Accelerating Billion-Scale ANNS On Modern Hardware

Jayjeet Chakraborty, Heiner Litz
Center for Research in Systems and Storage
University of California, Santa Cruz

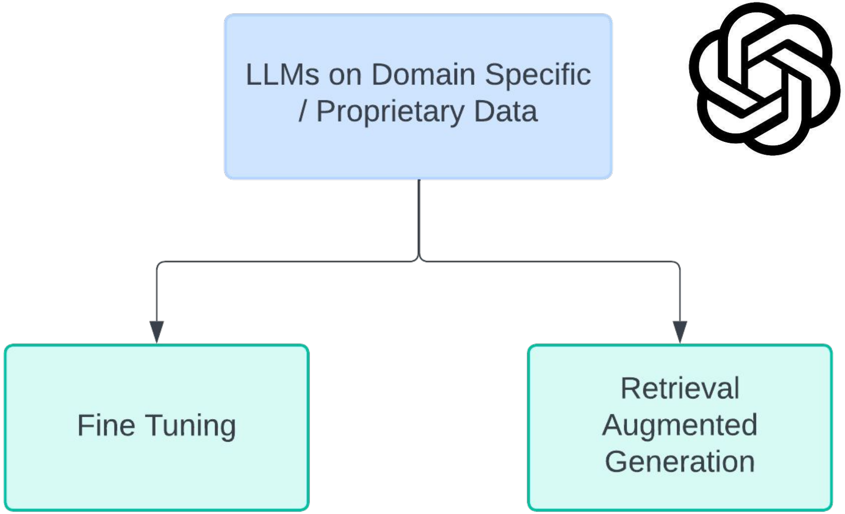
UC SANTA CRUZ
BaskinEngineering

Center for Research
in Systems and Storage

NSF

CRSS CONFIDENTIAL

Custom Language Models



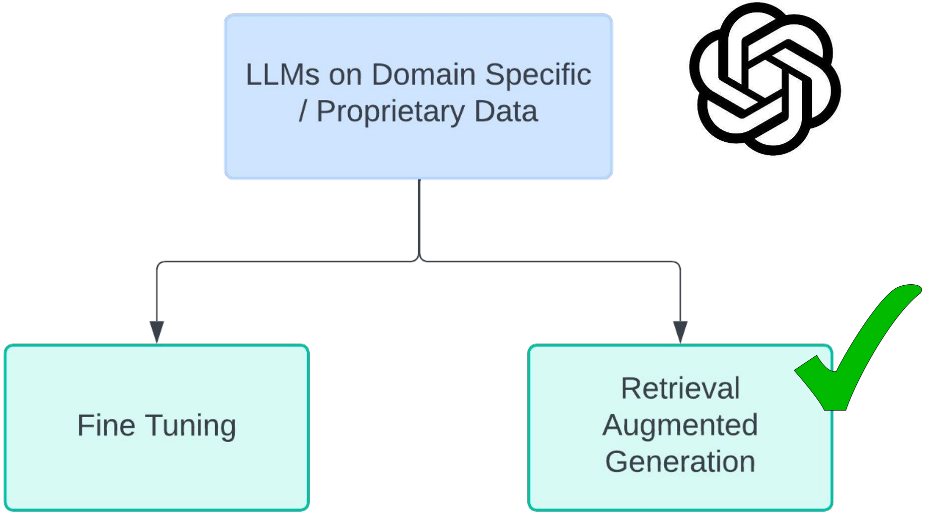
Fine Tuning

- High Infra Costs
- Time Consuming
- Training Data Dependent
- Loses Generic Capabilities

Retrieval Augmented Generation

- Minimal Costs
- Flexible Sources of Knowledge
- Retains Original Model Capabilities

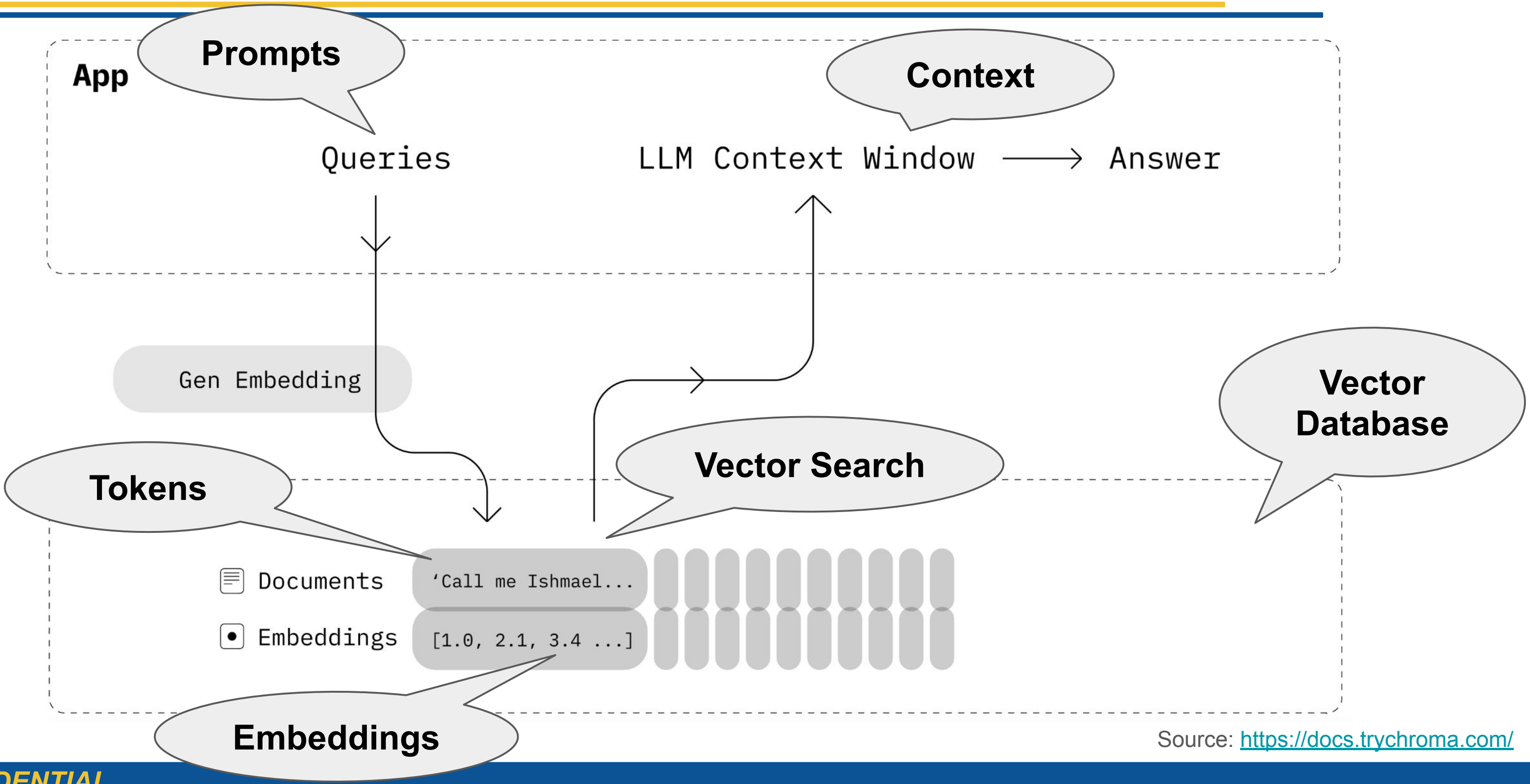
Custom Language Models



- High Infra Costs
- Time Consuming
- Training Data Dependent
- Loses Generic Capabilities

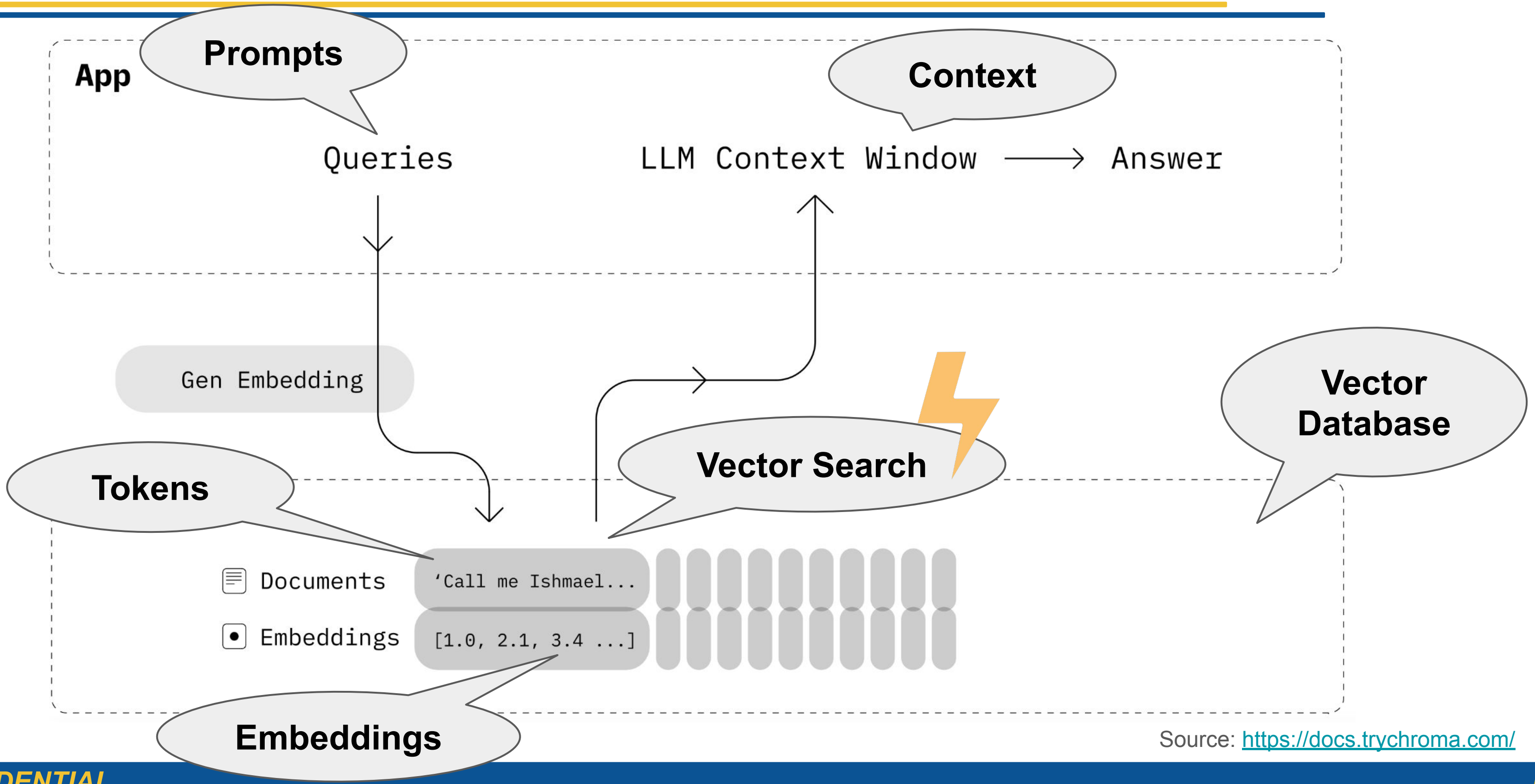
- Minimal Costs
- Flexible Sources of Knowledge
- Retains Original Model Capabilities

Retrieval Augmented Generation



Source: https://docs.trychroma.com/

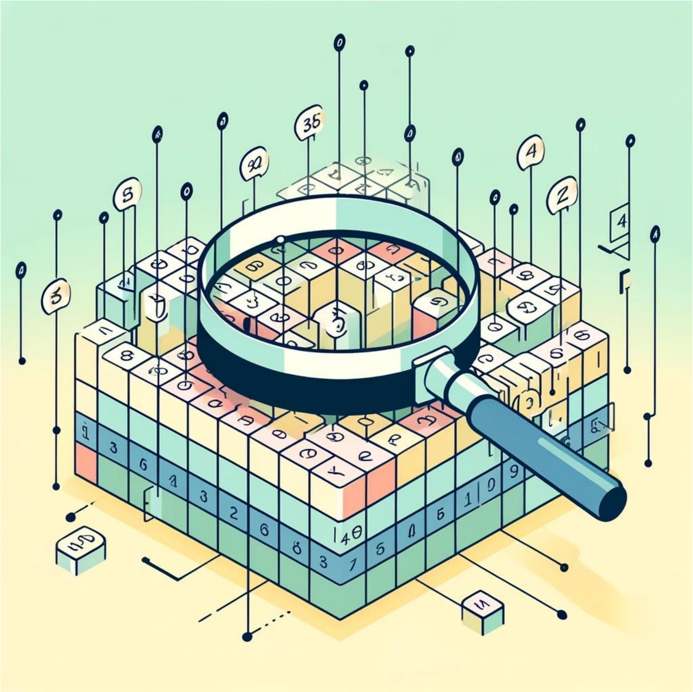
Retrieval Augmented Generation



Source: https://docs.trychroma.com/

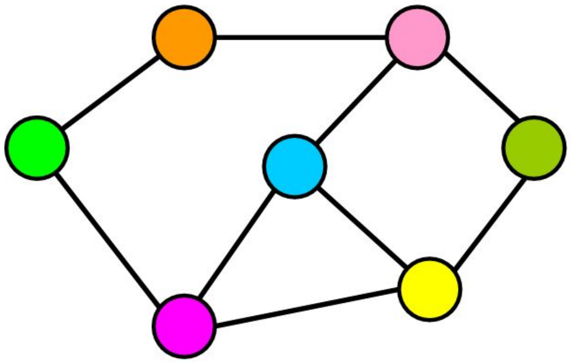
Vector Search

- ❖ Find tokens similar to a query using nearest neighbor searches
- ❖ Traditionally, **KNN** has been used
  - ➢ But on millions & billions of data points, not feasible
- ❖ Using **ANN** (Approximate Nearest Neighbor) algorithms allows trading off **accuracy** for **search speed**

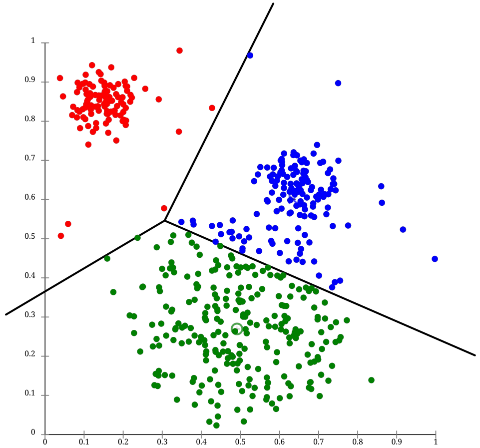


Source: Generated by DALL-E

Categories of ANN Algorithms

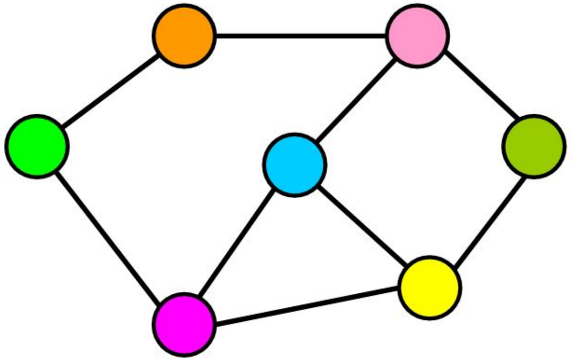


Graph-based

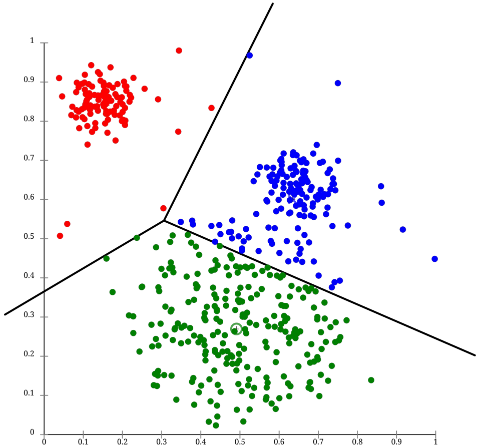


Cluster-based

Categories of ANN Algorithms

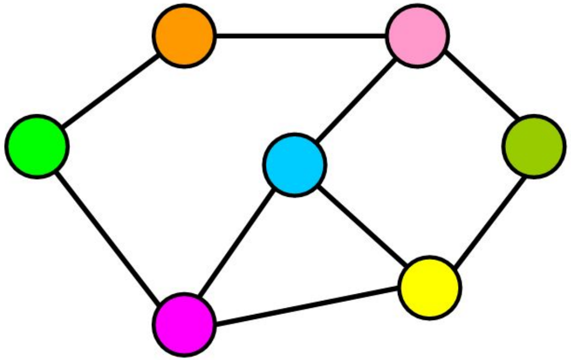


Ex: NSG, HNSW

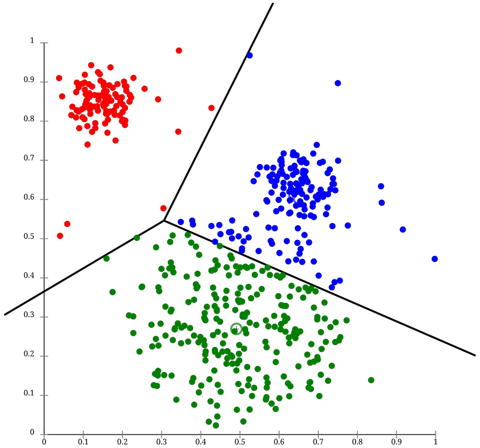


Ex: IVF, IVF-PQ

Categories of ANN Algorithms

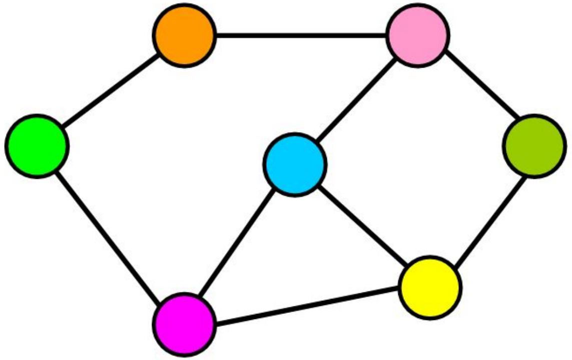


Branchy Character,
Better suited for **CPUs**

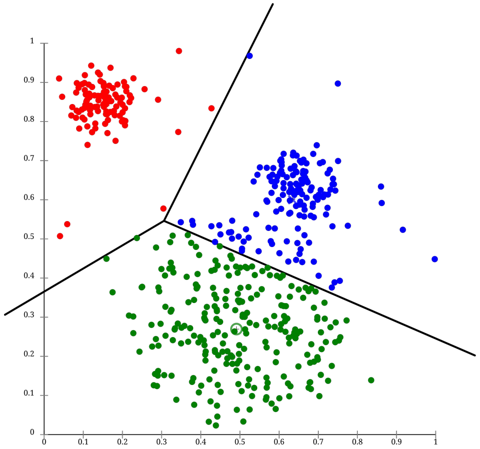


Data Parallel Character,
Better suited for **GPUs**

Categories of ANN Algorithms

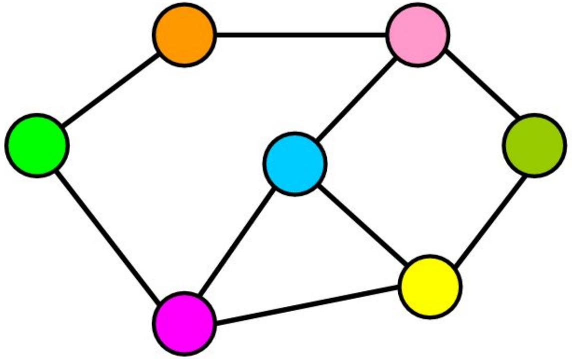


Inter-query parallelism
& limited Intra-query
parallelism

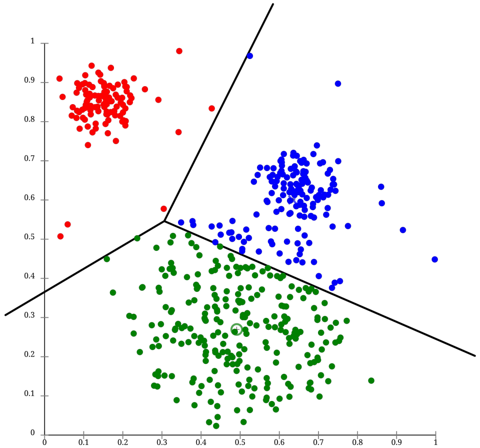


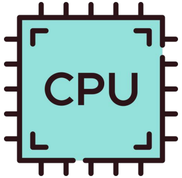
Good Intra & Inter-query
parallelism

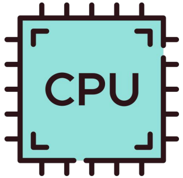
Performance Characteristics



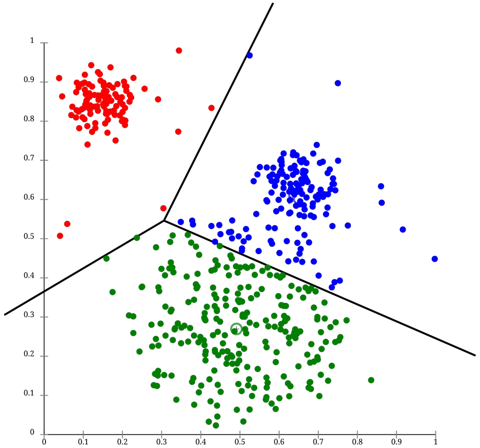
faster than

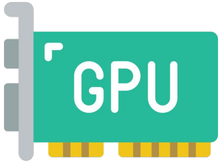




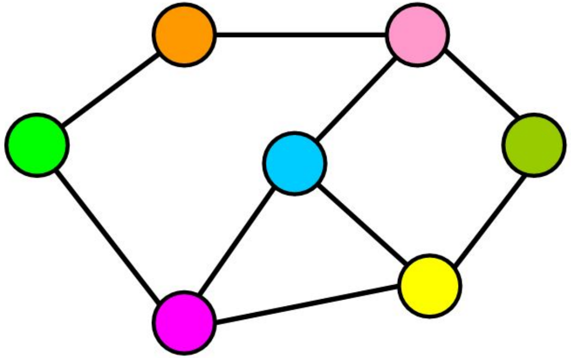


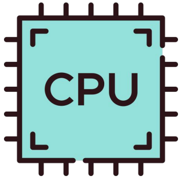
Performance Characteristics



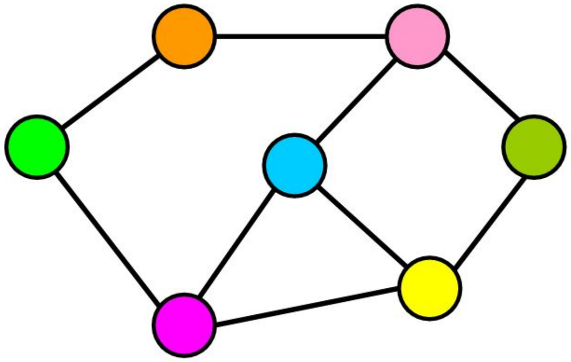


faster than

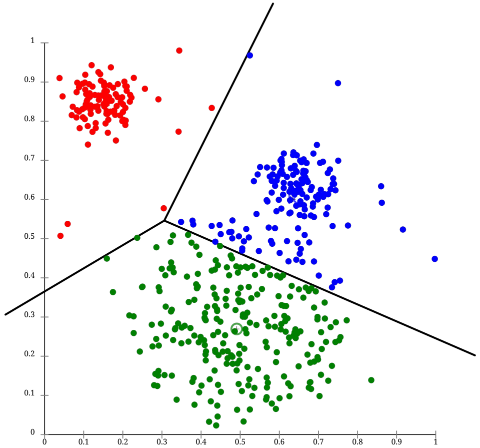


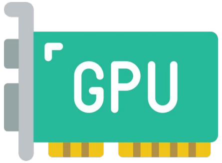


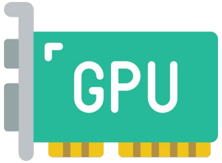
Performance Characteristics



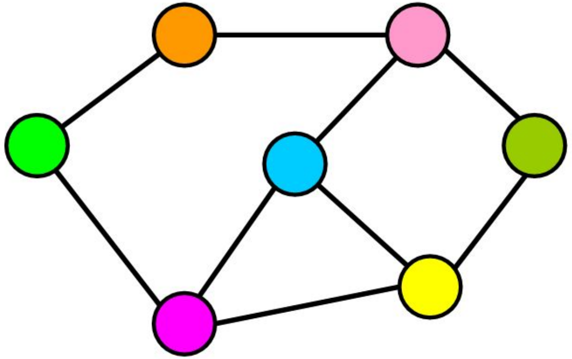
faster than

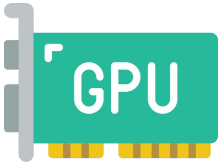






SOTA





Parallel Graph Construction and Search

Hand-tuned Kernels

GPU-friendly Data Structures

Utilizing register files, shared memory

Multi-stage pipelined execution using streams

SOTA

CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs

SONG: Approximate Nearest Neighbor Search on GPU

GGNN: Graph-based GPU Nearest Neighbor Search

GPU-accelerated Proximity Graph Approximate Nearest Neighbor Search and Construction

SOTA?

~~CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs~~

~~SONG: Approximate Nearest Neighbor Search on GPU~~

~~GGNN: Graph-based GPU Nearest Neighbor Search~~

~~GPU Accelerated Proximity Graph Approximate Nearest Neighbor Search and Construction~~

SOTA?

CAGRA: Highly Parallel Graph Con… and Approximate Nearest Neigh…

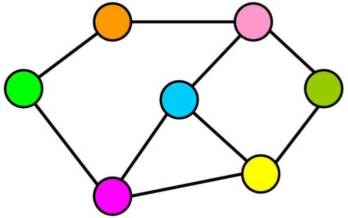
SONG: Approxi… GPU

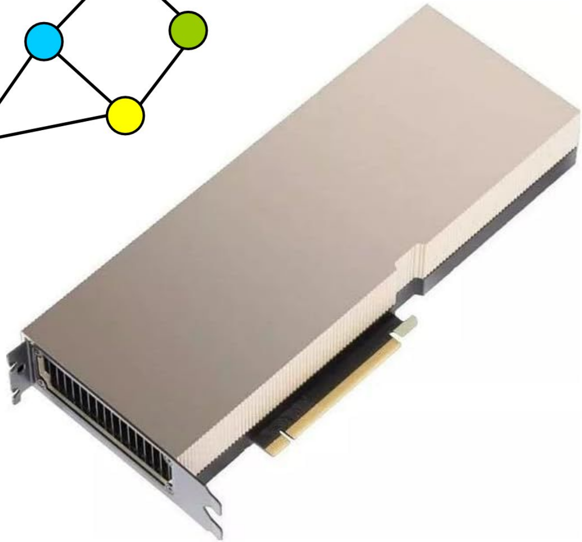
GGNN: Graph-based … Sea…

GPU-accelerated Prox… …imate Nearest Neighbor Search and Construction

All of these approaches assume *a single GPU* and that *datasets fit inside the GPU memory*

Limits of SOTA





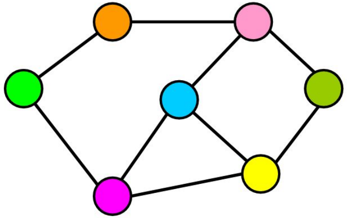
NVIDIA H100 80GB

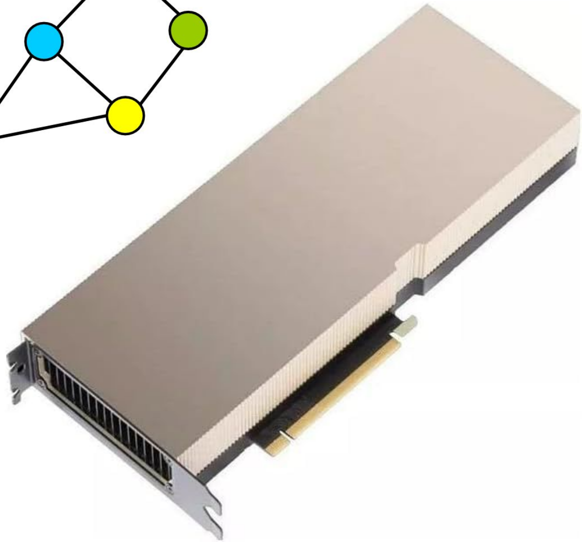
5M OpenAI Large Embeddings

10M OpenAI Small Embeddings

200M DEEP 1B Embeddings

Limits of SOTA





NVIDIA H100 80GB

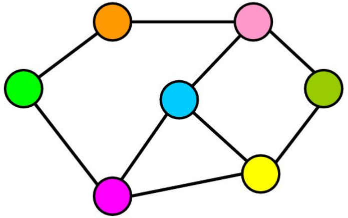
5M OpenAI Large Embeddings

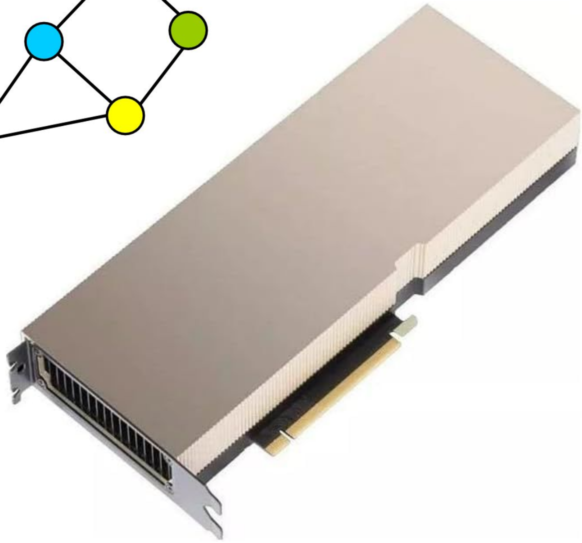
10M OpenAI Small Embeddings

200M DEEP 1B Embeddings

What about 1B vectors ?

Limits of SOTA





NVIDIA H100 80GB

5M OpenAI Large Embeddings

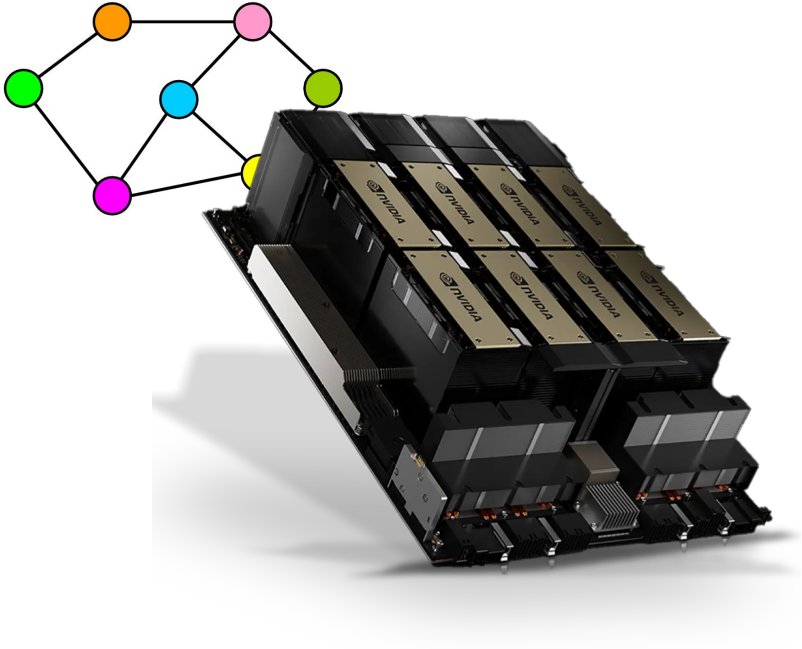
10M OpenAI Small Embeddings

200M DEEP 1B Embeddings

What about 1B vectors ?

PQ hurts accuracy and requires reranking

Limits of SOTA

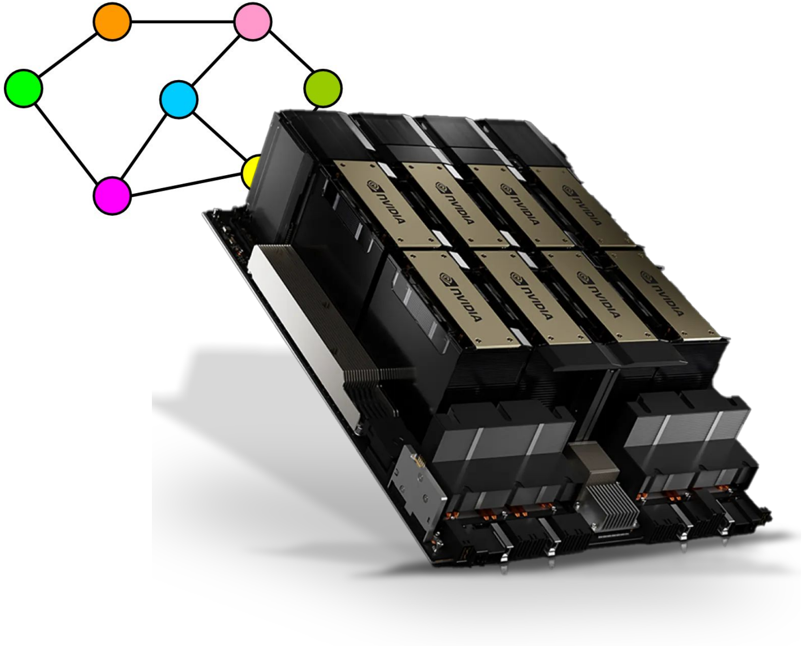


DGX H100 SXM 640GB

Running Graph-based algorithms on multiple GPUs is a huge inter-gpu **coordination** and **communication** overhead !

Poor scalability :(

Limits of SOTA



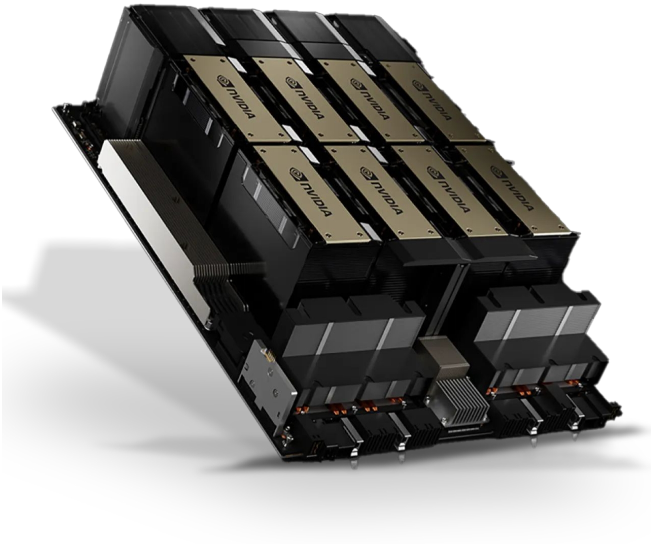
DGX H100 SXM 640GB

Running Graph-based algorithms on multiple GPUs is a huge inter-gpu **coordination** and **communication** overhead !

Poor scalability :(

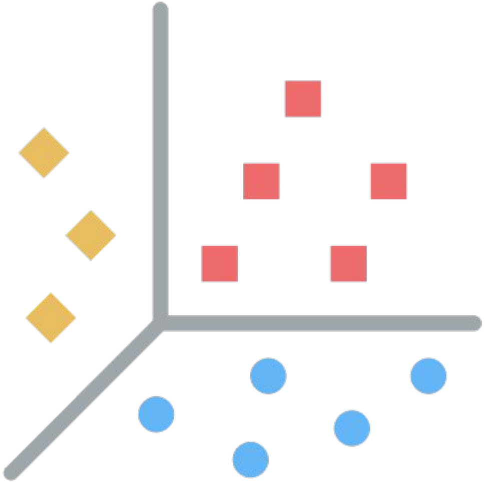
Low GPU utilization

Billion-Scale Searches



DGX H100 SXM 640GB

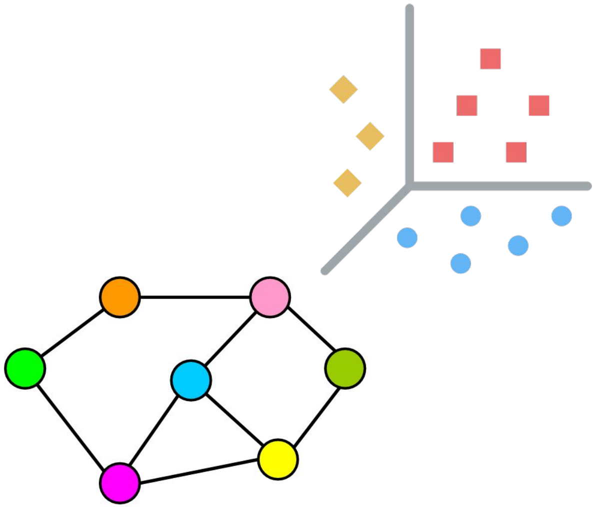
+



Cluster-based

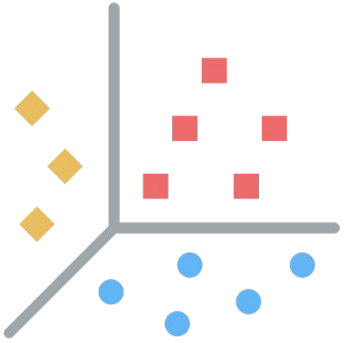
Billion-Scale Searches

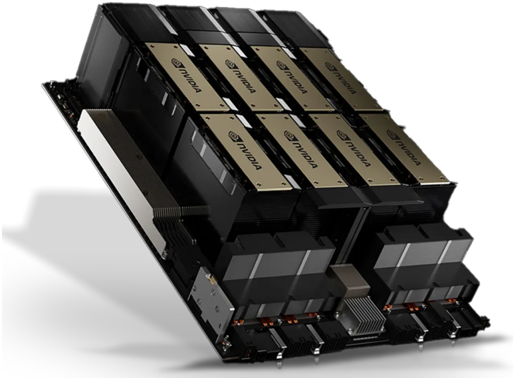
Multi-Core CPU + DRAM + GPU



Hybrid Graph + Cluster Index

Billion-Scale Searches



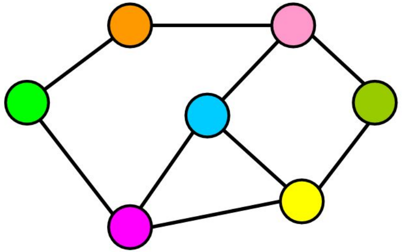


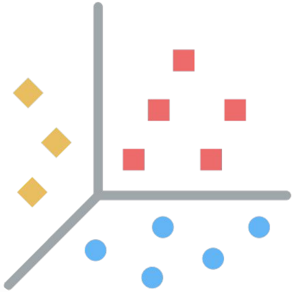
Highly parallel algorithm and Massively parallel hardware; Good scalability

Faster interconnect and higher bandwidth memory helps

Performance at a very very high cost
Each DGX is **~$300,000**

Billion-Scale Searches



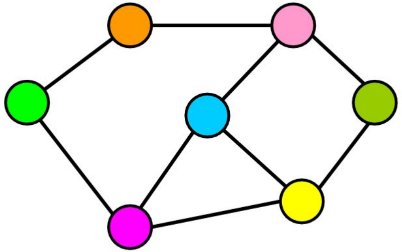


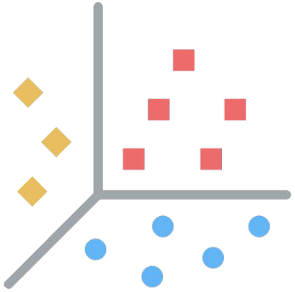
Use Hybrid Graph + Cluster based algorithms. HNSW-IVF-PQ ?

Let the CPU and GPU do at what they are really good at

Sapphire Rapids CPU + H100 GPU + 512 GB DDR5 memory < $50,000

Billion-Scale Searches





Use Hybrid Graph + Cluster based algorithms. HNSW-IVF-PQ ?

Let the CPU and GPU do at what they are really good at

Sapphire Rapids CPU + H100 GPU + 512 GB DDR5 memory < $50,000

Unified Virtual Memory (UVM)

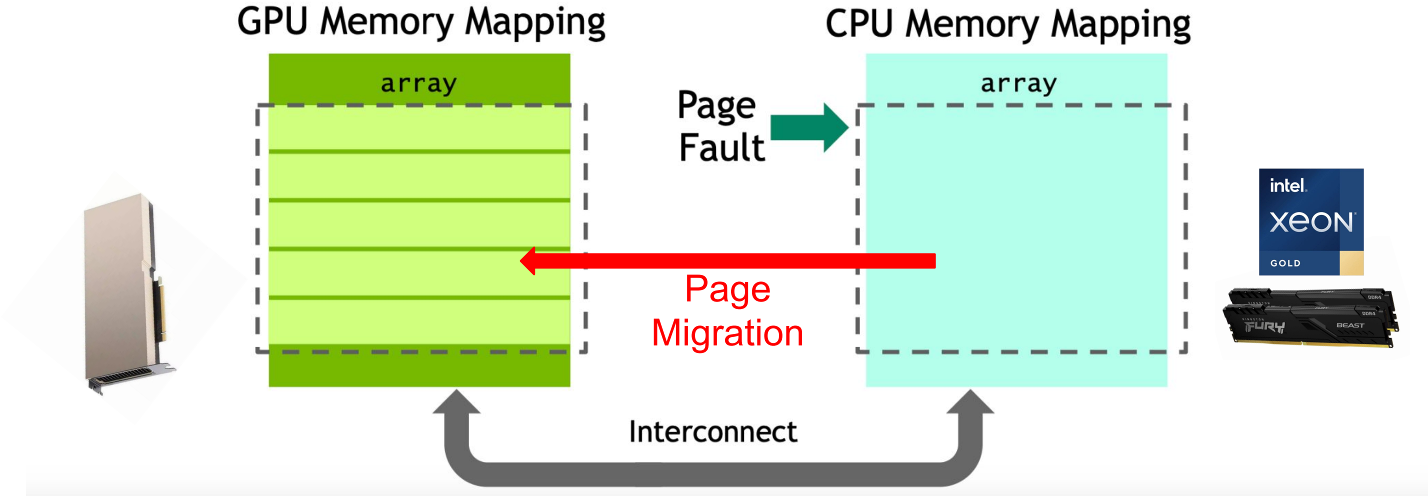
CUDA Unified Virtual Memory

GPU Code

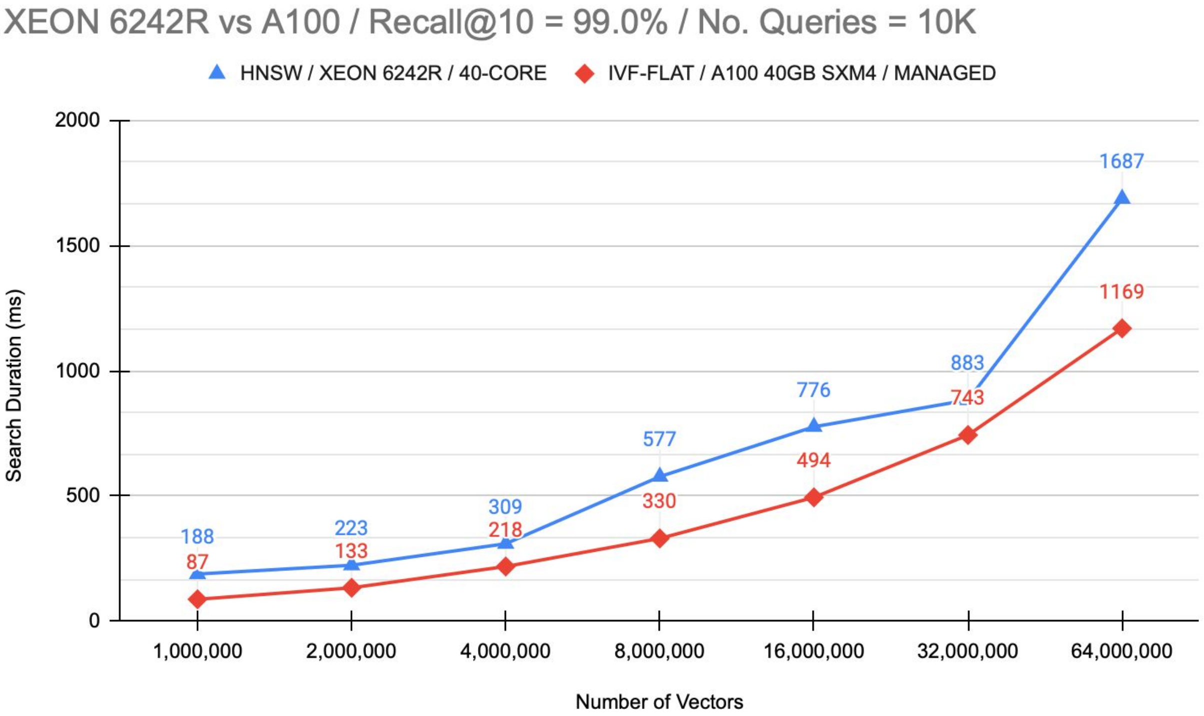
```
__global__
void setValue(char *ptr, int index, char val)
{
  ptr[index] = val;
}
```

CPU Code

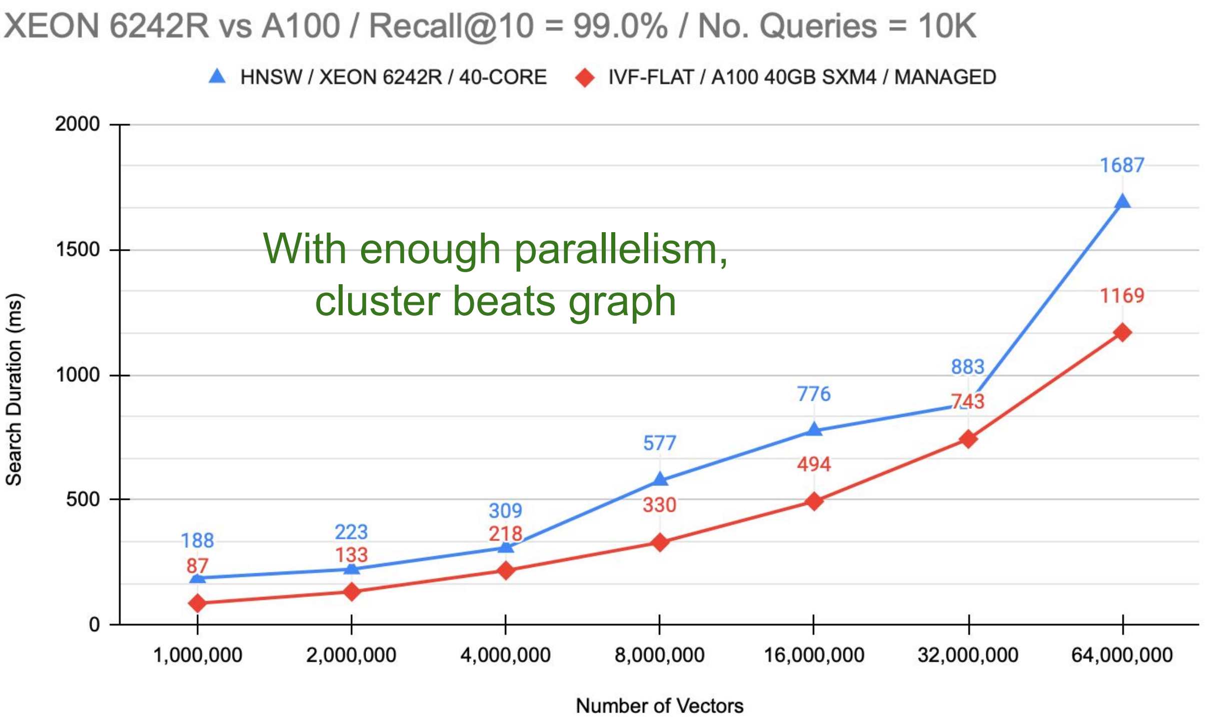
```
cudaMallocManaged(&array, size);
memset(array, size);
setValue<<<...>>>(array, size/2, 5);
```



CPU (Graph) vs. GPU (Cluster)

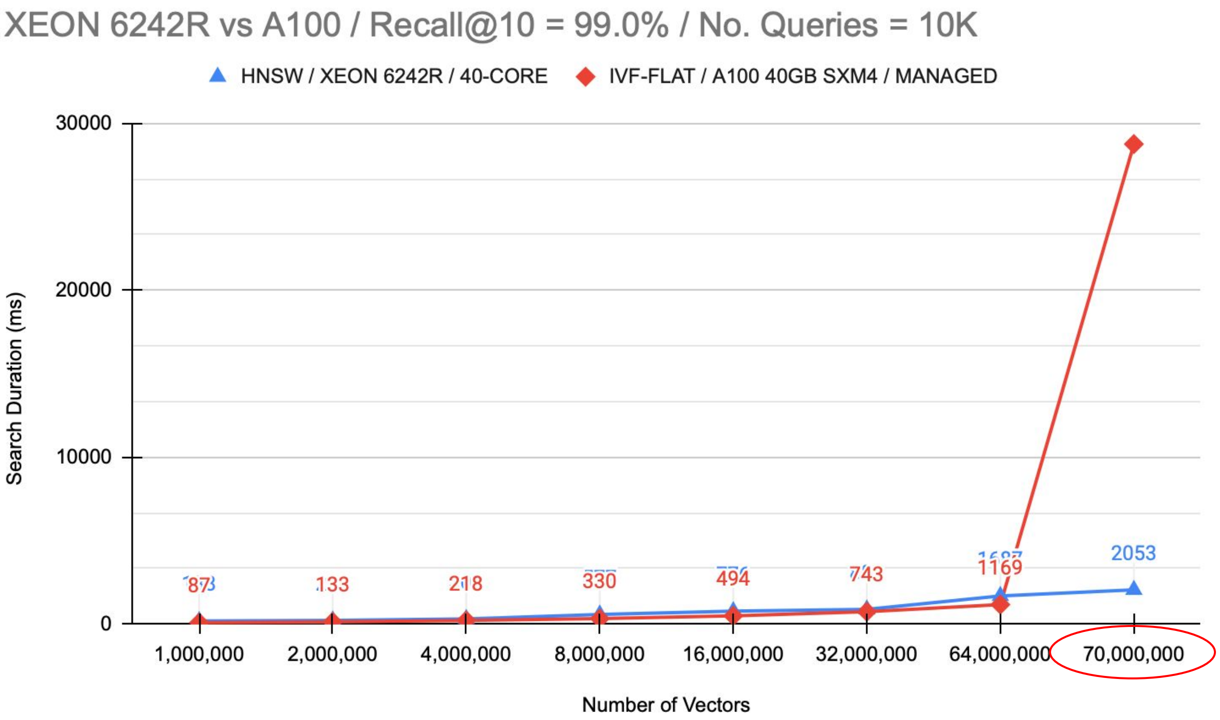


CPU (Graph) vs. GPU (Cluster)

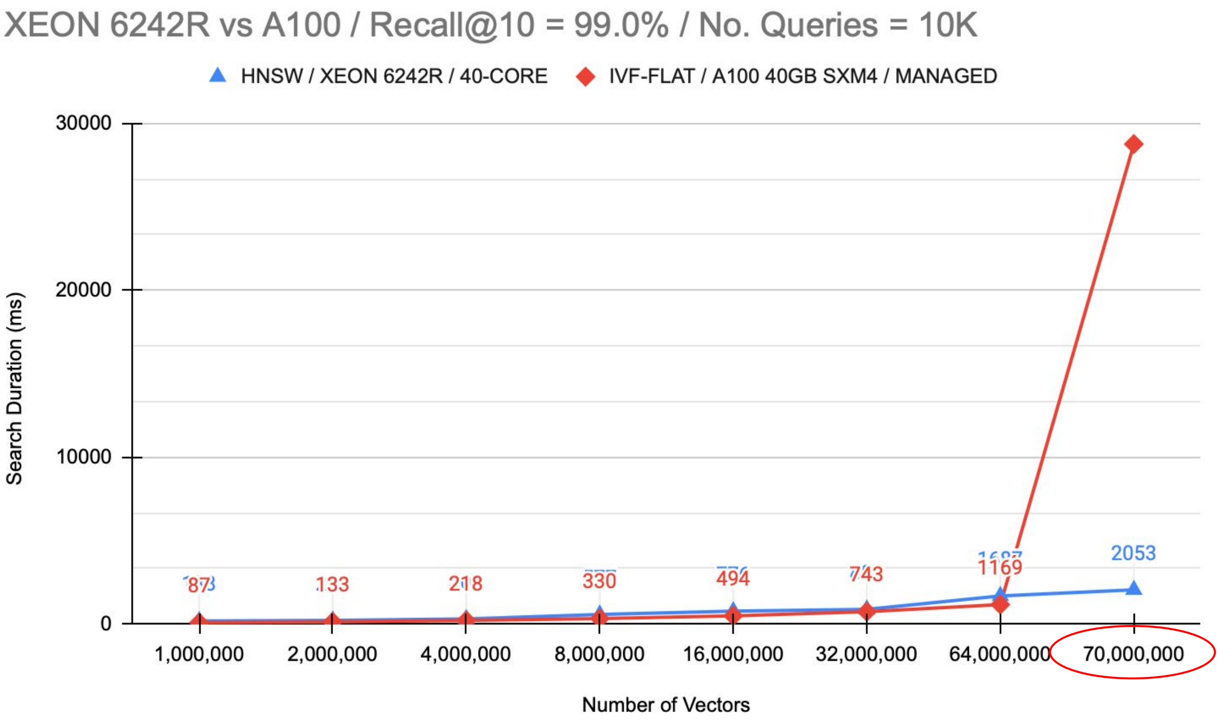


CPU (Graph) vs. GPU (Cluster)

CRSS

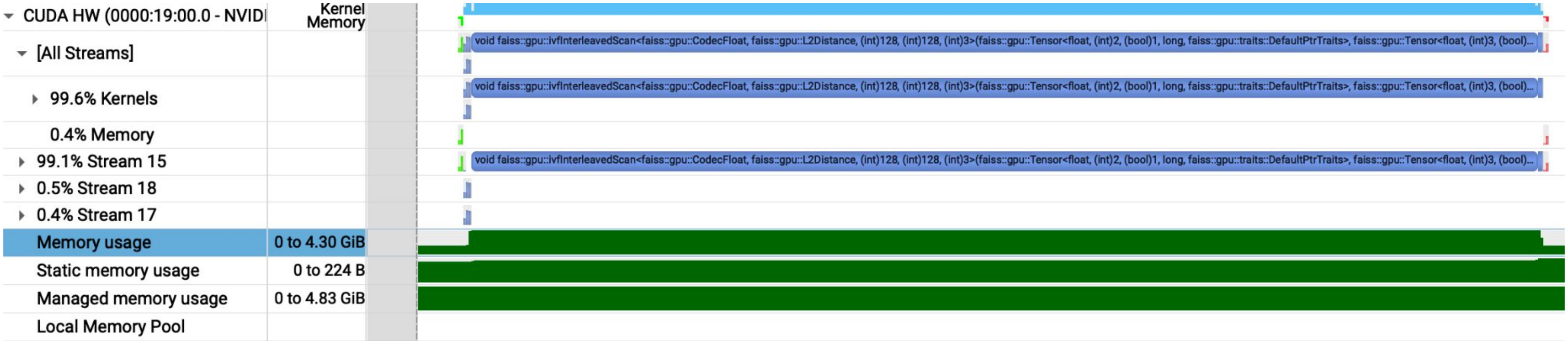


CPU (Graph) vs. GPU (Cluster)



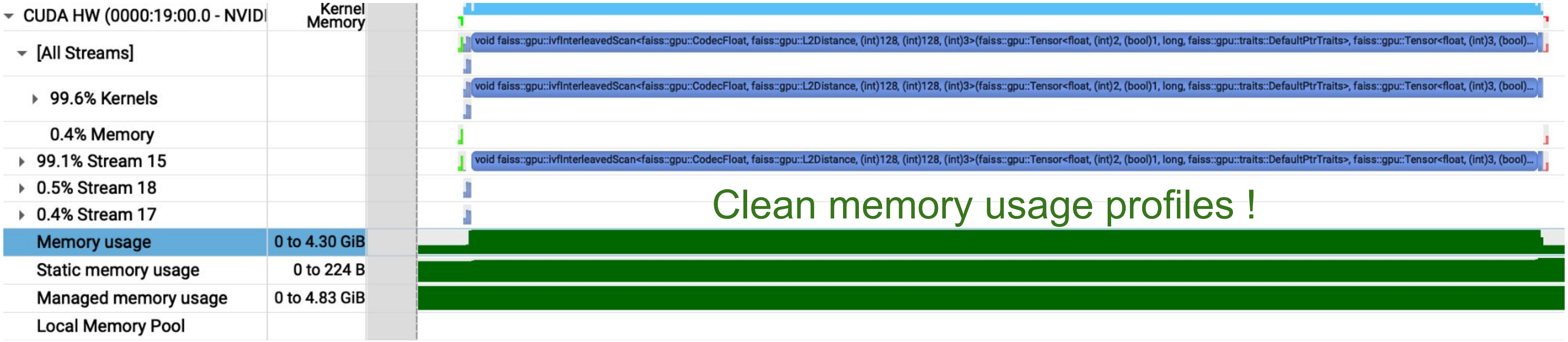
When dataset does not fit, exponential slow down in GPU

Profiling UVM



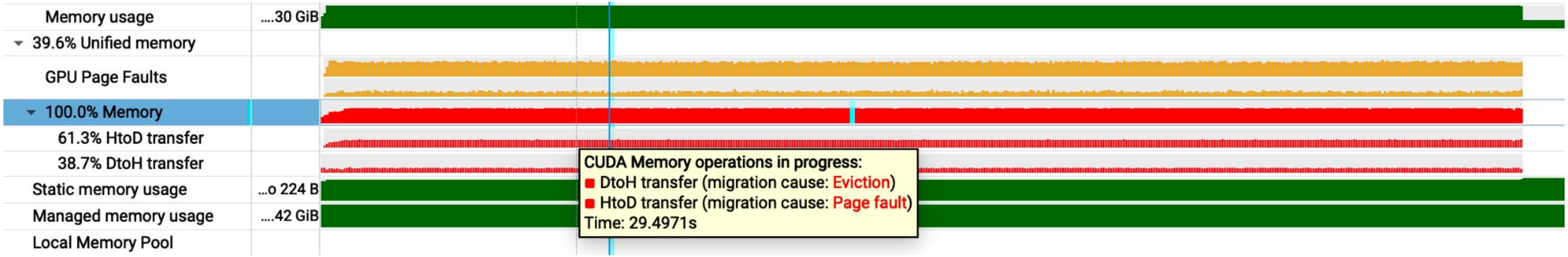
nsys profile when the dataset **fits** entirely in the GPU memory

Profiling UVM



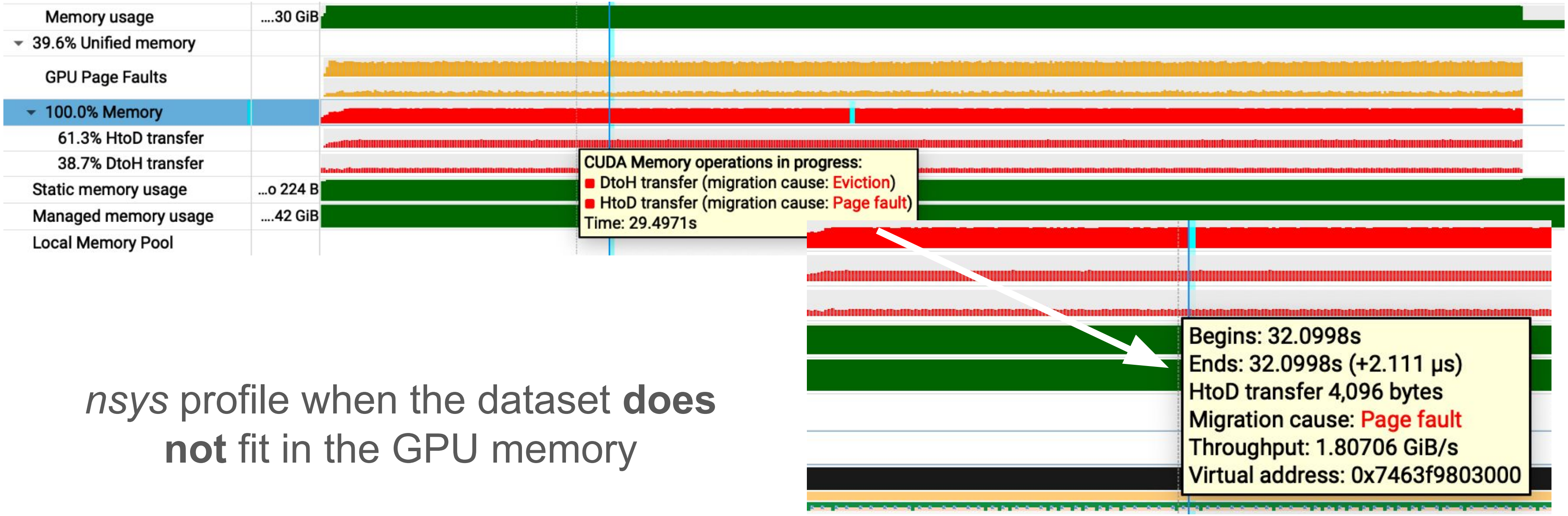
nsys profile when the dataset **fits** entirely in the GPU memory

Profiling UVM



nsys profile when the dataset **does not** fit in the GPU memory

Profiling UVM



nsys profile when the dataset **does not** fit in the GPU memory

Memory Management Modes in CUDA

cuda + pool

no. of vectors	cuda	async	managed	managed_pool	prefetch	prefetch_pool	n-probe
1,000,000	357	361	410	402	369	361	135
2,000,000	468	465	536	521	476	464	140
4,000,000	621	619	718	704	629	617	145
8,000,000	848	839	979	961	861	858	150
10,000,000	957	956	1111	1082	974	968	155
11,000,000	992	1011	1162	1132	1015	1009	155
12,000,000	0	0	1287	1312	2067	6109	155
14,000,000	0	0	2545	2582	1566	2408	160
16,000,000	0	0	2782	2230	2461	2166	160
18,000,000	0	0	3094	2757	2864	2737	160

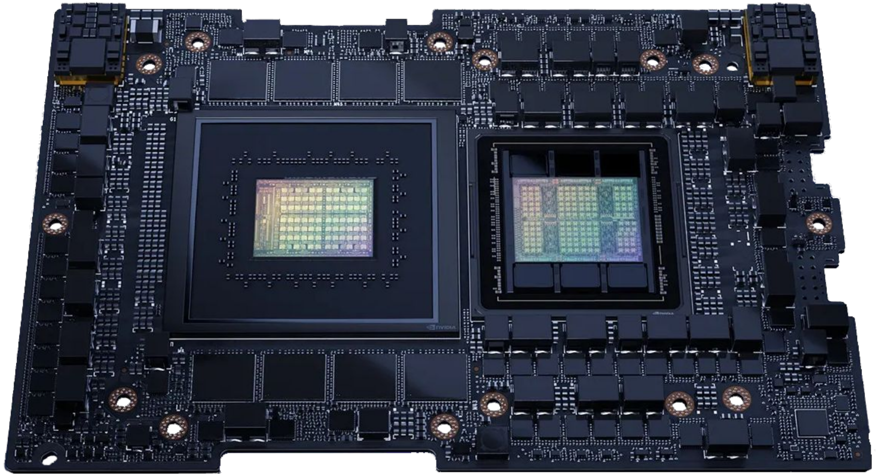
RECALL@10 = 99.0%

No. of Queries = 10,000

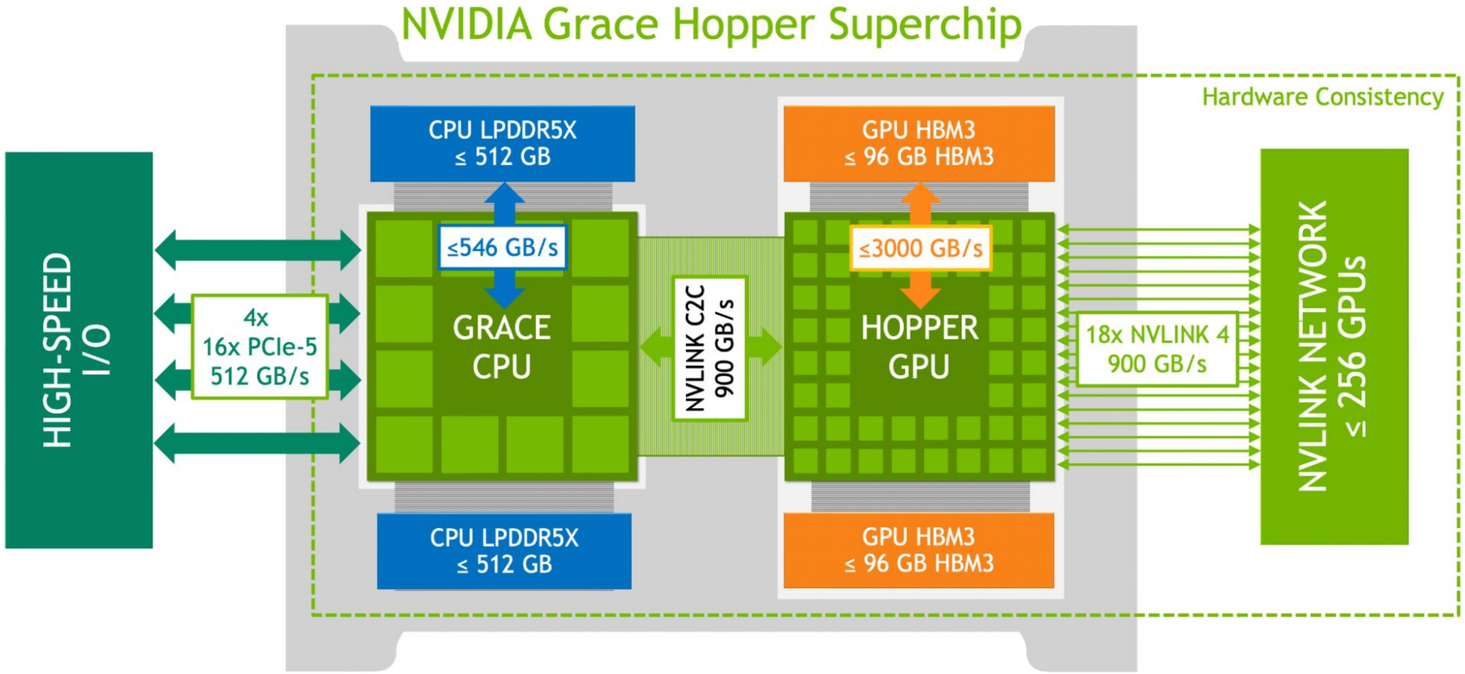
In **prefetch**, we try to prefetch every pointer allocated through **cudaMallocManaged**

NVIDIA Grace Hopper Superchip

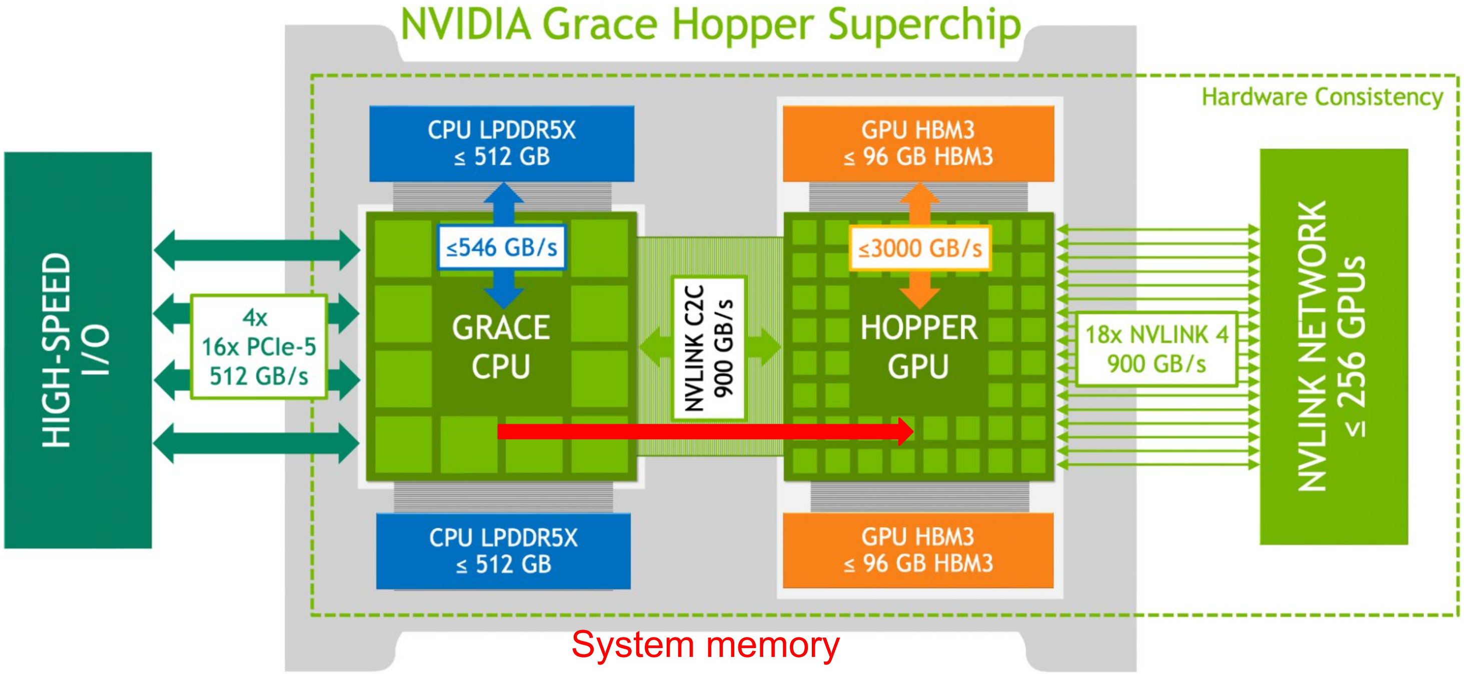
- **Grace CPU**
  - 72 ArmV9 Neoverse v2 cores
  - LPDDR5X 480 GB ECC memory
- **Hopper GPU**
  - H100 Tensor Core NVL
  - 96 GB HBM3e
- **C2C-NVLink**
  - Cache-coherent
  - 900 GB/s total bandwidth



Architecture of Grace Hopper

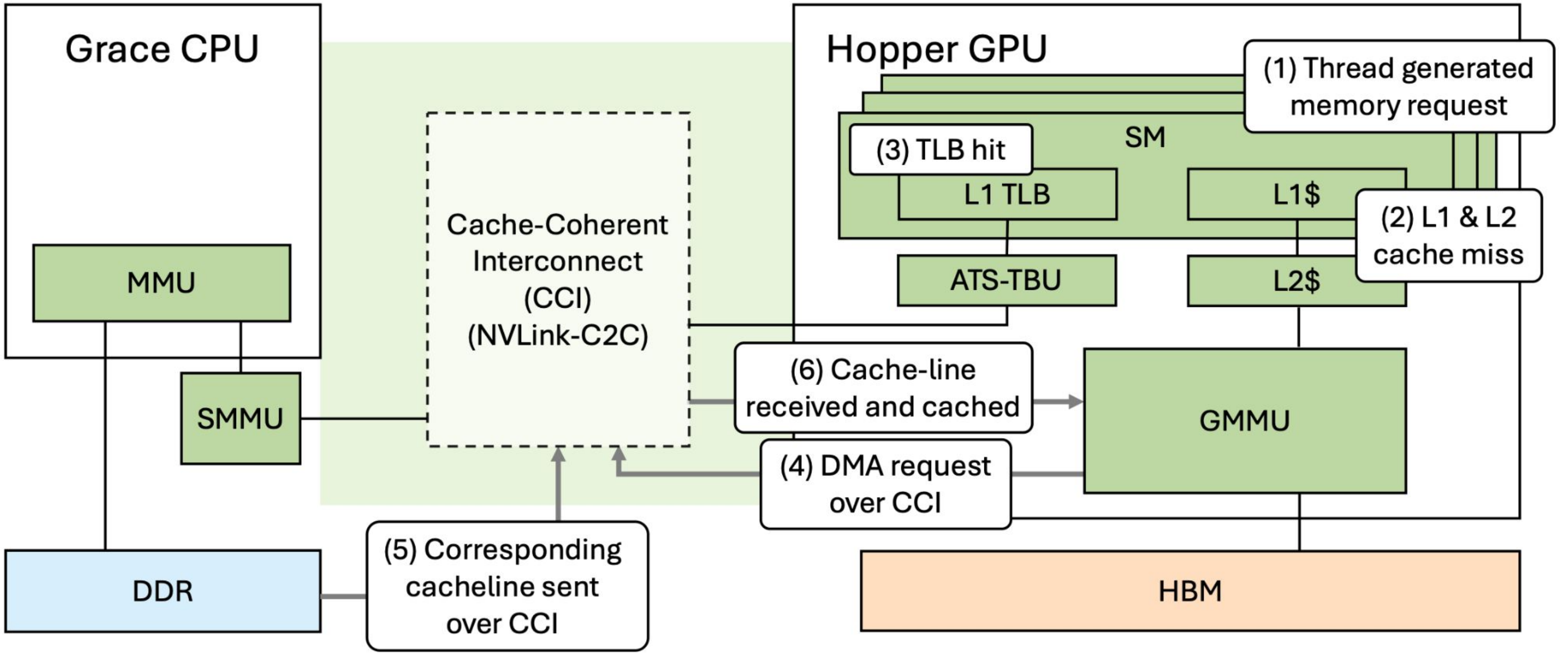


Architecture of Grace Hopper



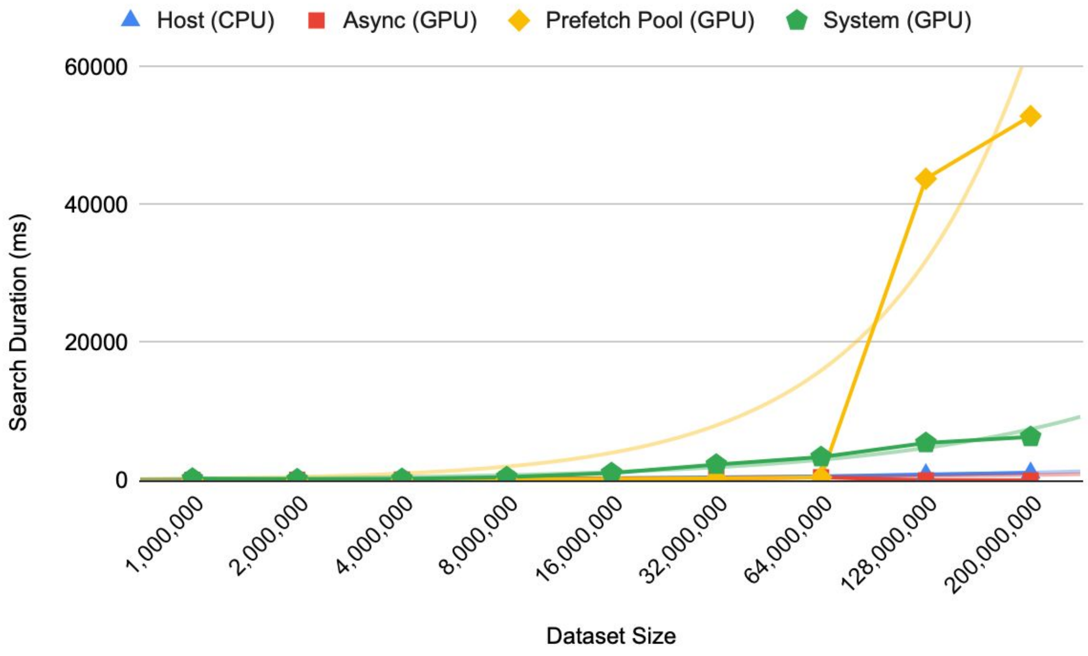
System memory management

System Memory in Grace Hopper

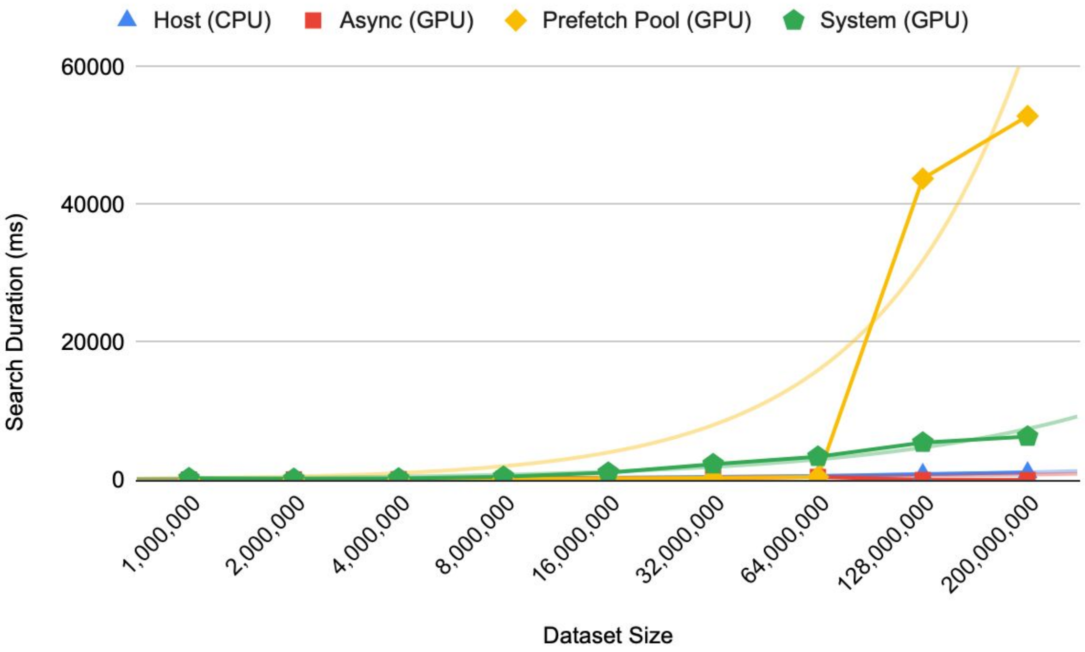


Source: https://arxiv.org/abs/2407.07850

System Memory >> UVM



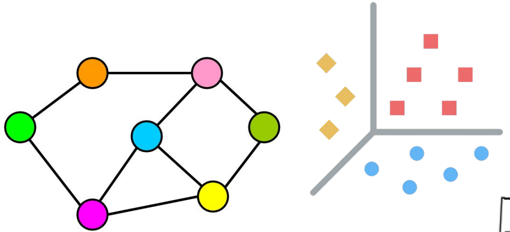
System Memory >> UVM



System memory in Grace Hopper scales much better than UVM :)

But system memory is slower in under-subscribed cases :(

A CPU/GPU Hybrid ANNS System



$$d_{L2}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

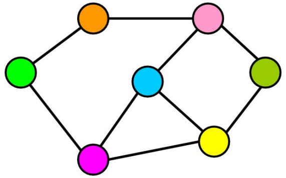
General Purpose
Cores /
Accelerators

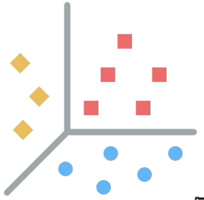
Use a Hybrid Index:
Graph (HNSW) +
Cluster (IVF)

$$d_{L2}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

CUDA Cores /
Tensor Cores

A CPU/GPU Hybrid ANNS System





$$d_{L2}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

General Purpose Cores / Accelerators

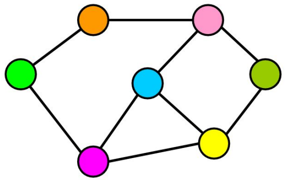
Store / traverse graph layers in CPU memory;

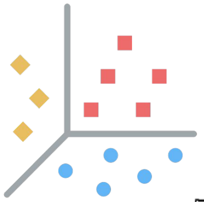
Store / process cluster layer in GPU memory

$$d_{L2}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

CUDA Cores / Tensor Cores

A CPU/GPU Hybrid ANNS System





$$d_{L2}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

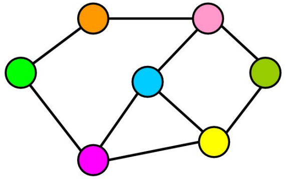
General Purpose Cores / Accelerators

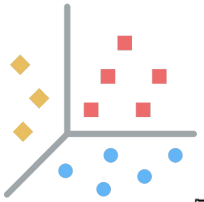
Perform heuristics based prefetching wherever possible

$$d_{L2}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

CUDA Cores / Tensor Cores

A CPU/GPU Hybrid ANNS System





$$d_{L2}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

$$d_{L2}(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

General Purpose Cores / Accelerators

Inter Query Parallelism:
GP cores, CUDA cores

Intra Query Parallelism:
Accelerators, Tensor Cores

CUDA Cores / Tensor Cores

Ongoing Work

Researching CPU/GPU collaborative vector search algorithms

Trying out heuristics based prefetching techniques to lessen the UM slowdown

Utilizing hardware accelerators on the CPU and GPU for accelerating distance calculation operations

Thank You

Questions?

jayjeetc@ucsc.edu

https://jayjeetc.github.io

Thank you to our sponsors!