Early Lessons from Characterizing Unified CPU-GPU Memory on DGX Spark

Seoyeong Lee and Xiaoyi Lu

Department of Electrical & Computer Engineering, {seoyeonglee, xiaoyilu}@ufl.edu
University of Florida

PADSYS Lab

Introduction and Motivation

- NVIDIA DGX Spark represents a new workstation-scale design point in which an Arm CPU and a Blackwell GPU share one coherent LPDDR5X-backed physical memory pool[1].

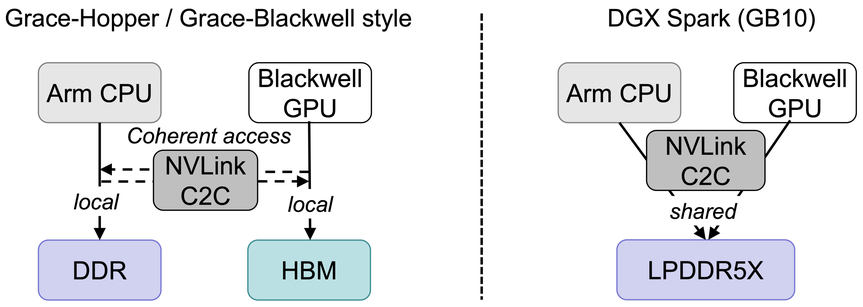


Fig. 1: Architectural comparison of separate and shared physical memory pools in coherent CPU–GPU systems[1][2]. Grace Hopper/Grace Blackwell retain separate CPU and GPU memory pools with coherent NVLink-C2C access (left); DGX Spark (GB10) shares one LPDDR5X pool between them (right).

- This organization differs from traditional discrete CPU-GPU systems, where host and device memories are physically separate and data movement is more explicit.
- We use roofline as a starting point to capture the steady-state CPU/GPU throughput gap on DGX Spark.

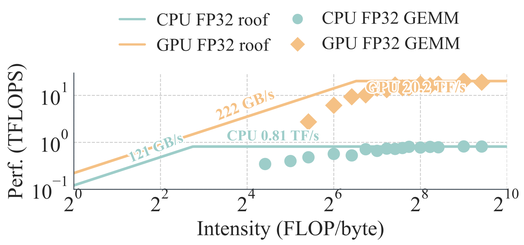


Fig. 2: Roofline-style FP32 view of DGX Spark. Empirical CPU and GPU bandwidth and compute roofs are overlaid with measured GEMM points.

Table. 1: Pinned CUDA H2D/D2H copy bandwidth measured using 1 GiB cudaMemcpyAsync transfers on DGX Spark.

Measured CUDA copy API bandwidth	
H2D	20.7 GB/s
D2H	20.6 GB/s

- However, roofline cannot separate the costs involved in a CPU–GPU memory-policy run.
- Because CPU and GPU share one physical memory pool, these costs can be easy to conflate in a single runtime or latency number.

Research Question

What costs can remain hidden when CPU and GPU share a unified memory pool?

- We examine three places where memory-policy costs can appear: ❶ accessing a shared buffer, ❷ setting it up and releasing it, and ❸ timing a complete workload.

Ownership Scope | Lifecycle Scope | Timing window

References

[1] NVIDIA, "Hardware Overview," *DGX Spark User Guide*, NVIDIA Documentation, 2026. https://docs.nvidia.com/dgx/dgx-spark/hardware.html
[2] N. Fujita, T. Boku, T. Yoshida, T. Shirai, and M. Tsuji, "GPU-CPU Shared Memory Performance Analysis on NVIDIA GH200," in *2025 IEEE International Conference on Cluster Computing Workshops (CLUSTER Workshops)*, pp. 1–2, 2025.

Acknowledgements

This work was supported in part by the U.S. Department of Energy under Award No. DE-SC0024207 and by the National Science Foundation under Grants 2623546, 2623548, and 2623610. The experiments also used equipment supported by the Farms Food Future (F3) Initiative under the U.S. Department of Commerce Economic Development Administration's Build Back Better Regional Challenge (Award No. 07-79-07913).

Key Performance Factors and Early Findings

Factor 1: Shared Memory Does Not Mean Ownership-Free Access

- We compare cold first-touch and steady-state same-buffer CPU-GPU handoff to measure the ownership-acquisition premium that remains despite a shared physical memory pool.

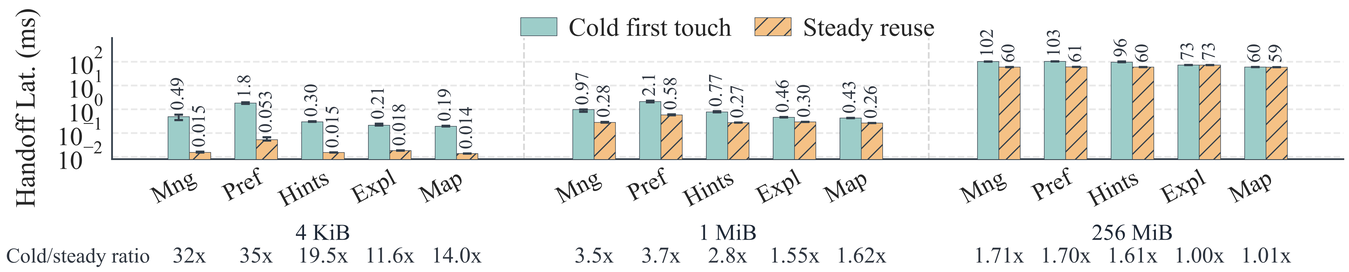


Fig. 3: Cold first-touch and steady-state same-buffer handoff latency by chunk size and memory policy.

- Finding 1: The minimum-latency policy changes between regimes: Mapped is the lowest-latency cold path across the shown sizes, whereas Managed/Hints are lowest in steady state at 4 KiB and 1 MiB.
- Finding 2: The dominant high-cost case shifts with size: Prefetch reaches the largest cold/steady premium at 4 KiB, as high as 35x, while managed-family paths retain a 1.6–1.7x premium at 256 MiB.

Factor 2: Lifecycle Boundary Changes Cost Attribution

- We separately time setup and teardown across buffer sizes to test whether end-to-end policy cost reflects recurring handoff or one-time lifecycle work.

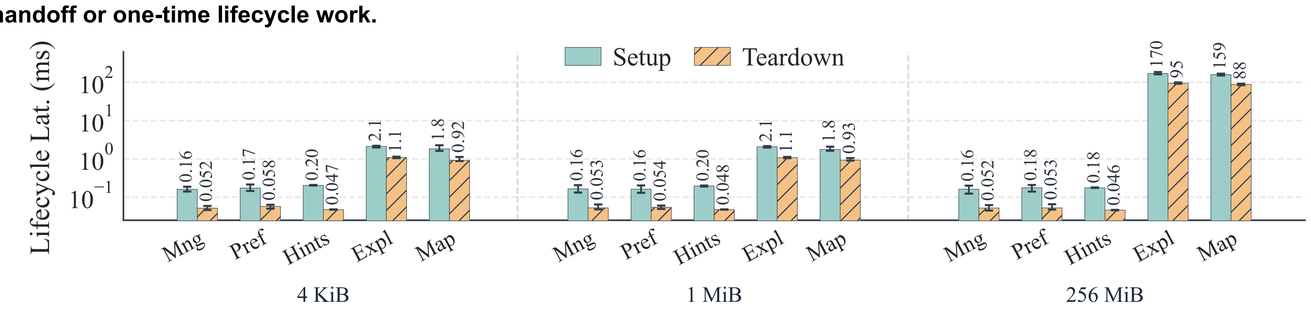


Fig. 4: Setup and teardown latency by buffer size and memory policy. Paired bars show median setup and teardown time.

- Finding 3: At 256 MiB, explicit and mapped paths spend about 264 ms and 246 ms in setup+teardown, while managed-family paths remain near 0.22–0.24 ms.
- Finding 4: Large-buffer policy overhead can therefore be dominated by lifecycle work, not by the recurring ownership-handoff path itself.

Factor 3: Timing Window Changes Policy Ranking

- We run a 400 MiB-per-array vector addition benchmark and compare end-to-end versus steady-state timing windows, then decompose the end-to-end runs to identify which phases drive the ranking shift.

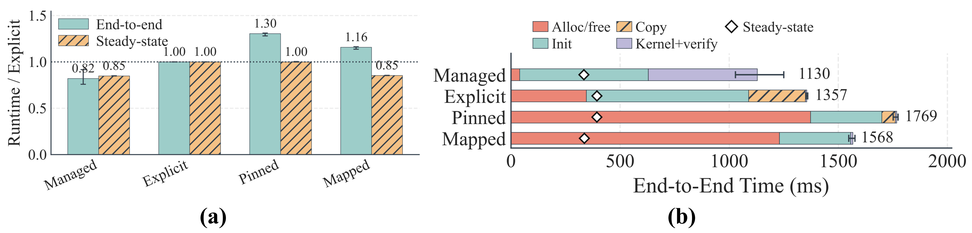


Fig. 5: End-to-end vs steady-state comparison for the 400 MiB-per-array vector addition benchmark. (a) Runtime normalized to explicit copy under end-to-end and steady-state reporting. (b) Absolute end-to-end phase breakdown; diamonds mark the steady-state total

- Finding 5: Policy ranking changes with the timing window: pinned and mapped are slower end-to-end, but pinned becomes near-explicit and mapped becomes near-managed in steady state.
- Finding 6: The shift comes from what the timed window charges: pinned/mapped expose 1.2–1.4 s of alloc/free, while explicit exposes a 260 ms copy term; steady-state totals compress to 334–394 ms.

Conclusion and Future Work

- Unified physical memory does not make CPU-GPU access ownership-free: cold first touch and steady reuse remain distinct.
- Shared-memory programming can hide cost attribution: setup/teardown and timing-window choices can change reported overhead and policy ranking.
- Future work will test these attribution effects under CPU-GPU overlap, placement control, Spark-to-Spark RoCE/NCCL, and full runtime/application workloads.