AMG: AMX–GPU Cooperative Acceleration of Billion-Scale ANNS via GEMM Reformulation

Daehoon Kim

Yonsei University

NVRAMOS 2026

International Workshop on Operating System Support for Next Generation Large Scale NVRAM

To appear at MICRO 2026 (59th IEEE/ACM International Symposium on Microarchitecture)

Retrieval-Augmented Generation (RAG)

The LLM answers from documents retrieved by vector search

- **RAG:** an LLM alone hallucinates and lacks private or recent data, so RAG retrieves relevant documents and grounds the answer on them
- **ANNS:** retrieval must find the closest embeddings among billions of vectors; exact search is too slow, so RAG relies on approximate search
- Even with ANNS, as the vector database grows to **billions** of vectors, retrieval takes **11–85%** of RAG execution time [HeterRAG, ISCA '25]

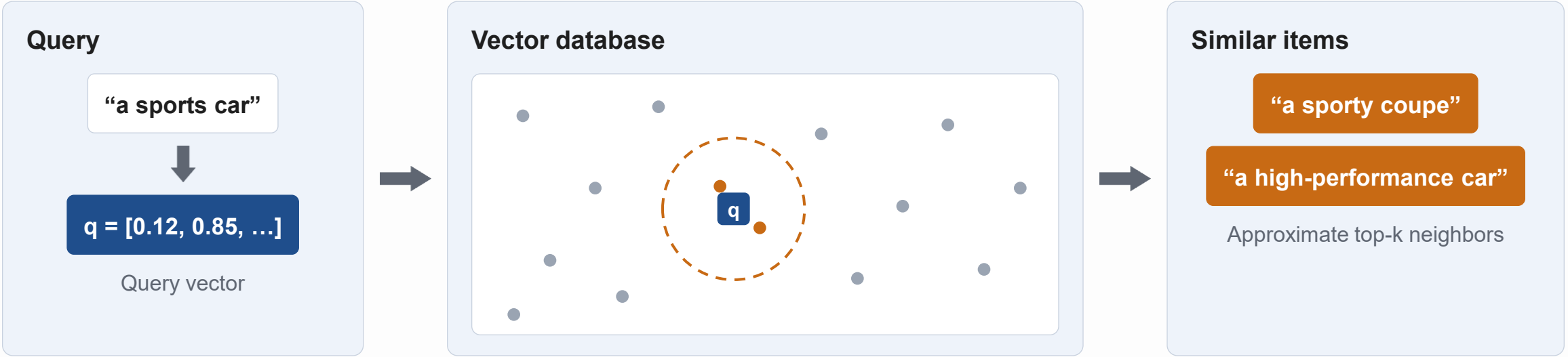


Fast ANNS at billion scale is what keeps RAG responsive and scalable.

Background: ANNS

Approximate Nearest Neighbor Search (ANNS)

- Retrieves **approximate top-k neighbors** of a query vector
- Enables semantic search through vector similarity
- Used across a wide range of applications, including RAG, recommendation, and image search



Nearby vectors represent similar meanings, so ANNS can find similar items.

Background: IVF-based ANNS

IVF (Inverted File index) first selects clusters, then searches their vectors

Coarse search

all queries Q × **centroids C** = **one GEMM**

$$D = Q\,C^{\top} \;:\; (M \times d)\,(d \times K) \rightarrow M \times K$$

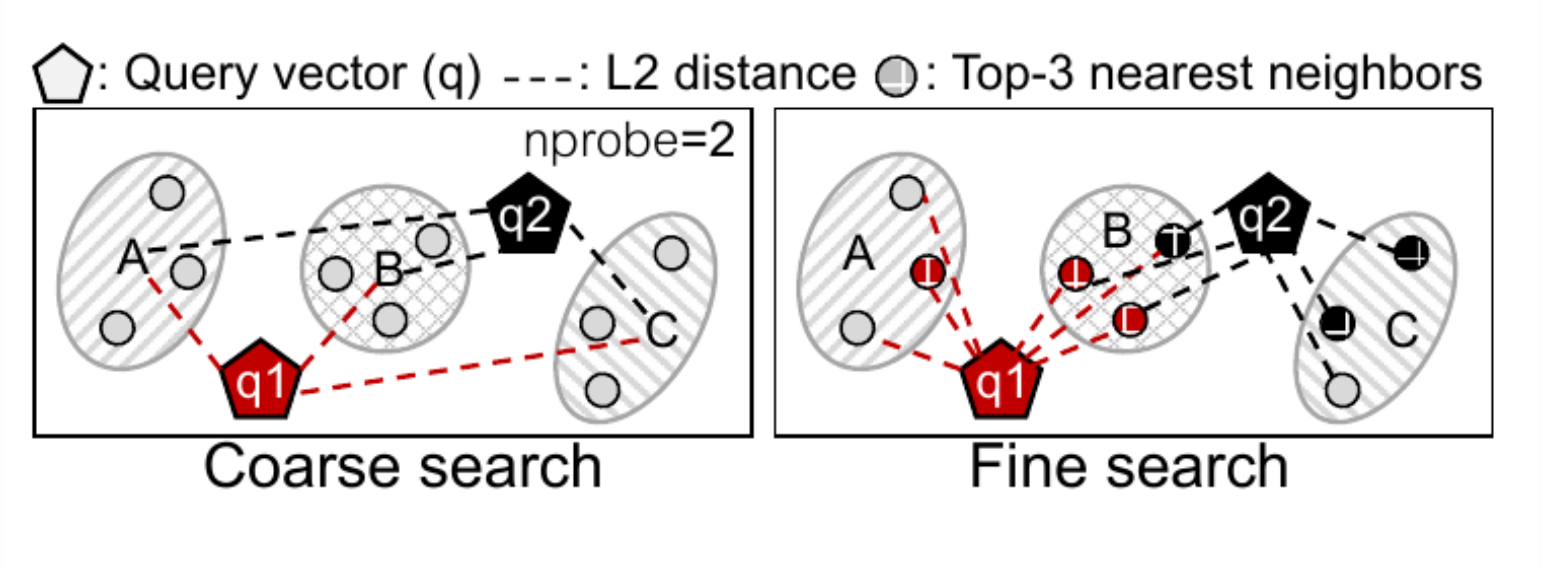
M queries × K centroids → nprobe clusters per query

Fine search

each query q_i × **its cluster Vc** = **many GEMVs**

$$d_i = q_i\,V_c^{\top} \;:\; (1 \times d)\,(d \times N_c) \rightarrow 1 \times N_c$$

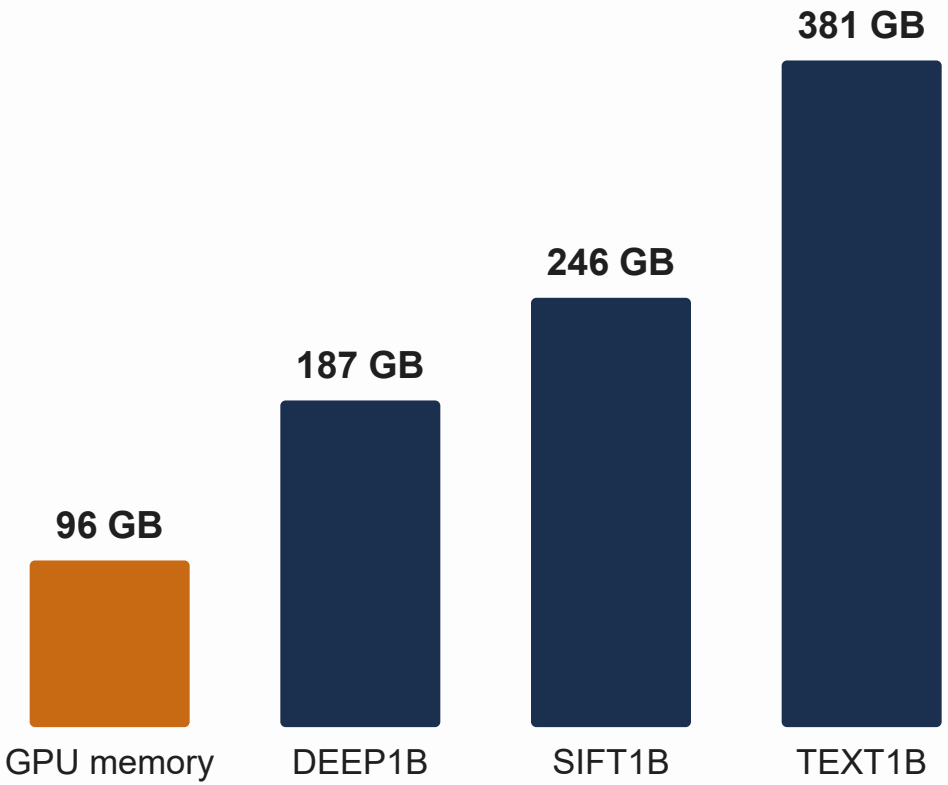
one query × N*c* vectors, repeated per (query, cluster) → top-k



q1 selects clusters A and B, q2 selects B and C (nprobe = 2)

Where should billion-scale ANNS run?

Billion-vector datasets exceed a single GPU's memory



Dataset size in BF16 vs. memory of the evaluated GPU

CPU

- ✓ Holds the dataset in host memory
- ✗ Limited search throughput

Multi-GPU

- ✓ More aggregate GPU memory
- ✗ Extra hardware cost and power

Single GPU
+ host memory

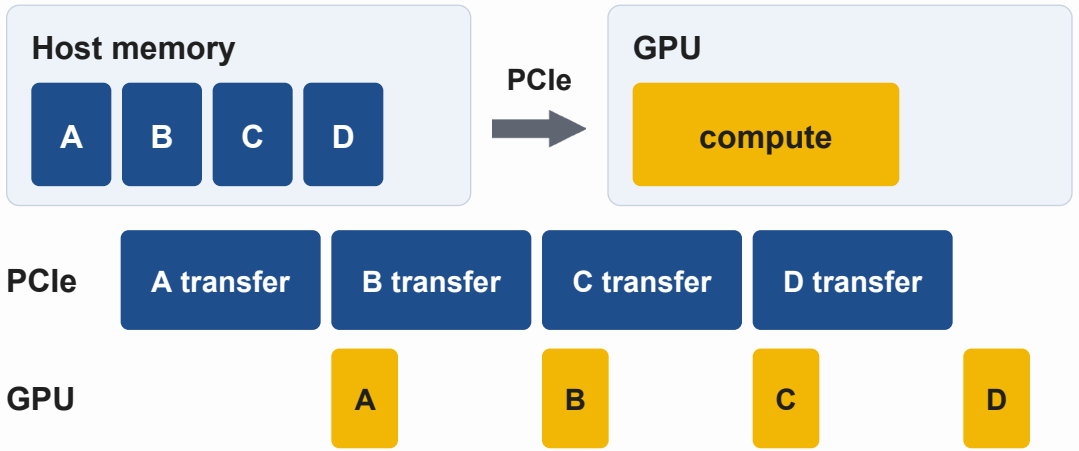
- ✓ High GPU compute throughput
- ✗ Data in host memory must cross PCIe

Our focus: a single GPU backed by large host memory — a practical, cost-efficient setup

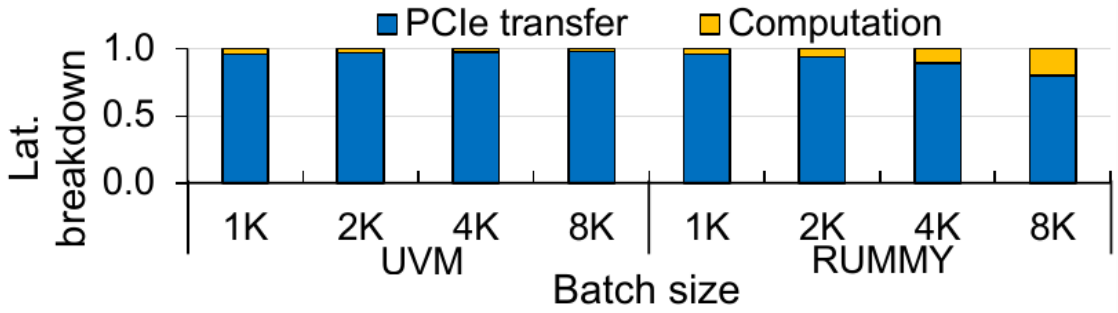
Even with overlap, PCIe is still the bottleneck

Overlap hides transfer time, but not transfer volume

RUMMY [NSDI '24]: data streamed from host memory



Each cluster loaded once; transfer overlaps compute. The GPU mostly waits on PCIe.



Runtime breakdown, SIFT1B, query batches 1024–8192

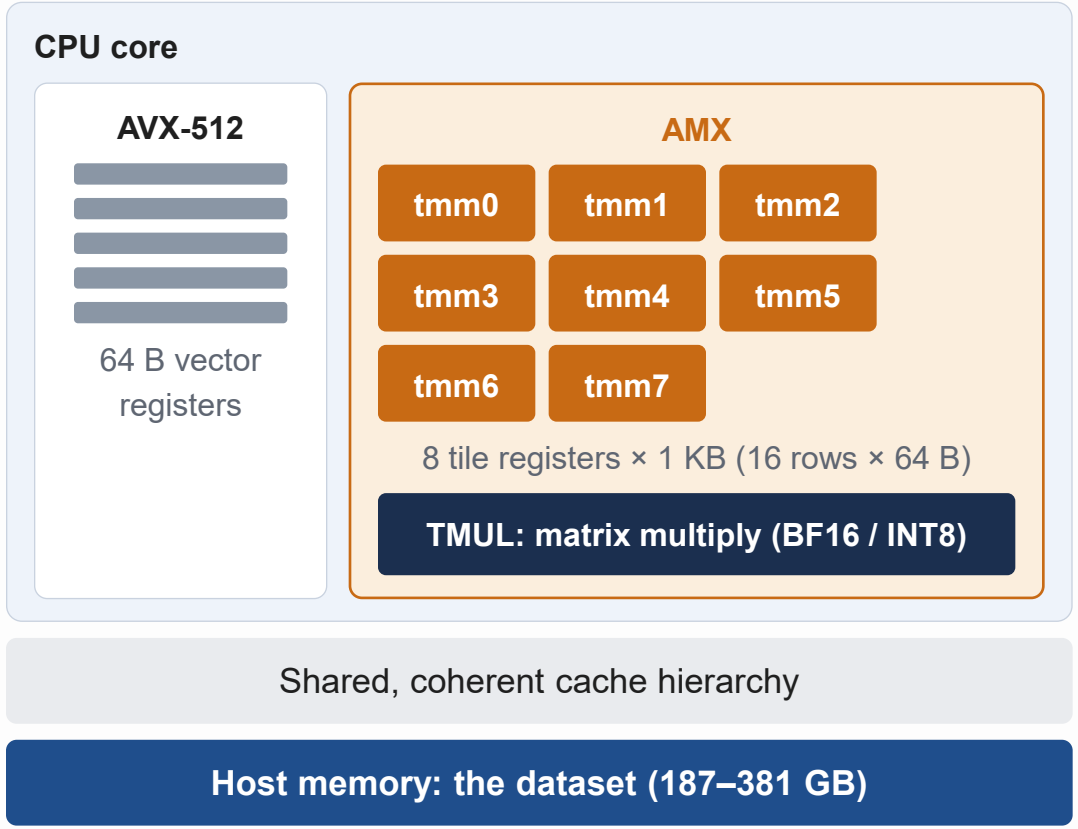
>80% **of RUMMY runtime is PCIe transfer**

96–98% with UVM

Offloading data to host memory is not enough — computation must move to the CPU too.

Intel AMX: an opportunity for CPU offloading

Matrix compute right next to host-resident data



16× **Larger registers than AVX-512**
A 1 KB tile vs. a 64 B vector register keeps matrix blocks on-chip for reuse

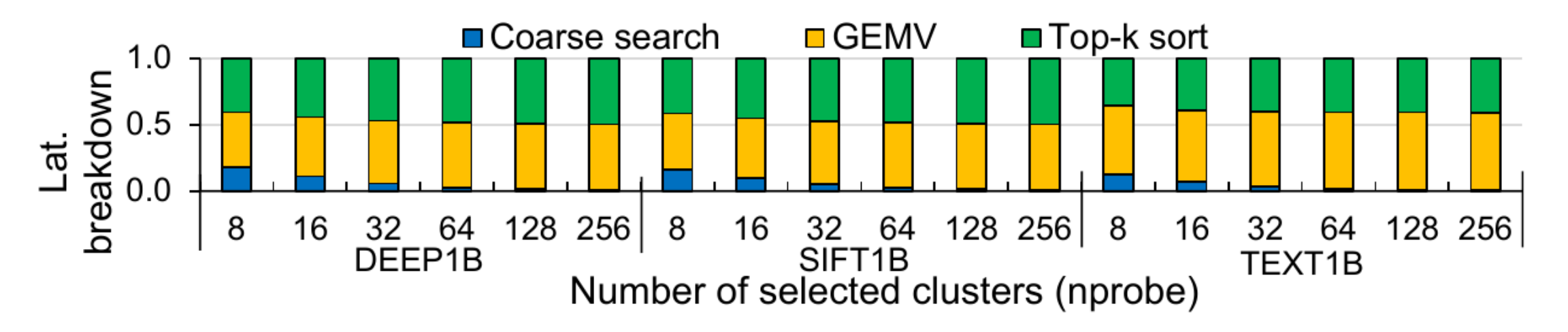
TMUL **Dedicated matrix unit**
Multiply-accumulate on whole tiles in BF16 and INT8

No PCIe **Inside every core**
Computes directly on host-resident data through the cache hierarchy

AMX makes CPU offloading promising, but can ANNS effectively leverage it?

The Challenge: GEMV limits AMX acceleration

GEMV dominates CPU time, yet AMX provides little speedup on it



Runtime breakdown of CPU-only IVF-ANNS using AVX-512 (no AMX), batch 2048

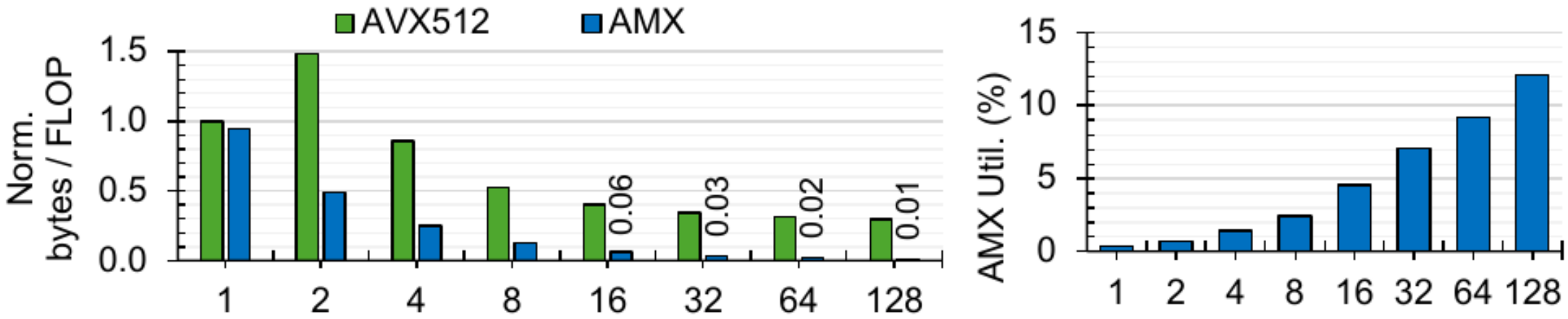
41–59% **of CPU runtime is GEMV**
Top-k adds another 36–49%

1.1× **AMX speedup over AVX-512 (GEMV)**
Too little data reuse across queries

Per-query GEMVs limit AMX acceleration — fine search needs to be reorganized.

AMX efficiency requires GEMM-level data reuse

Only matrix × matrix reuses data in the 1 KB tiles



Bytes per FLOP (normalized to AVX-512 GEMV) and AMX utilization vs. query batch size; 128-d BF16 queries × 32K-vector cluster

3.0–26.8× **fewer bytes per FLOP than AVX-512**
on GEMM; only 1.1× on GEMV

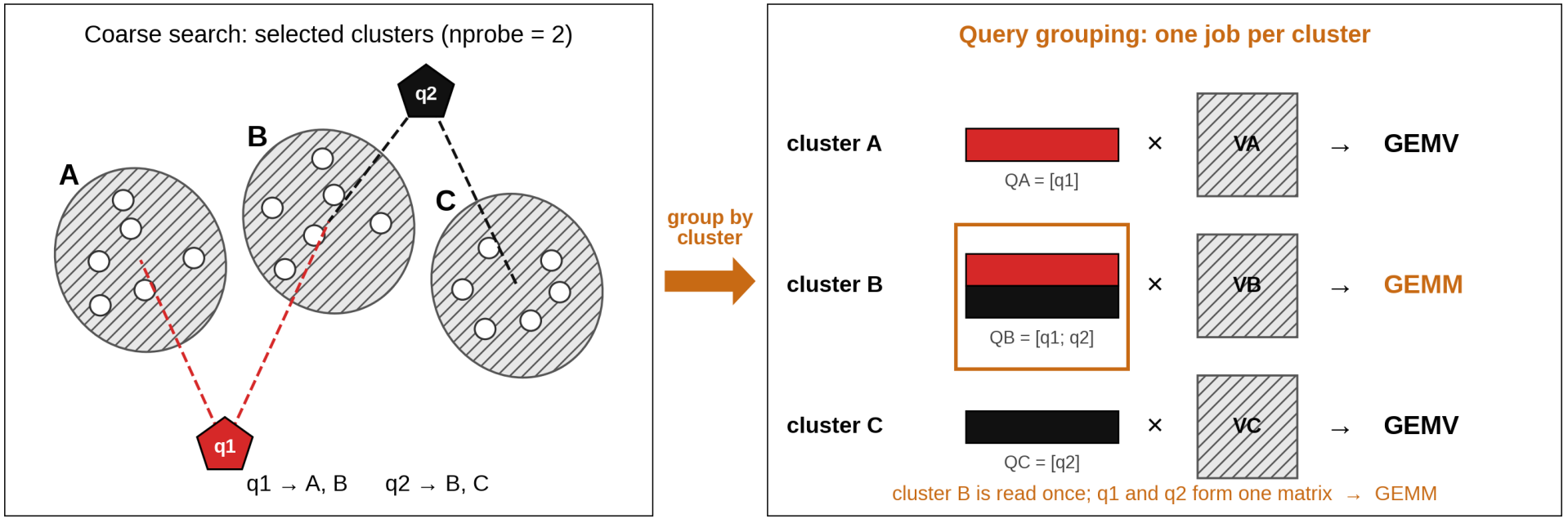
0.4% → 12.1% **AMX utilization, GEMV → GEMM**
batch 1 → 128 queries per cluster

Fine search must be reformulated from GEMV to GEMM to unlock AMX.

Key Idea: Query Grouping

We turn per-query GEMVs into per-cluster GEMMs

- Group the queries in a batch by the clusters they select → the queries of a cluster form one matrix Q*c*
- Same results, only the execution order changes; partial top-k per job, merged per query at the end

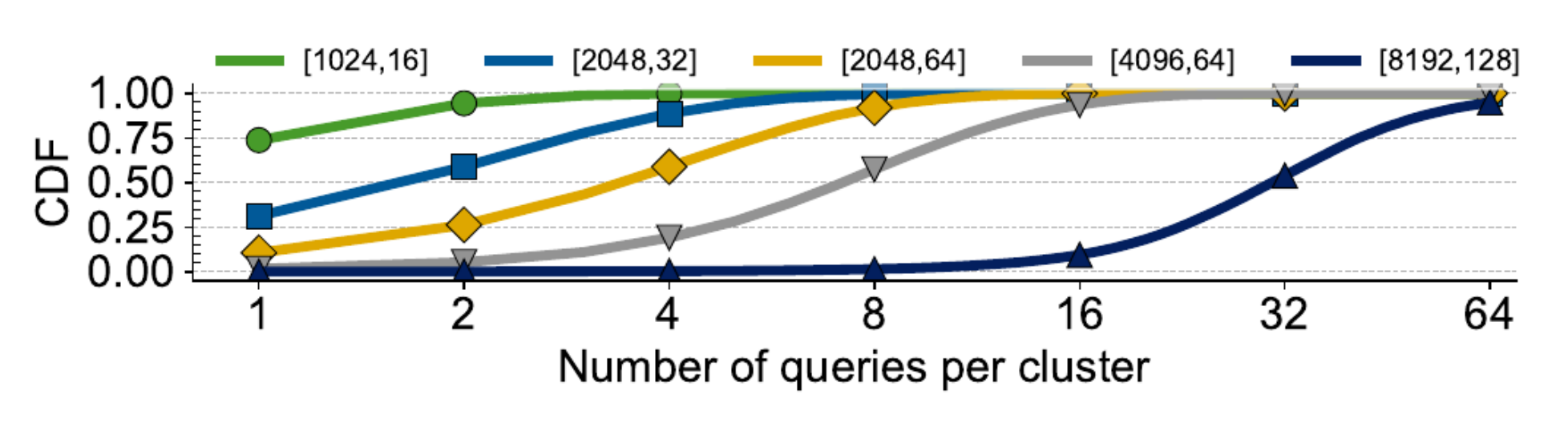


Per-cluster job: $D_c = Q_c V_c^\top \;:\; (M_c \times d)\,(d \times N_c) \to M_c \times N_c$

Query grouping enables efficient AMX acceleration through GEMM reformulation.

Real workloads exhibit high cluster sharing

Large batches and high nprobe convert most GEMVs into sizable GEMMs



CDF of the number of queries selecting the same cluster on DEEP1B, for [batch size, nprobe]

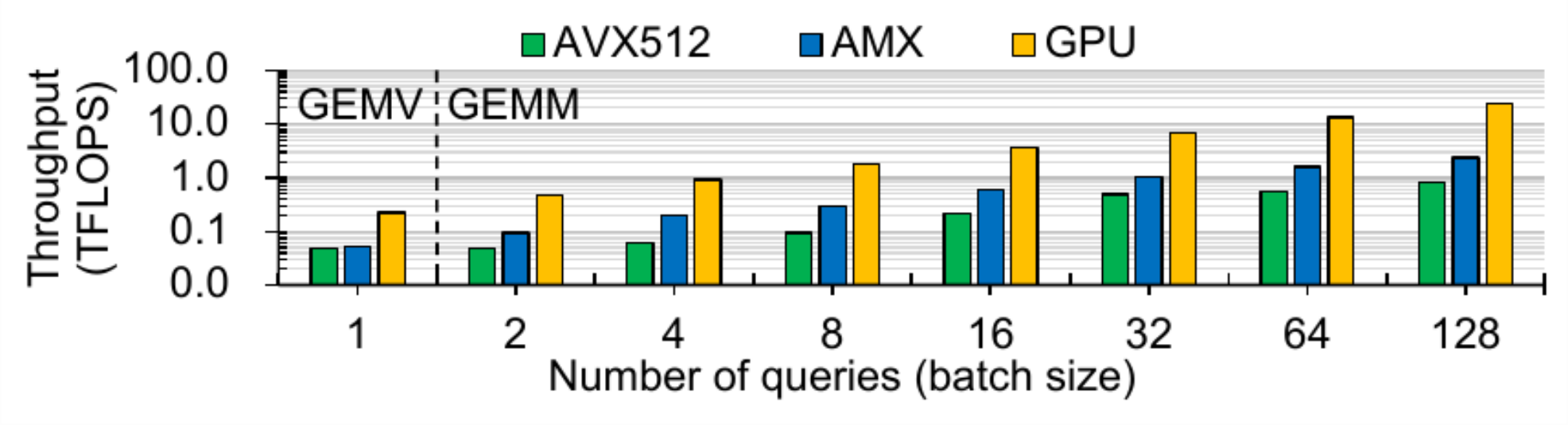
>70% **of clusters touched by a single query**
[batch 1024, nprobe 16] — little to group

>90% **of clusters shared by more than 16 queries**
[batch 8192, nprobe 128] — the serving regime

In the serving regime, most GEMVs become GEMMs, at only 1.0–2.3% grouping overhead.

Query grouping enables effective AMX acceleration

With GEMM, AMX outperforms AVX-512, and the GPU gains as well



Microbenchmark: 128-d BF16 queries × 32K-vector cluster, compute only (no PCIe)

1.9–3.3× **AMX vs. AVX-512 on GEMM**
only 1.1× on GEMV

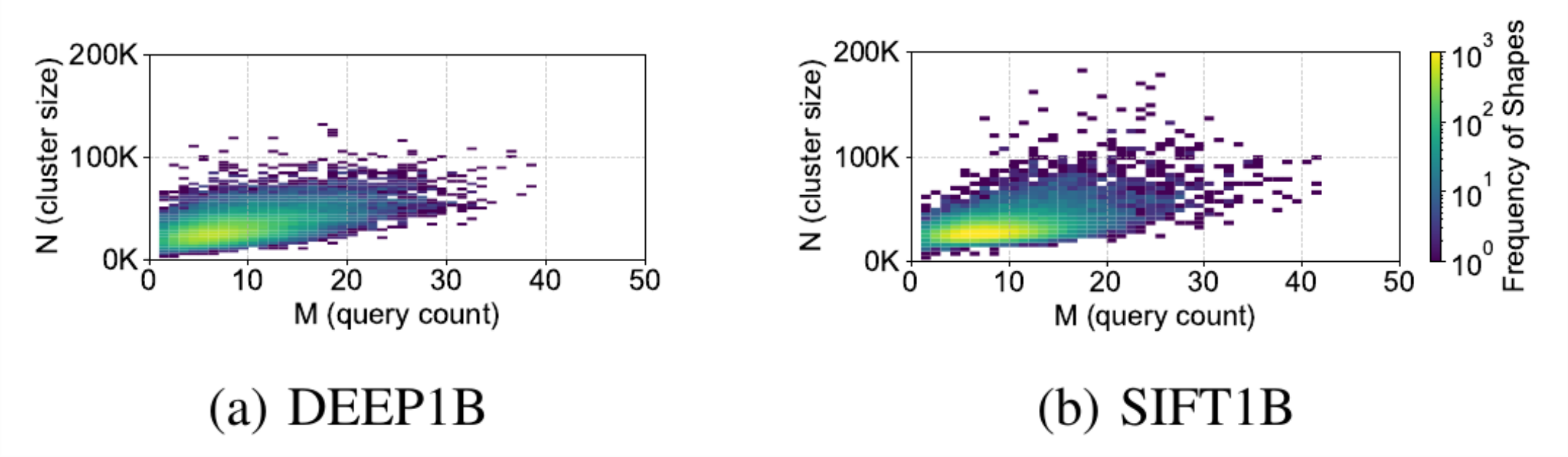
>100× **GPU, GEMV → GEMM**
batch 1 → 128: the GPU gains too

10–22% **AMX vs. GPU on GEMM**
AVX-512 vs. GPU: only 3–11%

With GEMM, AMX becomes effective, and it operates on host-resident data with no PCIe transfer.

Query grouping produces jobs of heterogeneous shapes

Cluster size N and query count M vary independently and are known only at runtime



Distribution of cluster-level job shapes after query grouping (batch 8192, nprobe 32)

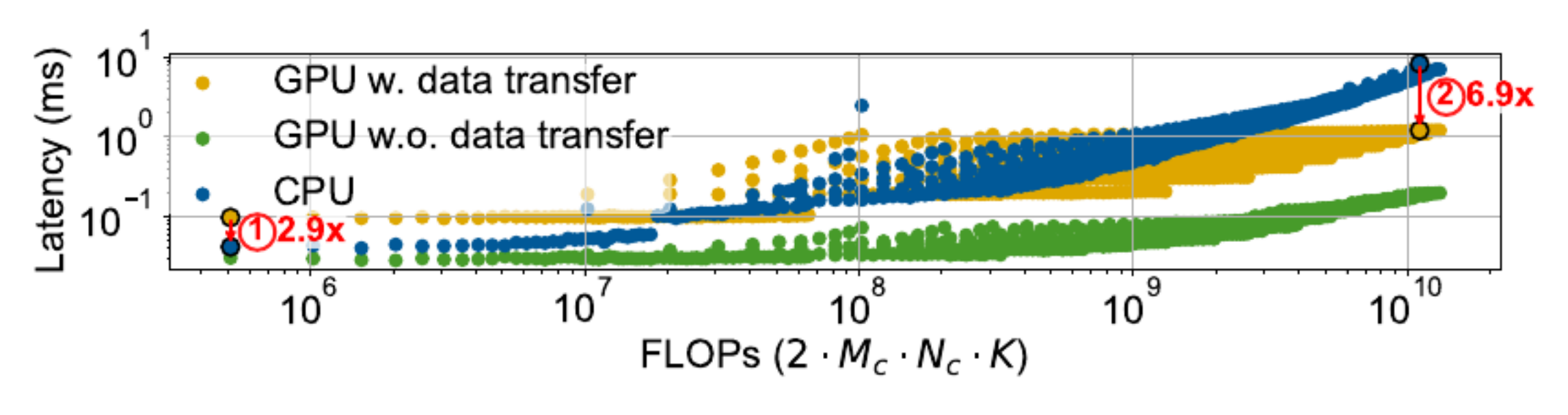
N **cluster size: up to ~200K vectors**
fixed when the index is built

M **queries that selected the cluster**
known only when the batch arrives

A fixed CPU–GPU split cannot fit a stream of jobs like this.

Different GEMM jobs favor different devices

Small jobs favor AMX; large jobs favor the GPU



GEMM execution time vs. job size on the CPU (AMX) and the GPU, with and without host ↔ GPU transfer

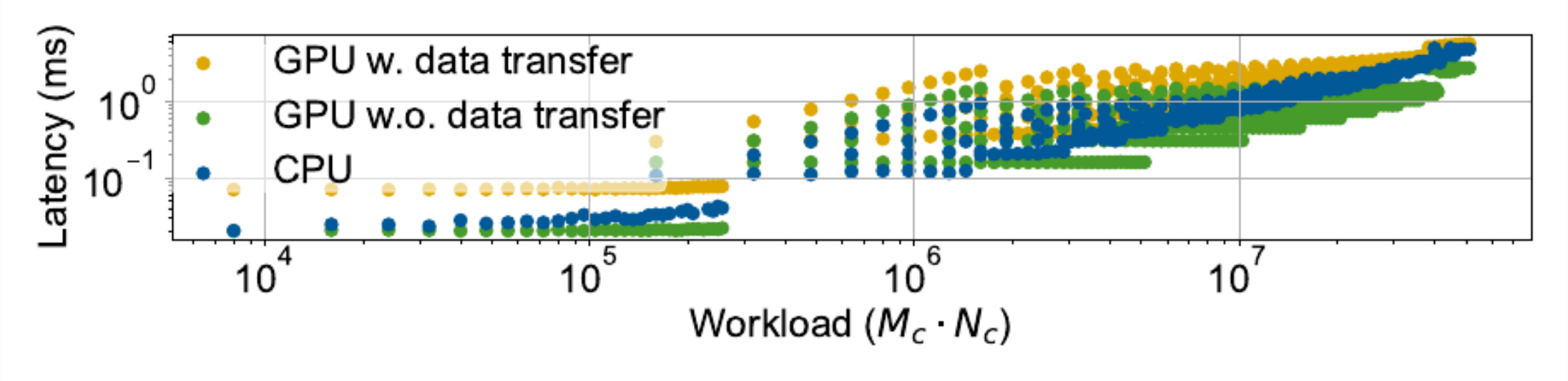
2.9× ① **AMX faster for small GEMMs**
PCIe transfers dominate GPU time

6.9× ② **GPU faster for large GEMMs**
GPU compute outweighs transfer cost

No single device always wins — schedule fine-search jobs according to their shape.

Top-k sorting should be co-located with its GEMM

Transferring the distance matrix across PCIe outweighs the cost of sorting



Top-k sort time on the CPU and the GPU across workload sizes (k = 10)

0.6–2.1× **GPU vs. CPU sorting, data already local**
No clear winner by shape

0.3–0.8× **GPU throughput when the matrix crosses PCIe**
Sorting on the other device loses

A job = GEMM + partial top-k, always on one device.

AMG overview

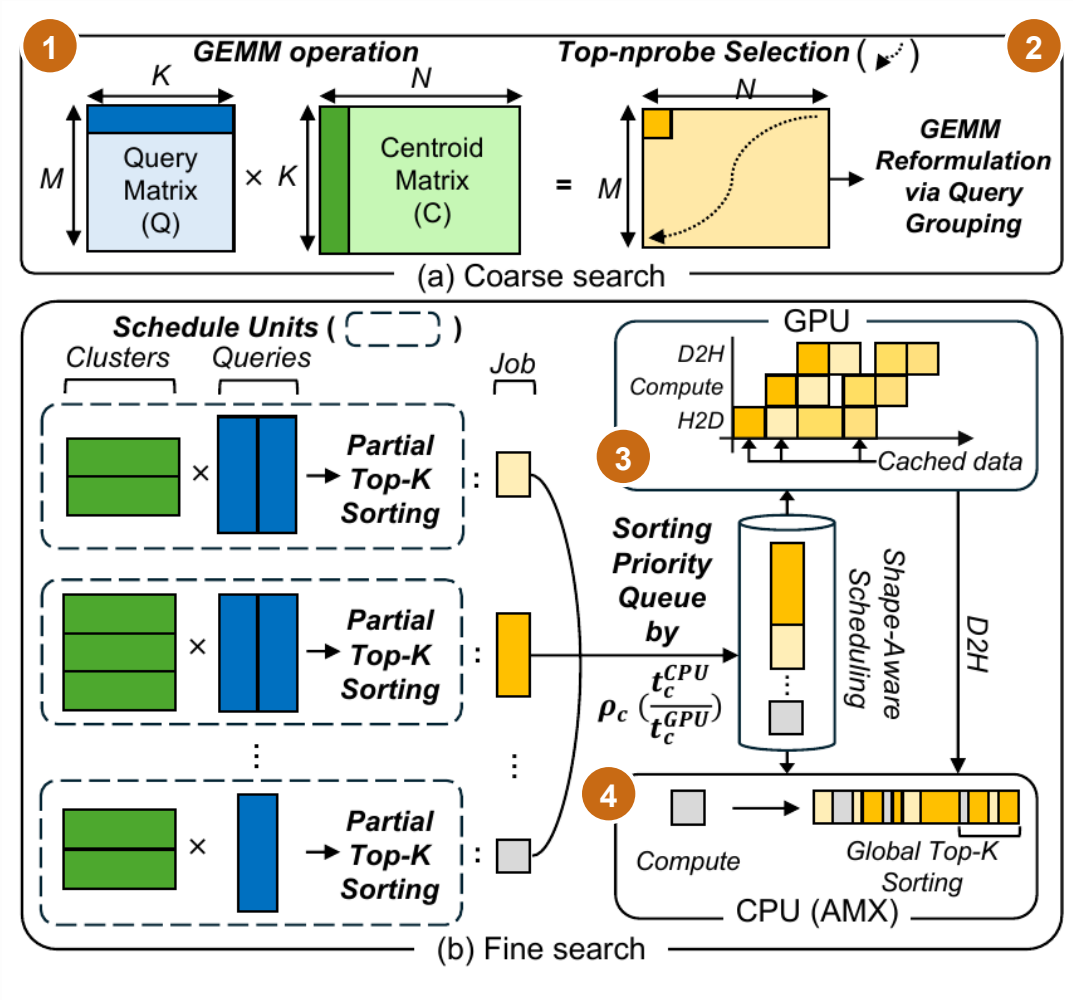
AMG = query grouping + shape-aware scheduling + GPU pipelining/caching

1. **GPU coarse search**
   Selects clusters for each query

2. **Query grouping**
   One job per cluster: GEMM + partial top-k

3. **Shape-aware scheduling**
   Each job goes to the GPU or AMX

4. **Global top-k**
   Merges partial results per query



Shape-aware scheduling

Each device pulls jobs from its own end of a shared queue

① Lightweight cost model

$$t_c^{\mathrm{CPU}} = \hat{T}_{\mathrm{GEMM}} + \hat{T}_{\mathrm{Topk}}$$

$$t_c^{\mathrm{GPU}} = \hat{T}_{\mathrm{H2D}} + \hat{T}_{\mathrm{GEMM}} + \hat{T}_{\mathrm{Topk}} + \hat{T}_{\mathrm{D2H}}$$

$$\hat{T}(M_c, N_c) = a_0 M_c + a_1 N_c + a_2 M_c N_c + a_3$$

Linear regression on job shape, fit offline

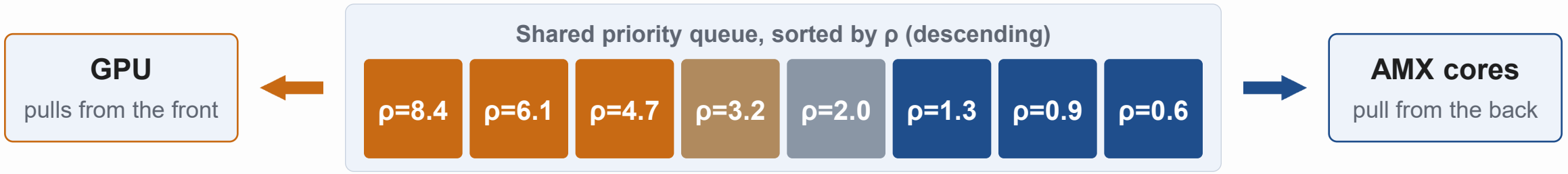
② Priority of a job

$$\rho_c = \frac{t_c^{\mathrm{CPU}}}{t_c^{\mathrm{GPU}}}$$

High → the GPU helps most
Low → AMX is a good fit

③ Work pulling

Each device takes the next job from its own end as soon as it is free. No fixed split, no idle device.



Dynamic job assignment keeps both devices busy without a fixed CPU–GPU split.

GPU pipelining & caching

Standard GPU optimizations — with a cache policy built for AMG

GPU pipelining

Overlap copying and computing across jobs; keep several jobs in flight

GPU cluster caching

Keep some clusters in GPU memory → no PCIe copy for them

Our cache policy: which clusters get GPU memory

$$s_c = \frac{t_c^{\mathrm{CPU}} - t_{c,\,\mathrm{hit}}^{\mathrm{GPU}}}{\text{cluster size}}$$

- Not “most used”, but “GPU time saved over AMX”
- Per byte: most benefit from limited GPU memory
- Ranked once, offline, since cluster contents are fixed at index build

Scheduler reuses it: cached cluster → transfer time = 0 → ρ goes up → the GPU takes the job first

Cache the clusters where the GPU saves the most time over AMX, per byte.

Experimental setup

Three billion-scale datasets, four baselines, one AMX server with one GPU

Evaluated system

CPU	2× Intel Xeon 6554S (Granite Rapids), 2.2 GHz, 36 cores each with AMX (72 cores total), Hyper-Threading off
GPU	NVIDIA RTX PRO 6000 (Blackwell), 96 GB GDDR7, connected over PCIe 5.0 ×16
Memory	1 TB host memory (16 × 64 GB DDR5-4800), 2 × 360 MB last-level cache
Software	Ubuntu 22.04; faiss v1.11.0 with nlist = 32K; GEMM kernels from cuBLAS (GPU) and oneMKL (CPU)

Baselines

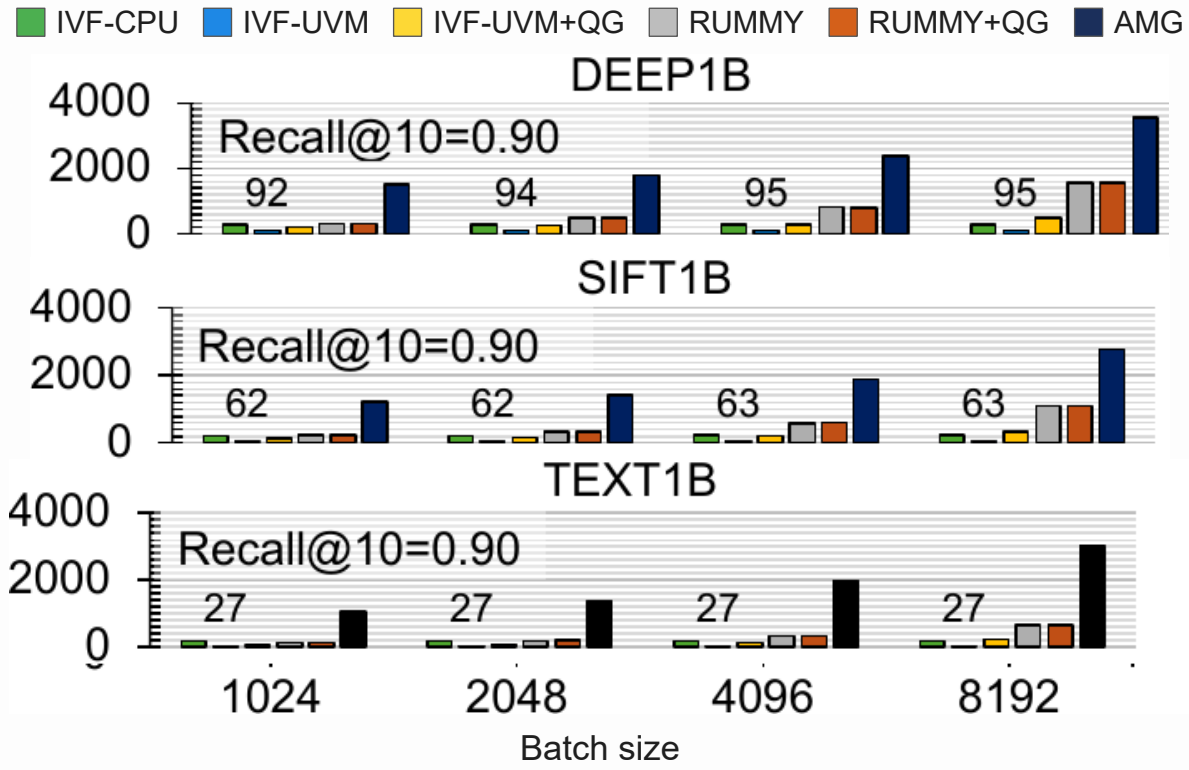
IVF-CPU	faiss on the CPU with AVX-512; no PCIe transfer, but limited compute
IVF-UVM	faiss GPU with CUDA unified memory; clusters fetched by page faults over PCIe
RUMMY	state-of-the-art single-GPU system [NSDI '24]; each cluster copied once, transfer overlapped with compute
+QG	IVF-UVM and RUMMY with our query grouping added (isolates the effect of grouping)

Datasets (1B vectors each, BF16) · metric: QPS at Recall@10 = 0.90, batch 1024 to 8192

DEEP1B	96-d image embeddings (Yandex), L2 distance, 187 GB
SIFT1B	128-d SIFT descriptors (INRIA), L2 distance, 246 GB
TEXT1B	200-d text-to-image embeddings (Yandex), cosine, 381 GB

End-to-end throughput

AMG is fastest on every dataset and batch size



Recall@10 = 0.90. Number above each group = lowest bar's QPS.
+QG = baseline with our query grouping

2.1–8.8×

higher QPS than RUMMY

state-of-the-art single-GPU system [NSDI '24]

5.4–26.6×

higher QPS than IVF-CPU

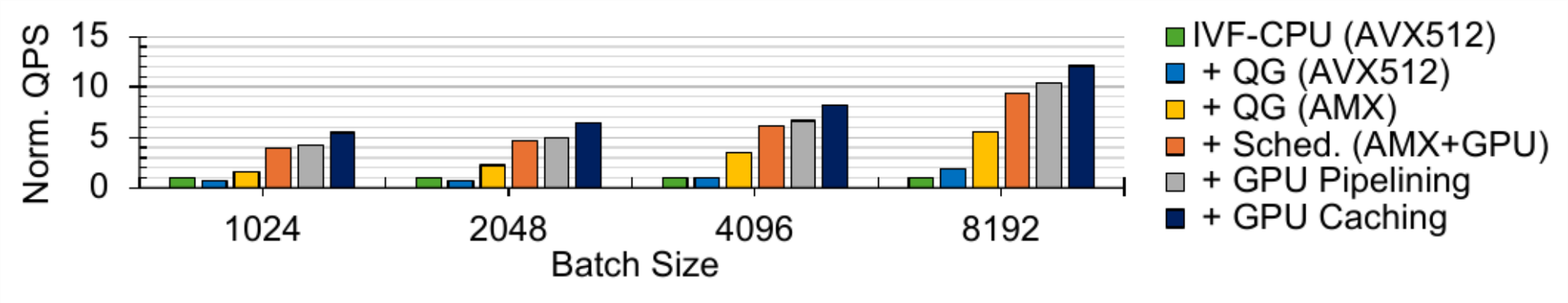
CPU-only baseline

2× Xeon 6554S (AMX) + RTX PRO 6000 (96 GB), PCIe 5.0

Query grouping alone does not help RUMMY (0.95–1.02×); the gain comes from AMX–GPU cooperation.

Performance breakdown by technique

Each technique adds on top of the previous one (SIFT1B)



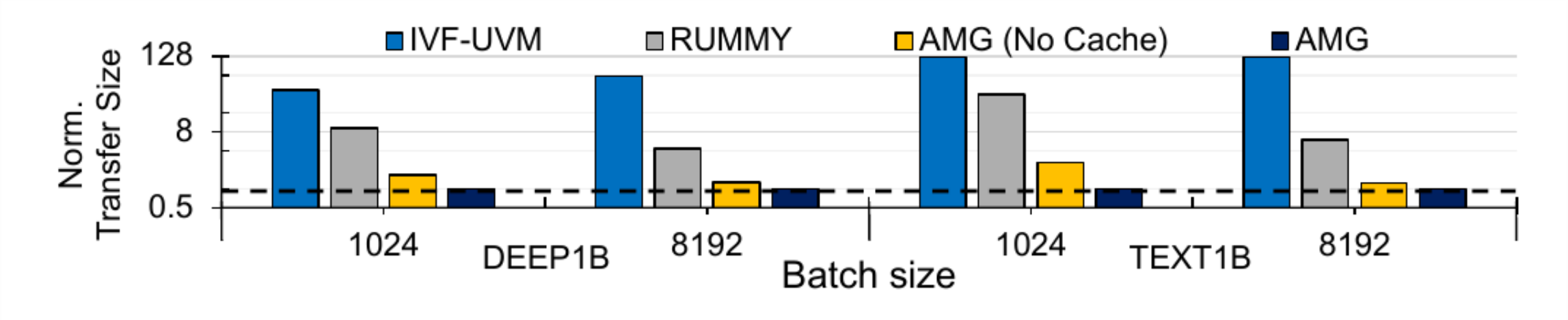
Cumulative QPS, normalized to IVF-CPU (each card: speedup over IVF-CPU; in parentheses: gain over the previous step)

IVF-CPU (AVX512)	+ QG (AVX512)	+ QG (AMX)	+ AMX–GPU cooperation	+ GPU Pipelining	+ GPU Caching
1×	0.7–1.8×	1.6–5.6×	4.0–9.4×	4.3–10.4× (+7–10%)	5.6–12.1× (+17–29%)

Grouping makes AMX fast; AMX–GPU cooperation turns that into end-to-end gains.

AMG significantly reduces PCIe traffic

AMX handles the many small jobs; the GPU takes the FLOP-intensive ones



Total PCIe transfer volume per batch, normalized to AMG

4.4–31.2×

less PCIe traffic than RUMMY

unified memory moves up to 123× more

11–19%

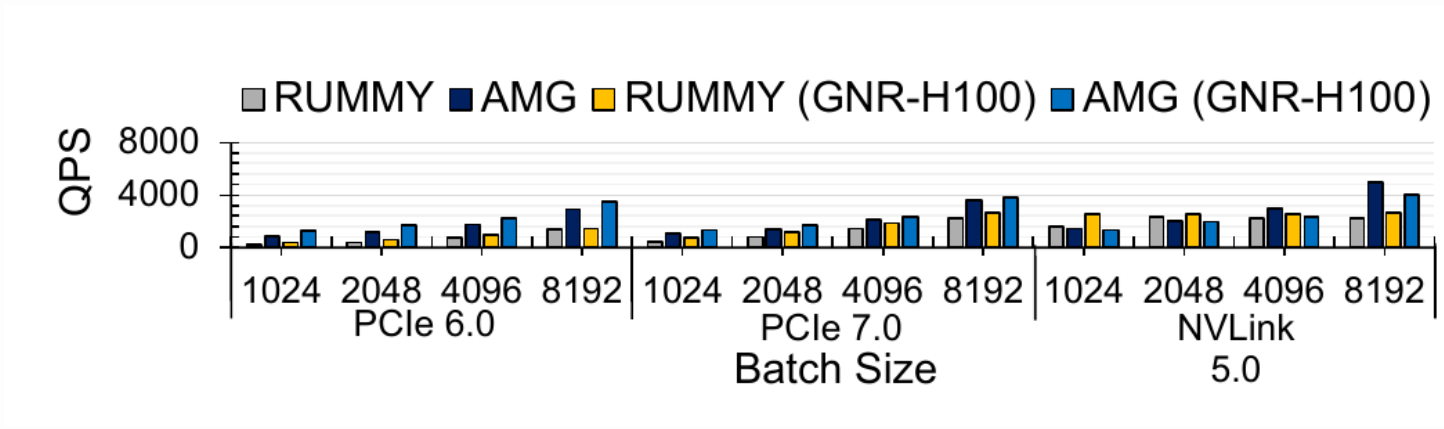
of cluster data crosses PCIe

yet the GPU still runs 53–66% of the FLOPs

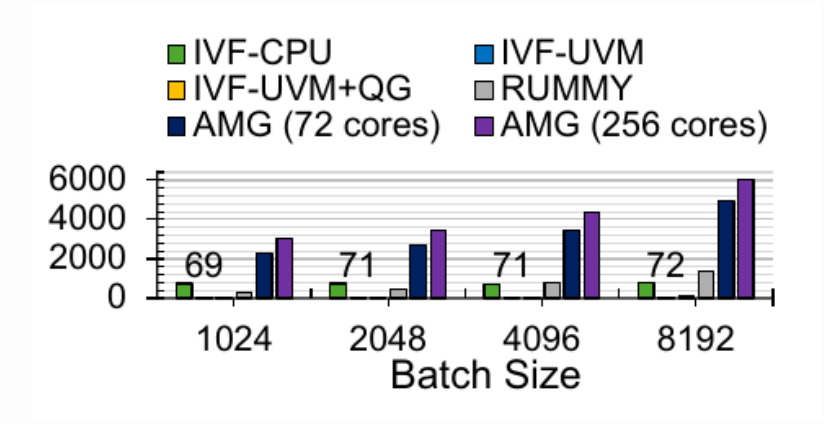
Most jobs run on AMX, where the data resides; only FLOP-intensive jobs cross PCIe.

Robustness to faster interconnects and a stronger GPU

Emulated PCIe 6.0 / 7.0 and NVLink 5.0, and a Granite Rapids + H100 server



AMG vs. RUMMY on TEXT1B under emulated PCIe 6.0, PCIe 7.0 and NVLink 5.0



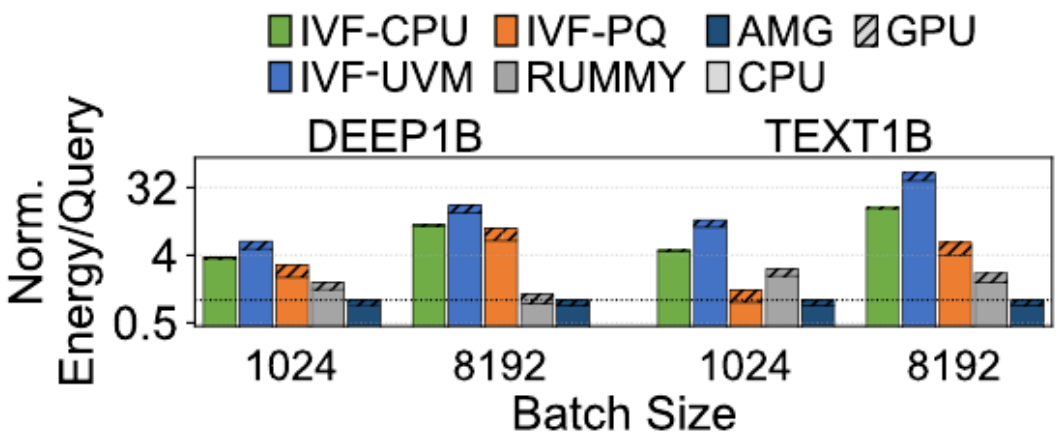
End-to-end throughput on Granite Rapids + H100 80 GB (SIFT1B)

1.4–3.7× **over RUMMY on PCIe 6.0 / 7.0**
NVLink 5.0: RUMMY saturates; AMG reaches 2.2× at batch 8192

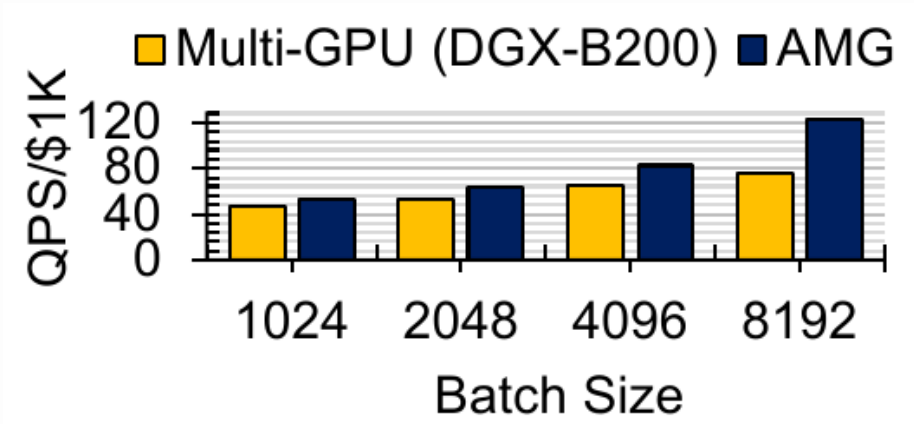
3.7–6.7× **over RUMMY on GNR + H100**
256 AMX cores add another 22–35%

Energy and cost efficiency

Less energy per query than RUMMY; more throughput per dollar than a DGX-B200



Energy per query, normalized to AMG (RAPL + NVML)



QPS per $1K of 3-year TCO on SIFT1B: AMG (one RTX PRO 6000) vs. DGX-B200 (8× B200)

1.2–2.6× **less energy per query than RUMMY**

84–85% of AMG's energy goes to the CPU

1.16–1.62× **higher QPS per dollar than an 8-GPU DGX-B200**

ANNS needs memory capacity more than compute

Conclusion

AMG makes AMX and the GPU work together for billion-scale ANNS

1 Key Challenge

PCIe bottlenecks motivate CPU offloading, but per-query GEMV cannot use AMX efficiently.

>80%

of RUMMY runtime is PCIe transfer

2 Query Grouping

Per-query GEMVs become per-cluster GEMMs, so AMX pays off.

1.1× → 3.3×

AMX over AVX-512: GEMV → GEMM

3 AMX–GPU Cooperation

Shape-aware scheduling and GPU caching move less data over PCIe.

4.0–9.4×

over IVF-CPU, with cooperation

Result: 2.1–8.8× higher QPS than a state-of-the-art single-GPU system (RUMMY)

Thank You

Questions are welcome

Code and experiment scripts: github.com/minlno/MICRO-2026-AMG

Artifact: doi.org/10.5281/zenodo.21523619

Daehoon Kim · daehoonkim@yonsei.ac.kr