wall-clock time across DAG topologies, strategies, and stream counts on H200 and MI300A.

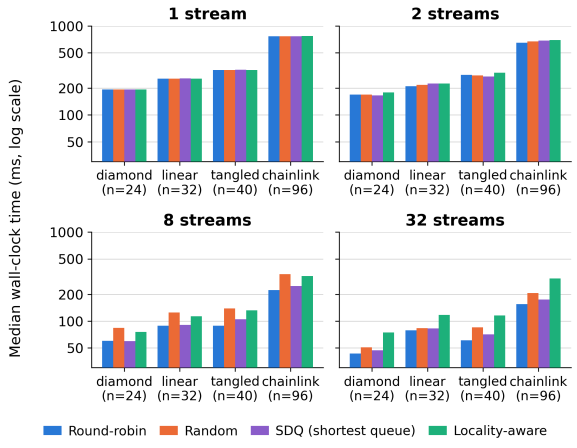


Figure 2: NVIDIA H200 (CUDA) Results

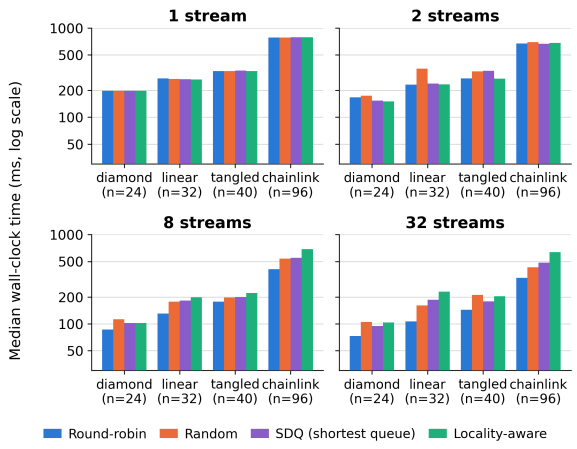


Figure 3: AMD MI300A (ROCm) Results

At one stream, all four policies are within 3% of each other, confirming that differences come from stream parallelism, not assignment overhead. On H200, round-robin, SDQ, and random scale steadily with stream count, reaching 4.5×, 4.0×, and 3.6× average speedup at 32 streams, with round-robin reaching a busy fraction of up to 5.2. Locality-aware assignment reaches 2.4× at 8 streams and then plateaus (2.5× at 32), likely because following producer streams concentrates dependent tasks on few streams, but it issues 1.6–2.2× fewer synchronization events than round-robin. On MI300A all policies scale less and round-robin's lead over SDQ and random grows: it reaches 2.5×, versus 1.75× for SDQ, 1.7× for random, and 1.5× for locality, whose event savings widen to 2.3–8×. Topology shapes the magnitude, not the ranking: on H200, tangled (5.3×) and chainlink (5.0×) gain most under round-robin and linear least (3.3×), while on MI300A all four fall within 2.3–2.7×.

4 Conclusion

We extended Dagger.jl with N concurrent GPU streams per device. Round-robin is the most robust policy, delivering the largest wall-clock gains (4.5× on H200, 2.5× on MI300A at 32 streams), with SDQ close behind on H200. At 32 streams, policy choice alone changes wall-clock time by up to 1.9× on H200 and 2.2× on MI300A. Locality-aware assignment gives up about 40% of round-robin's speedup in exchange for 1.6–8× fewer synchronization events, which may pay off where events are costlier, such as multi-GPU or multi-node DAGs. Future work includes tracking device-side queue depth for SDQ, profiling-based policies, and benchmarking Dagger.jl on numerical kernels such as Cholesky factorization.

References

[1] Rabab Alomairy, Felipe Tome, Julian Samaroo, and Alan Edelman. 2024. Dynamic task scheduling with data dependency awareness using julia. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)*. IEEE.

[2] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Mathieu Faverge, Thomas Herault, and Jack Dongarra. 2013. Parsec: a programming paradigm exploiting heterogeneity for enhancing scalability. *Computing in Science & Engineering*.

[3] Beau Johnston, Narasinga Rao Miniskar, Aaron Young, Mohammad Alaul Haque Monil, Seyong Lee, and Jeffrey S. Vetter. 2024. Iris: exploring performance scaling of the intelligent runtime system and its dynamic scheduling policies. In *2024 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*. IEEE, 58–67.

[4] Julian Samaroo, Rabab Alomairy, Mose Giordano, and Alan Edelman. 2024. Efficient dynamic task scheduling in heterogeneous environments with julia.