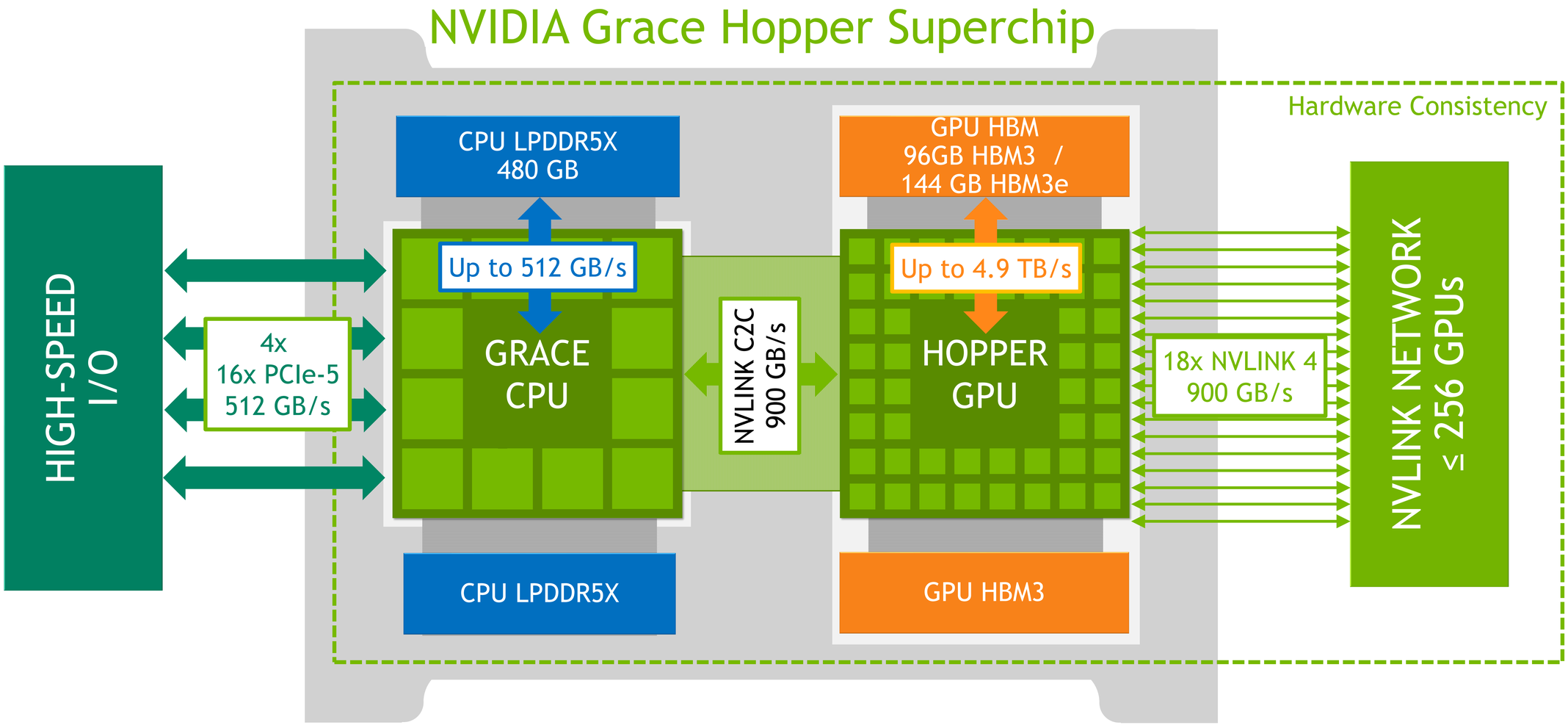
Grace Hopper Superchip Architecture and Performance Optimizations for AI Applications

Matthias Jouanneaux, Vishal Mehta, NVIDIA DevTech Compute |
GPU Technology Conference/March 20th, 2024

Grace Hopper Superchip

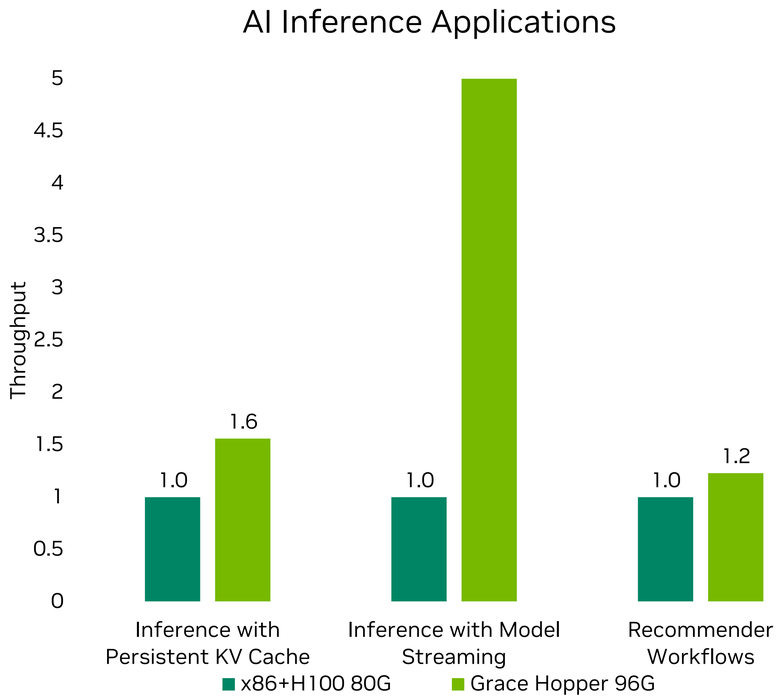
GPU can access CPU memory at CPU memory speeds

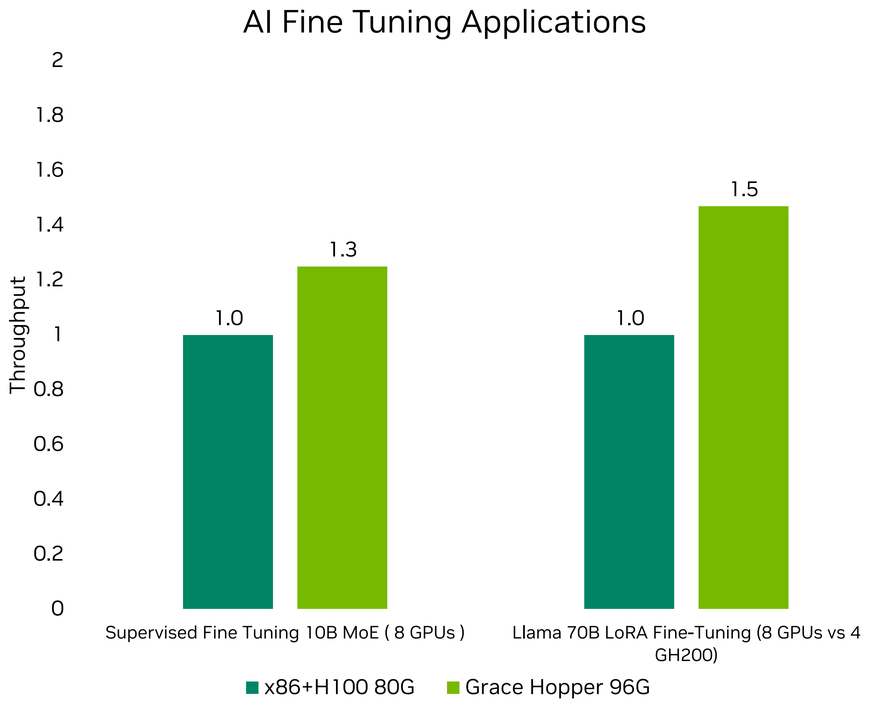


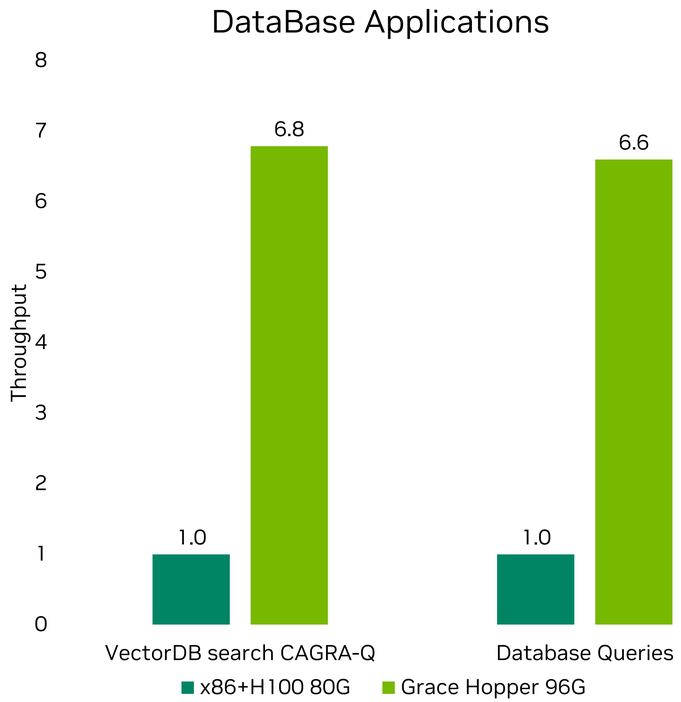
https://resources.nvidia.com/en-us-grace-cpu/nvidia-grace-hopper

Grace Hopper Performance sneak peek

Improved GPU utilization for AI applications







GH200: Grace CPU 480GB LPDDR5, H100 96GB HBM3
DGX H100 80GB HBM3

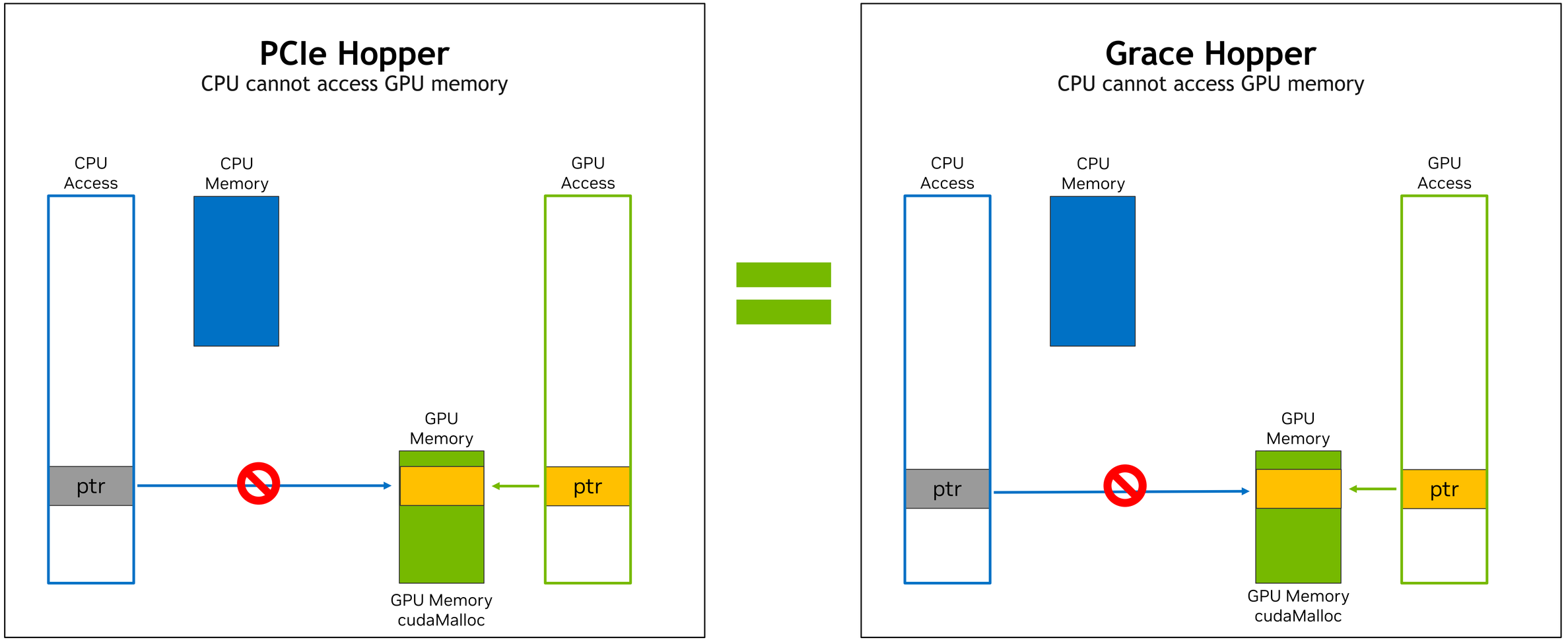
Agenda

- Grace Hopper CUDA Features
- Fine Tuning Applications
- LLM Inference
- Recommender Workflows
- Database Queries

Grace Hopper CUDA Features

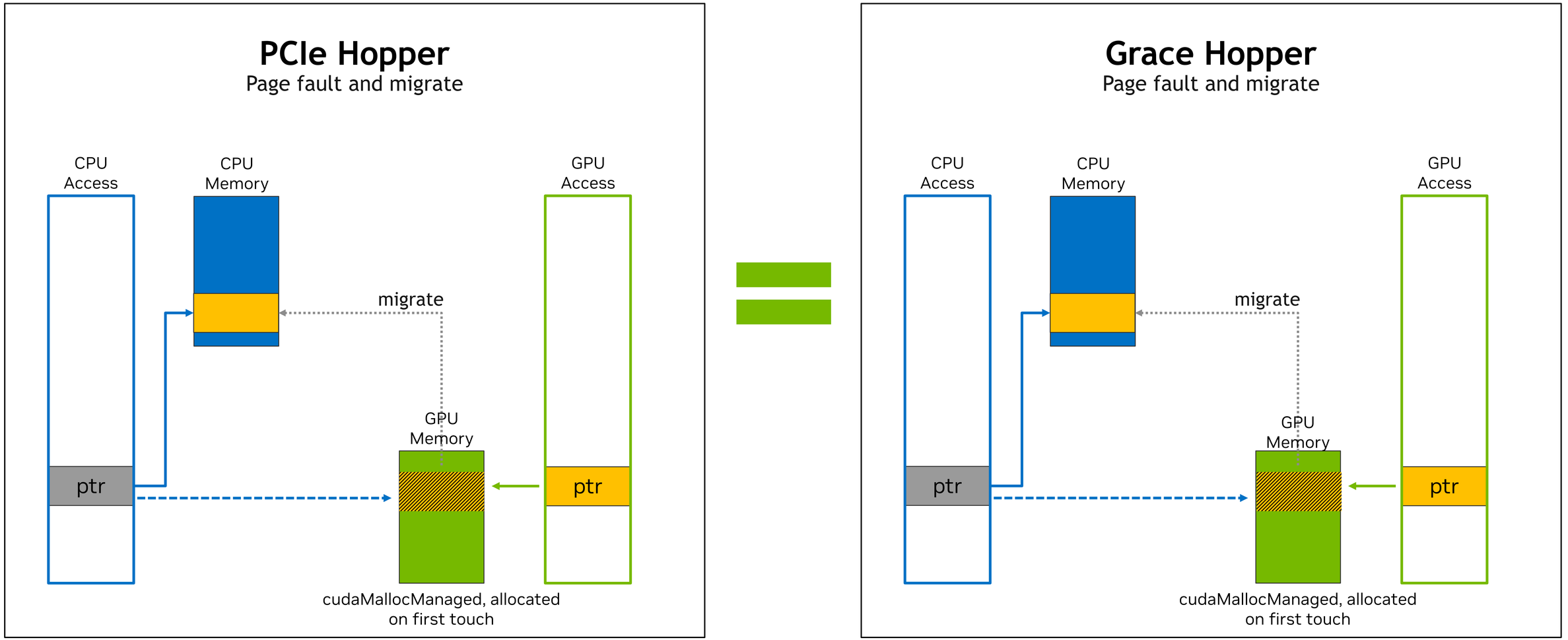
Memory Allocators

Device-only Memory Management (cudaMalloc)



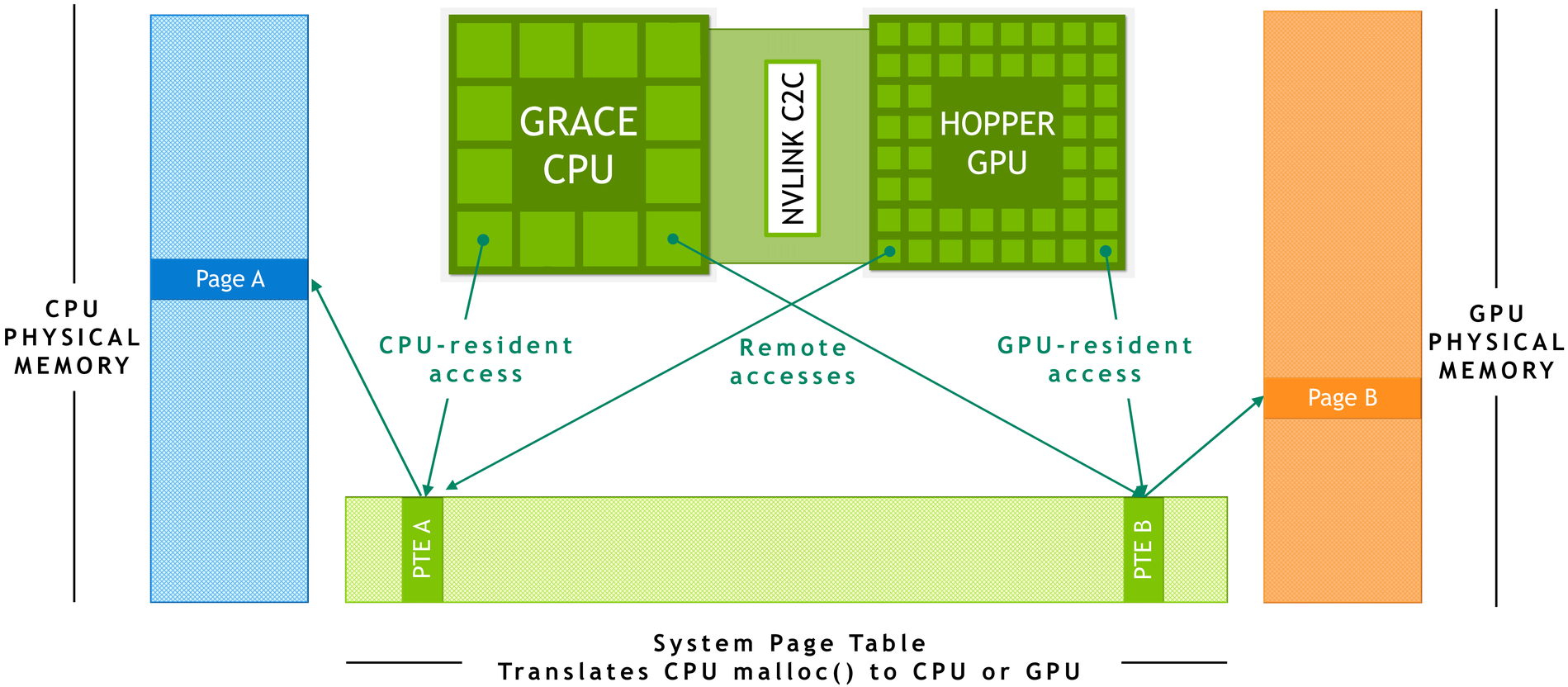
Memory Allocators

CUDA Managed Memory (cudaMallocManaged)



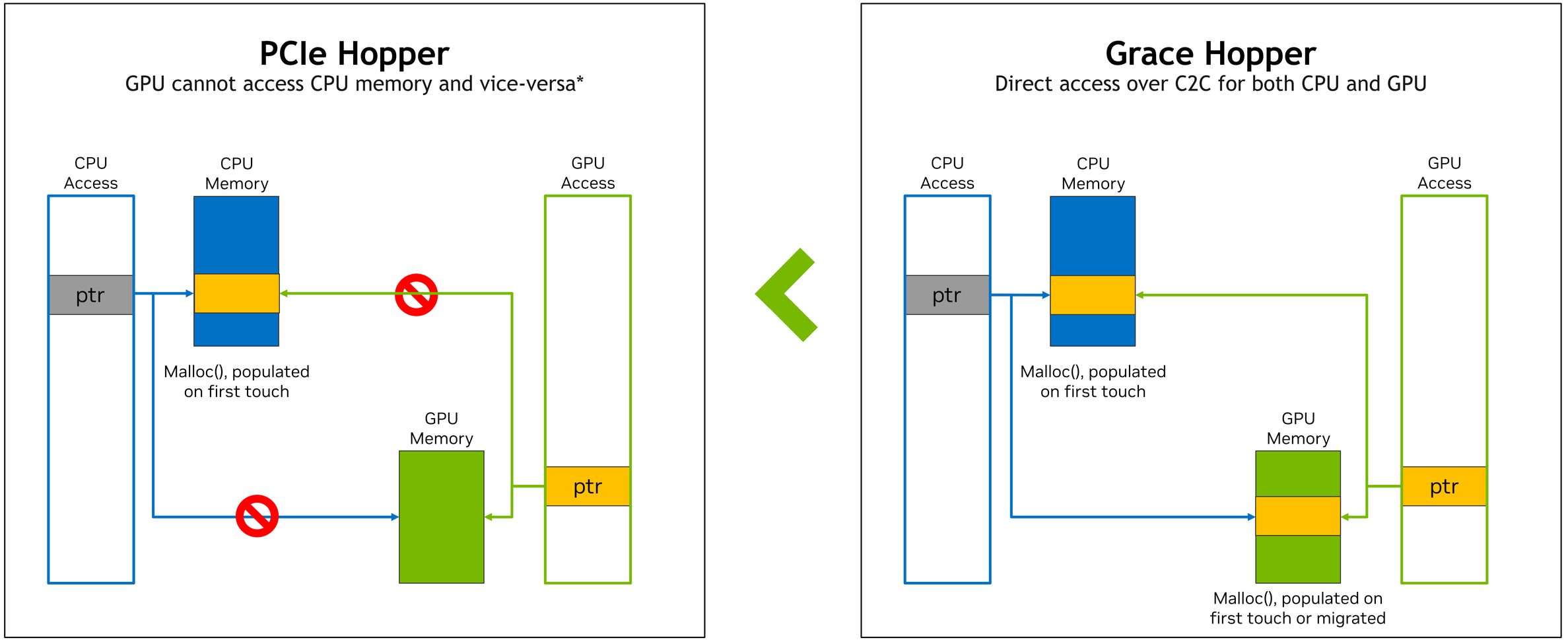
Grace Hopper Unified Memory

Address Translation Service (ATS)



Grace Hopper Memory Allocators

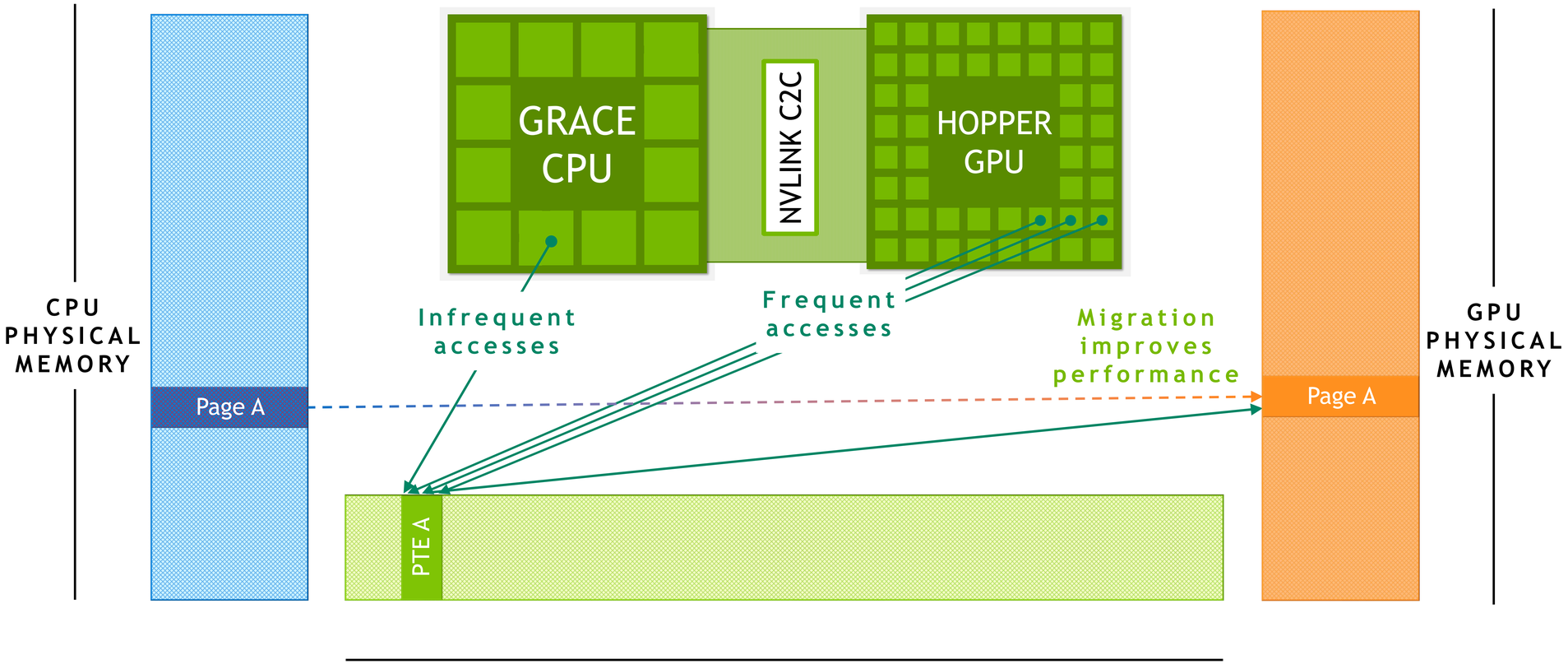
System Allocated Memory (malloc/mmap)



* Heterogeneous Memory Management (HMM) can emulate the behaviour using page faults and migration

Grace Hopper Optimizations

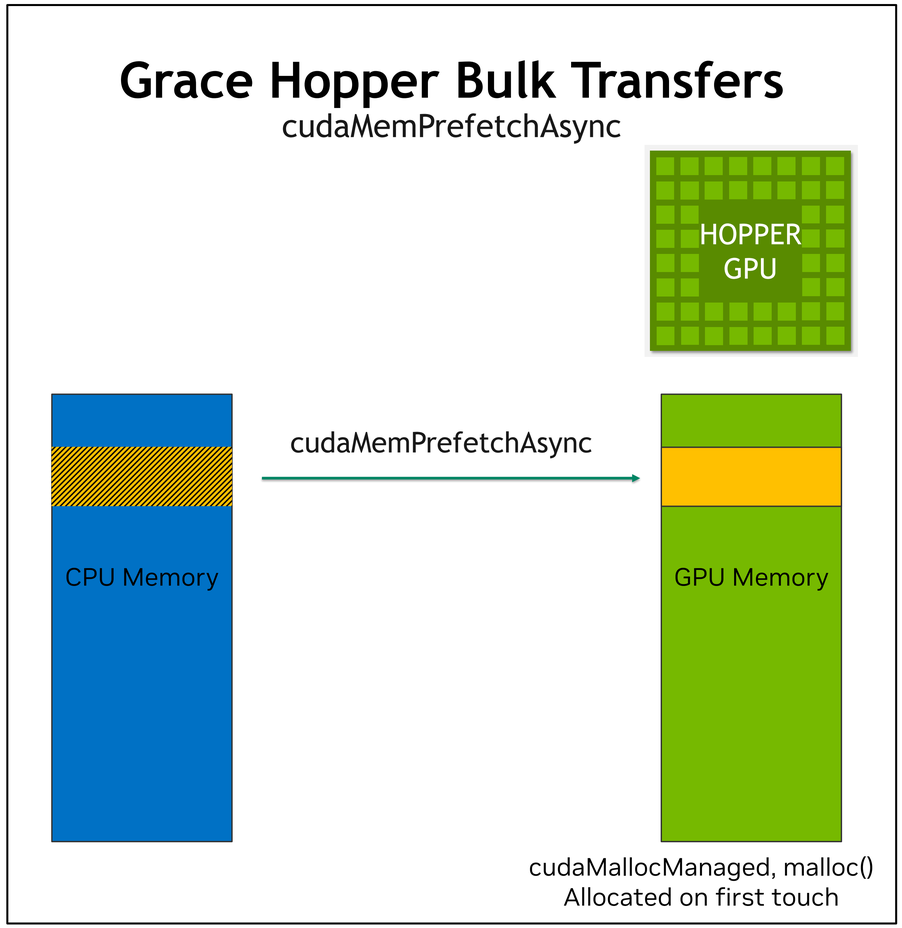
Automatic migrations for Unified Memory CPU -> GPU

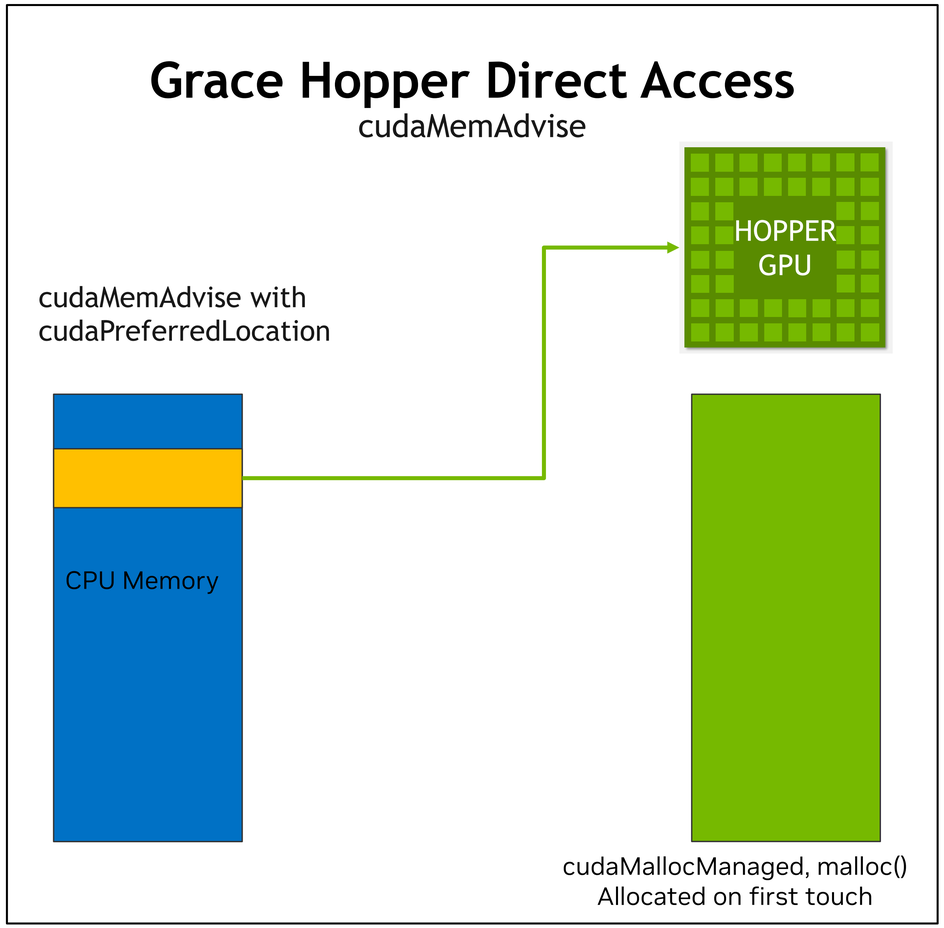


Grace Hopper Optimizations

CUDA memory tuning APIs

- On Grace Hopper CUDA memory tuning APIs can be used for both cudaMallocManaged and System Allocated Memory.

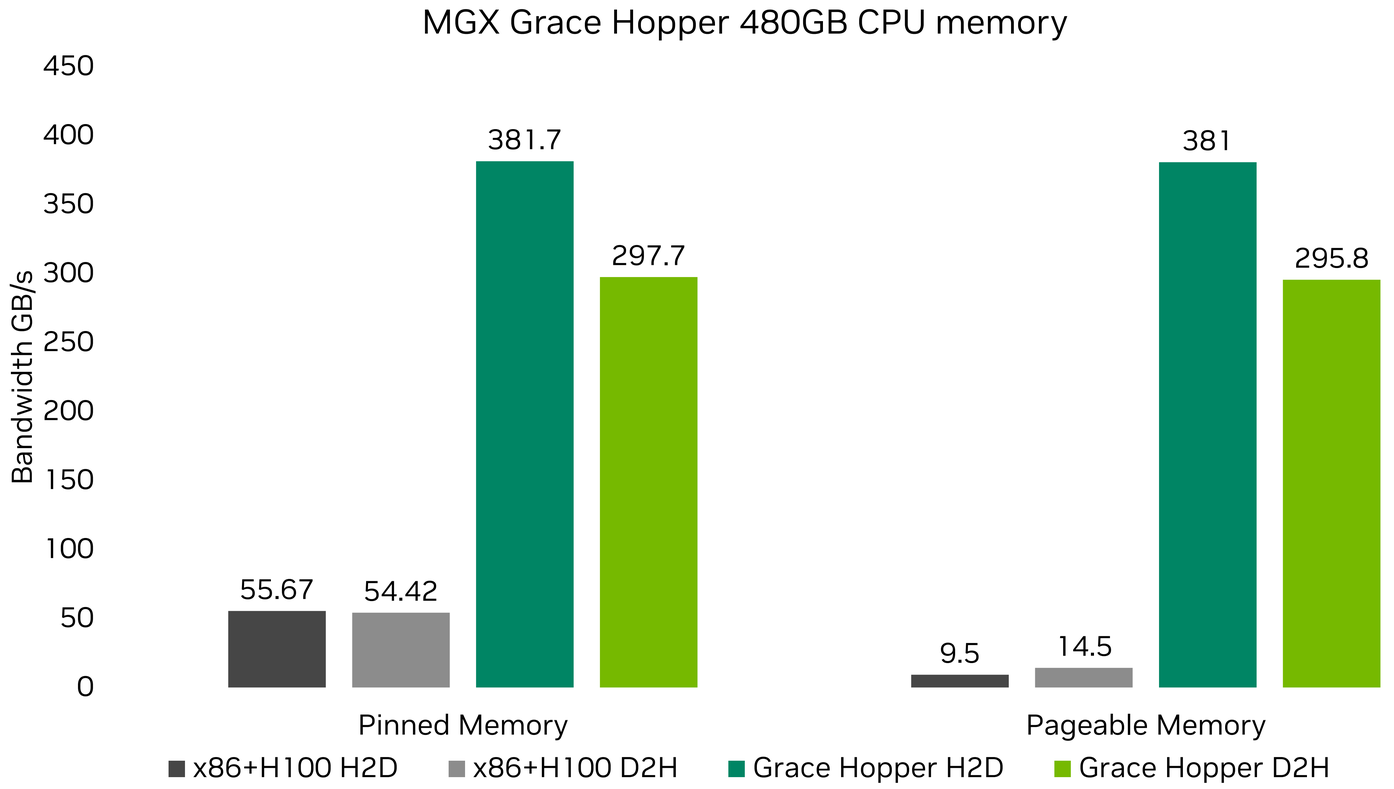




Grace Hopper Optimization

NVLink-C2C performance

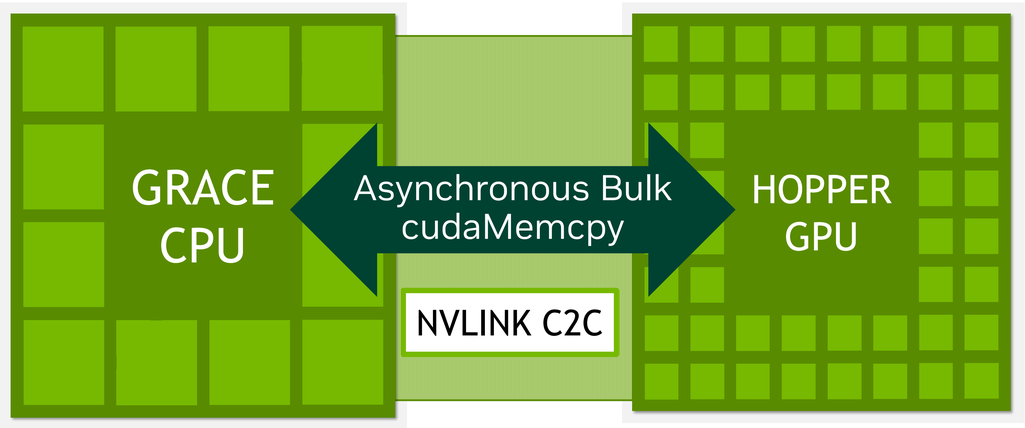
- cudaMemcpy Performance for Pageable and Pinned memory



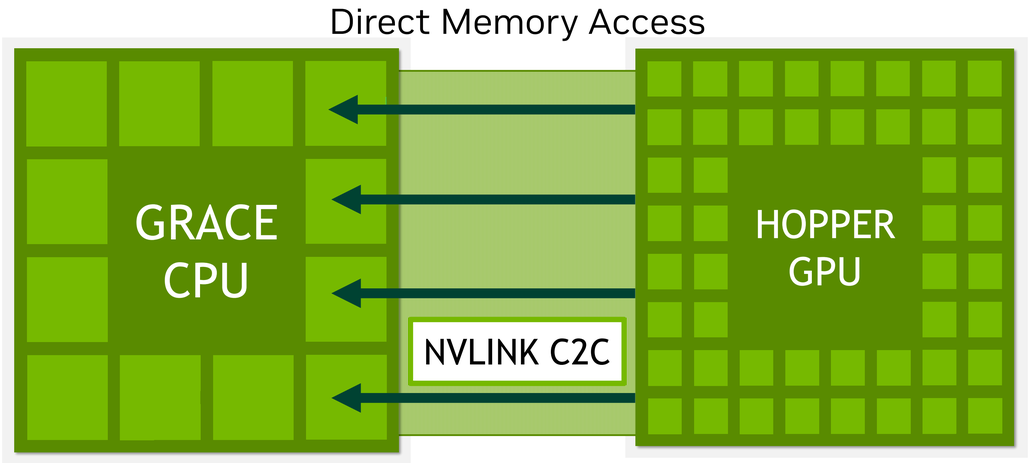
Optimizing applications on Grace Hopper

Grace Hopper Optimization

Key optimization techniques for AI applications



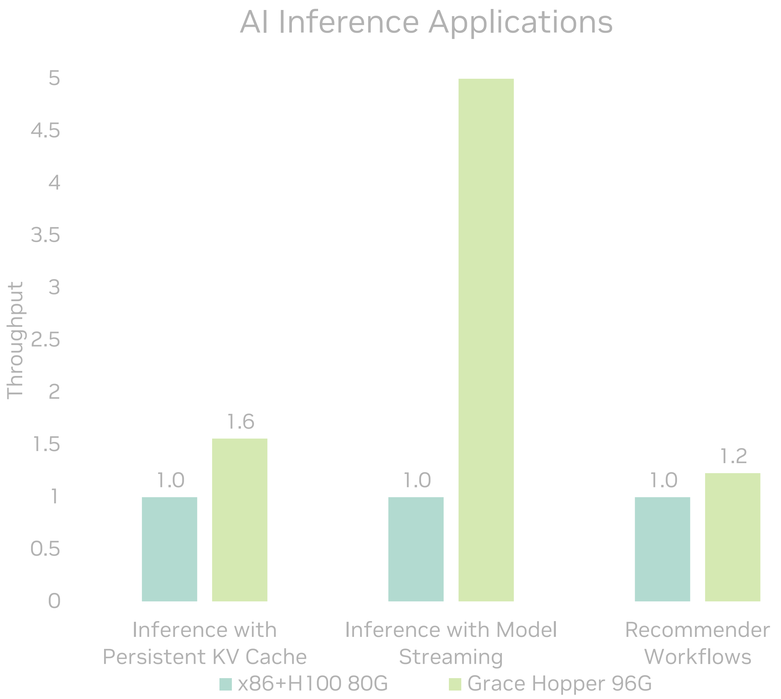
CPU memory as extension over NVLink C2C,
with asynchronous offloading capabilities
for activations and parameters

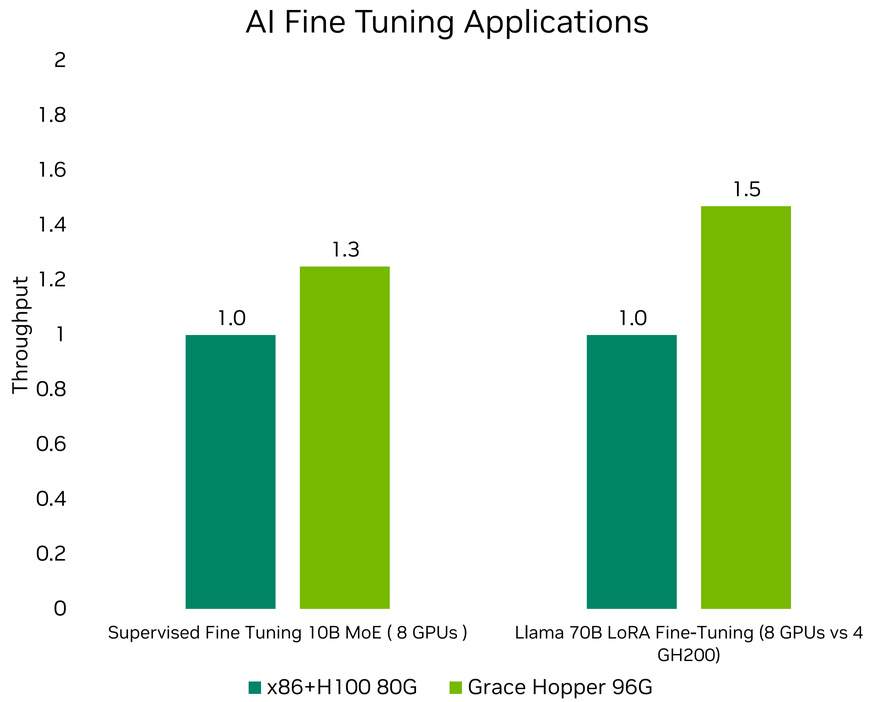


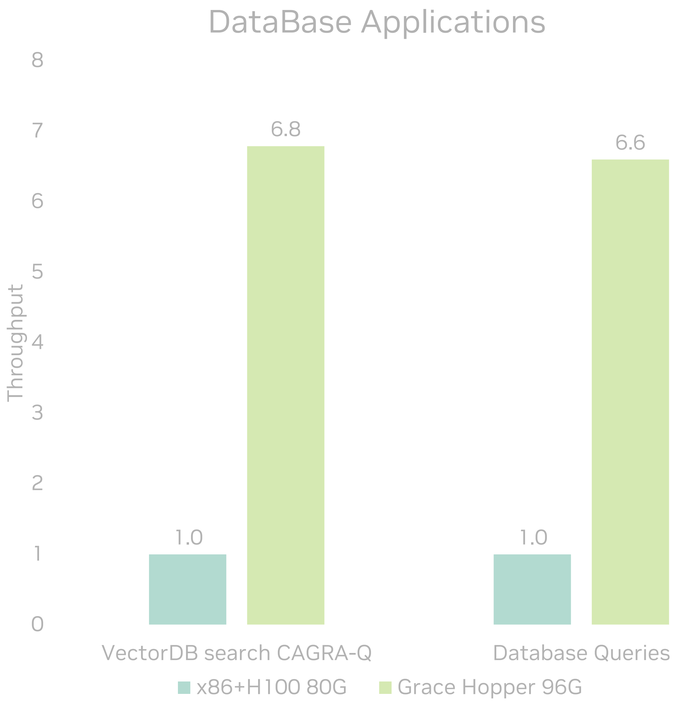
Explicit high bandwidth and capacity cache with
direct CPU memory access over NVLink C2C.

Grace Hopper Performance sneak peek

Improved GPU utilization for AI applications







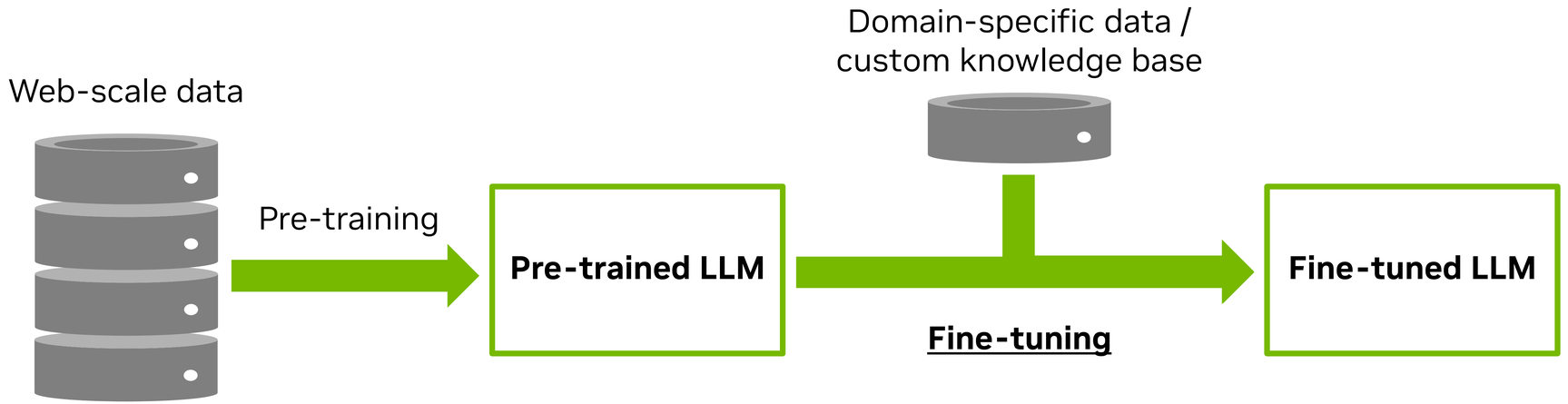
GH200: Grace CPU 480GB LPDDR5, H100 96GB HBM3
DGX H100 80GB HBM3

Supervised Fine Tuning (SFT)

Thanks to: Matthias Langer (DevTech Compute APAC)

Supervised Fine Tuning

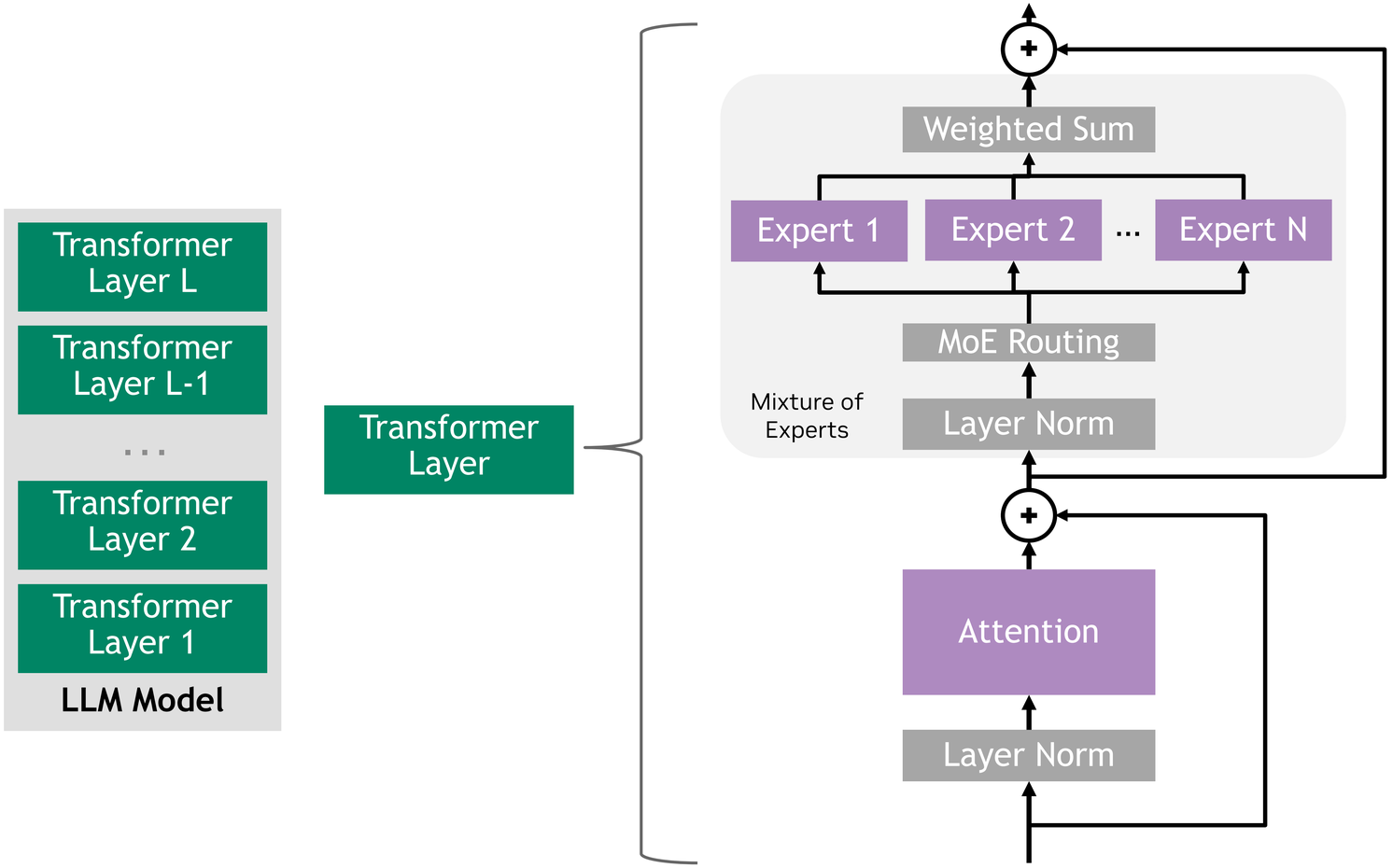
Fine Tuning using Grace Hopper Unified Memory, PyTorch and Megatron



Supervised Fine Tuning

Mixture of Experts (MoE)

- Each Transformer Layer is memory intensive due to multiple Feed Forward expert layers.
- Continual Pre-training involves augmenting trained models with new datasets.

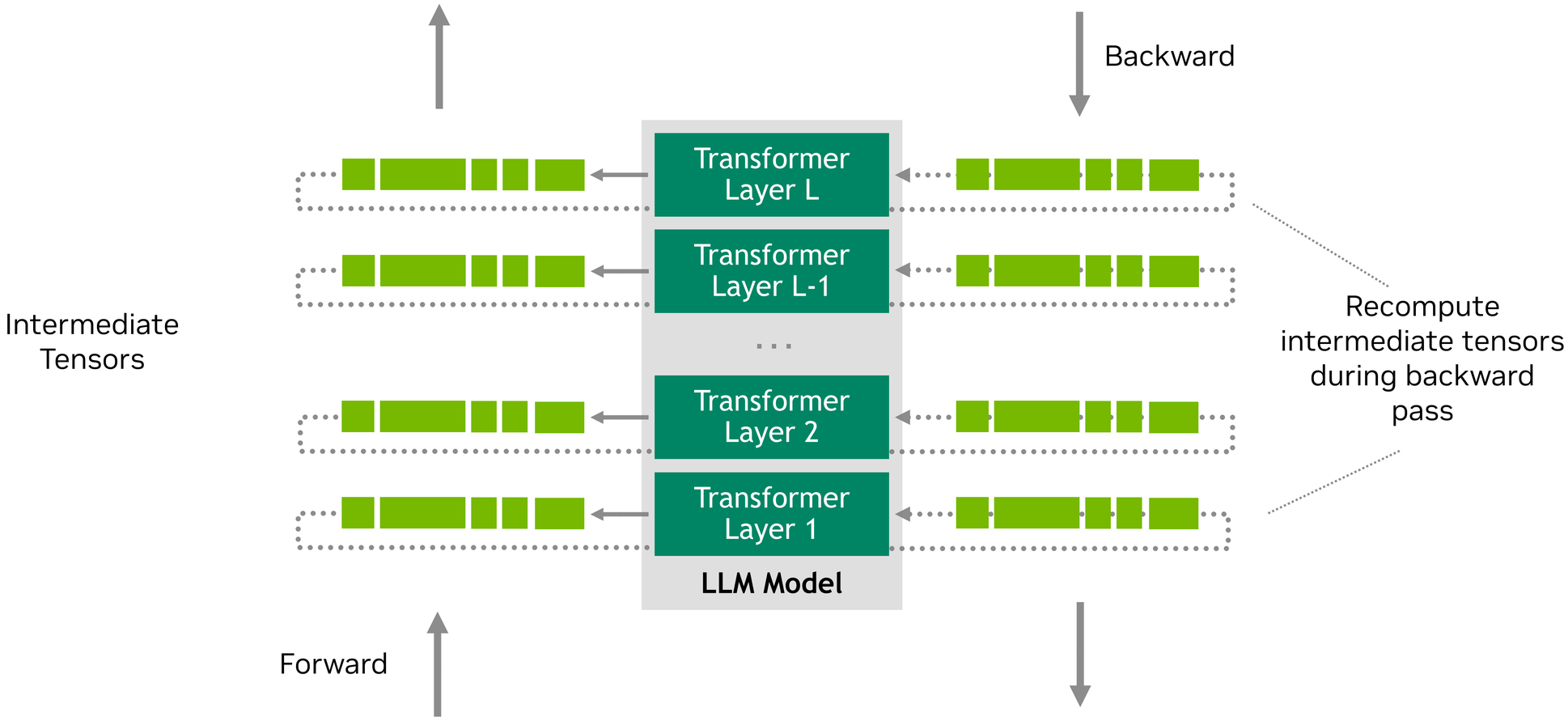


Supervised Fine Tuning

Recompute activations to utilize memory more efficiently

Technique: Activation checkpointing

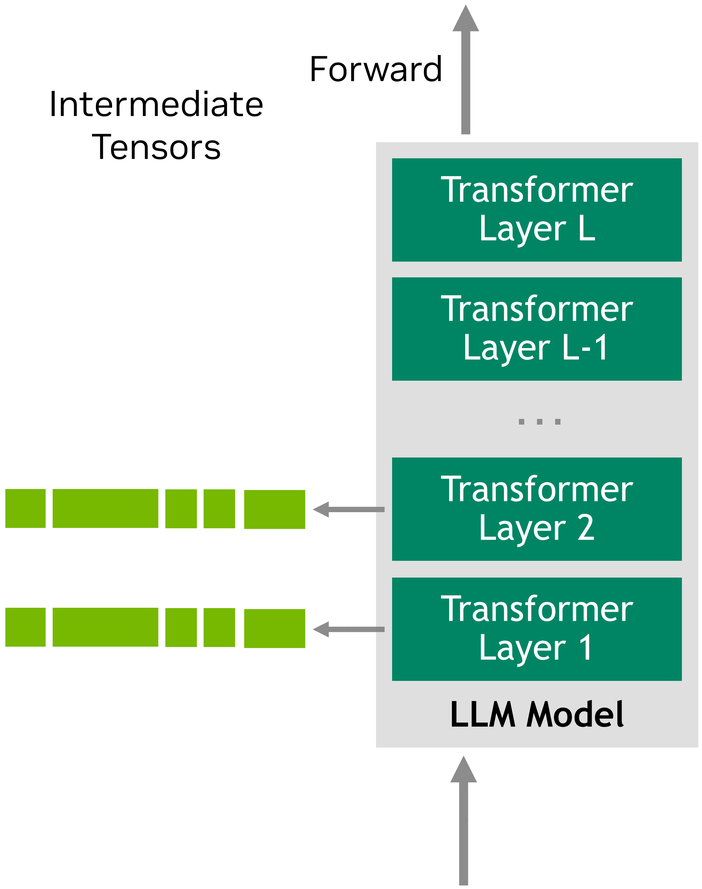
- Intermediate tensors have high memory requirements
- Re-computation has low memory Overhead but redundant computations and higher energy use.



Supervised Fine Tuning

Asynchronous offloading to CPU memory

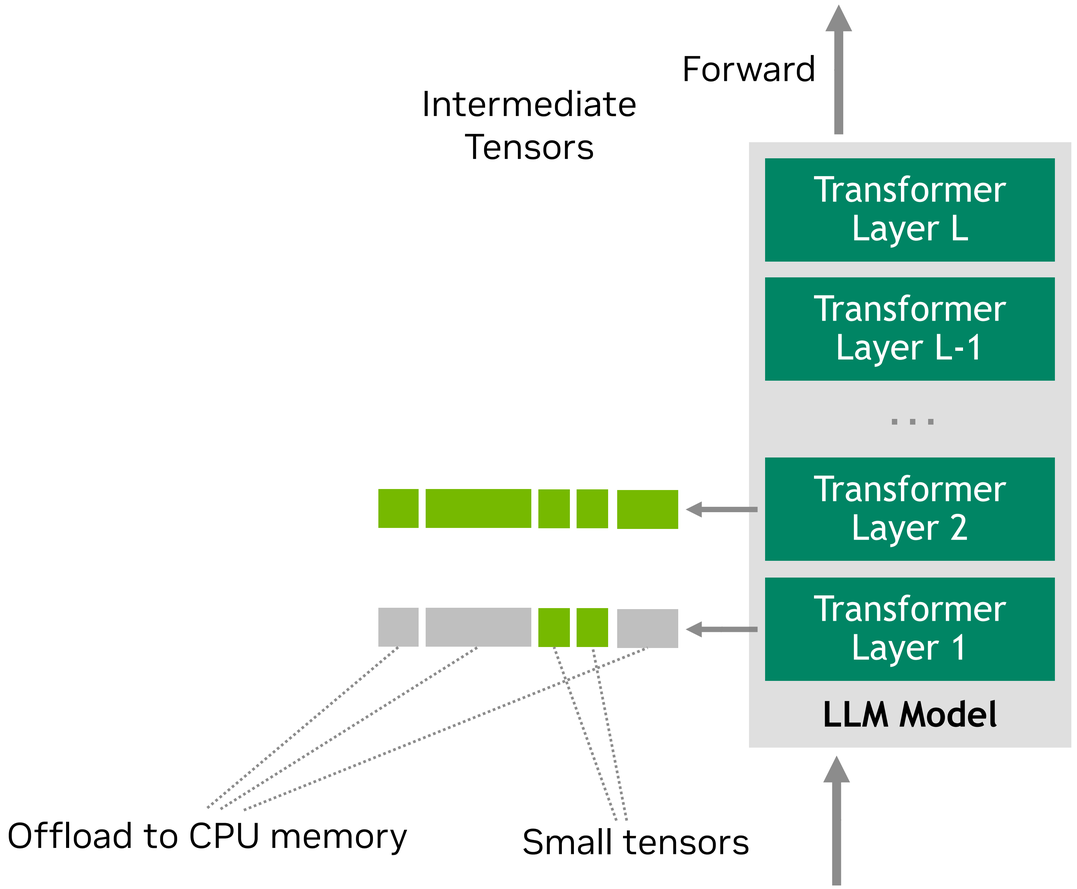
Technique: Asynchronous activation offloading



Supervised Fine Tuning

Asynchronous offloading to CPU memory

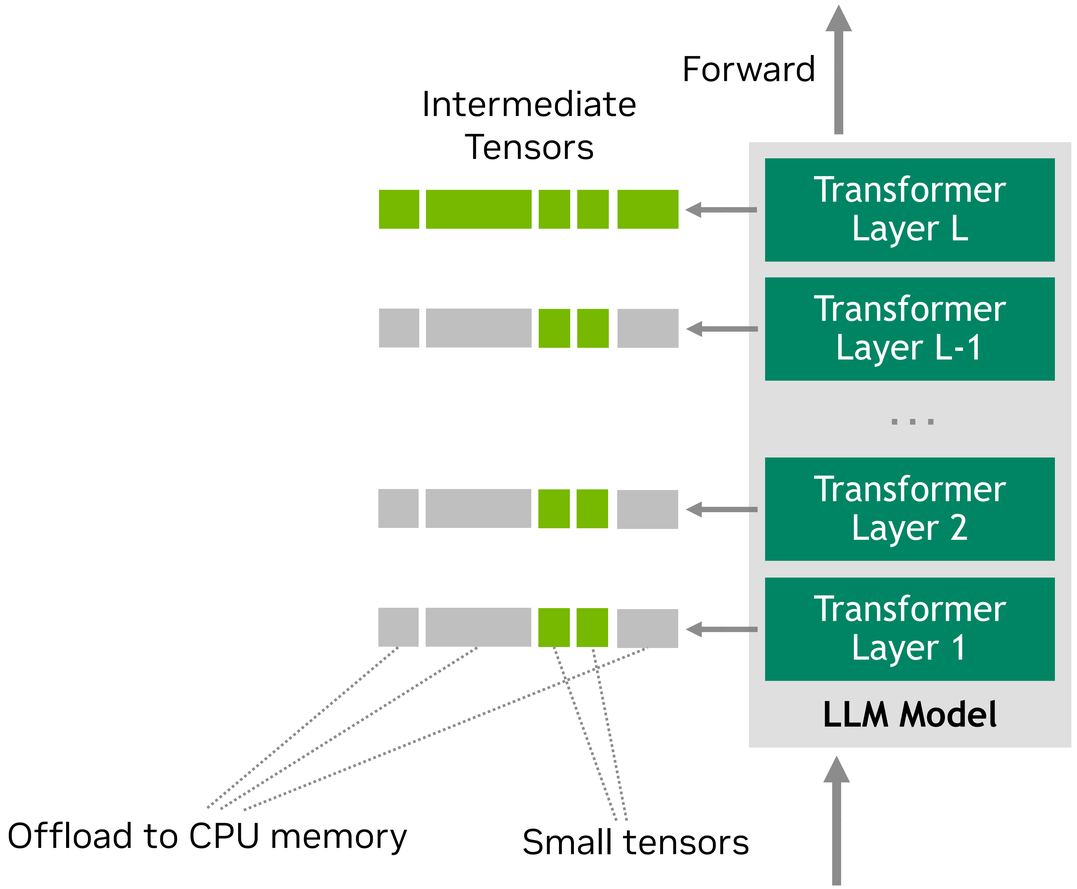
Technique: Asynchronous activation offloading



Supervised Fine Tuning

Asynchronous offloading to CPU memory

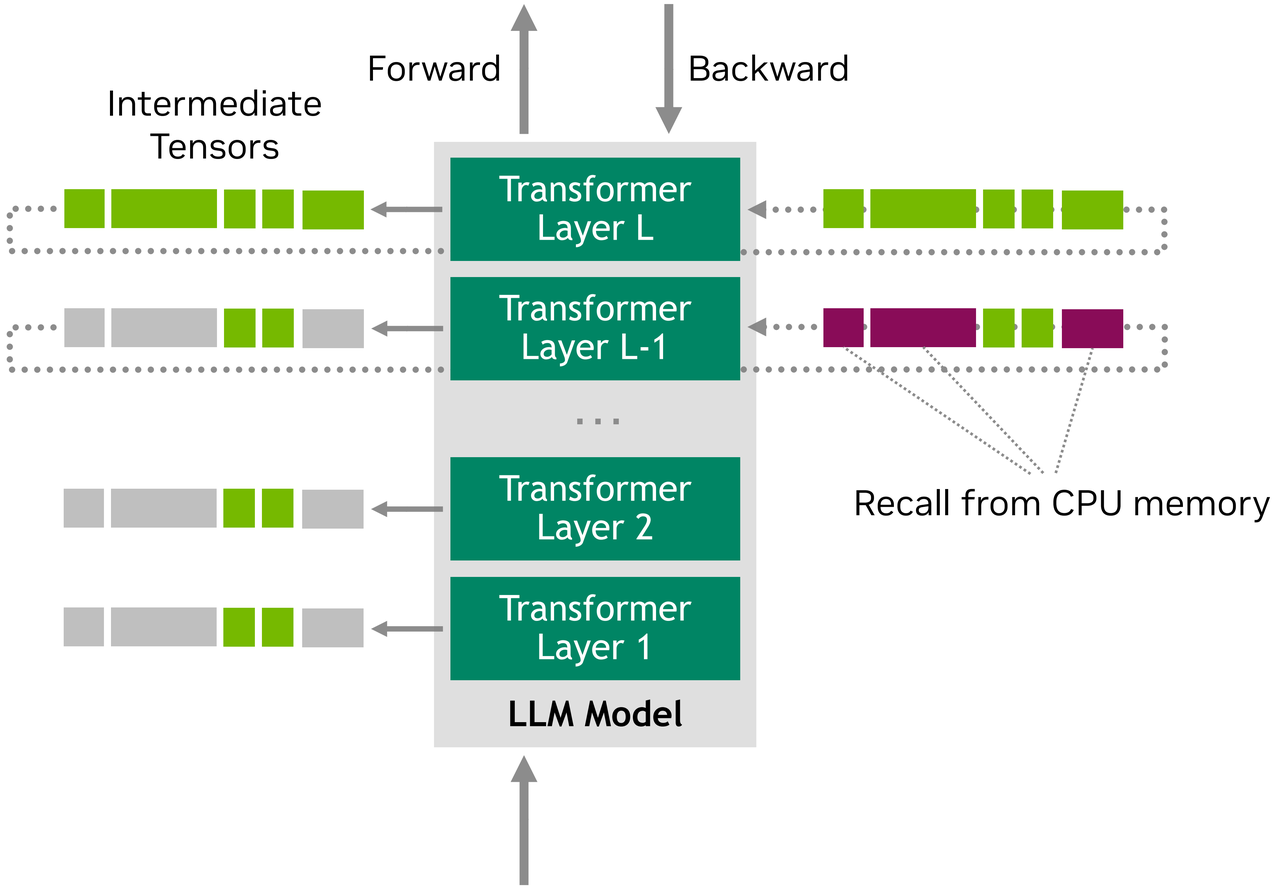
Technique: Asynchronous activation offloading



Supervised Fine Tuning

Asynchronous offloading to CPU memory

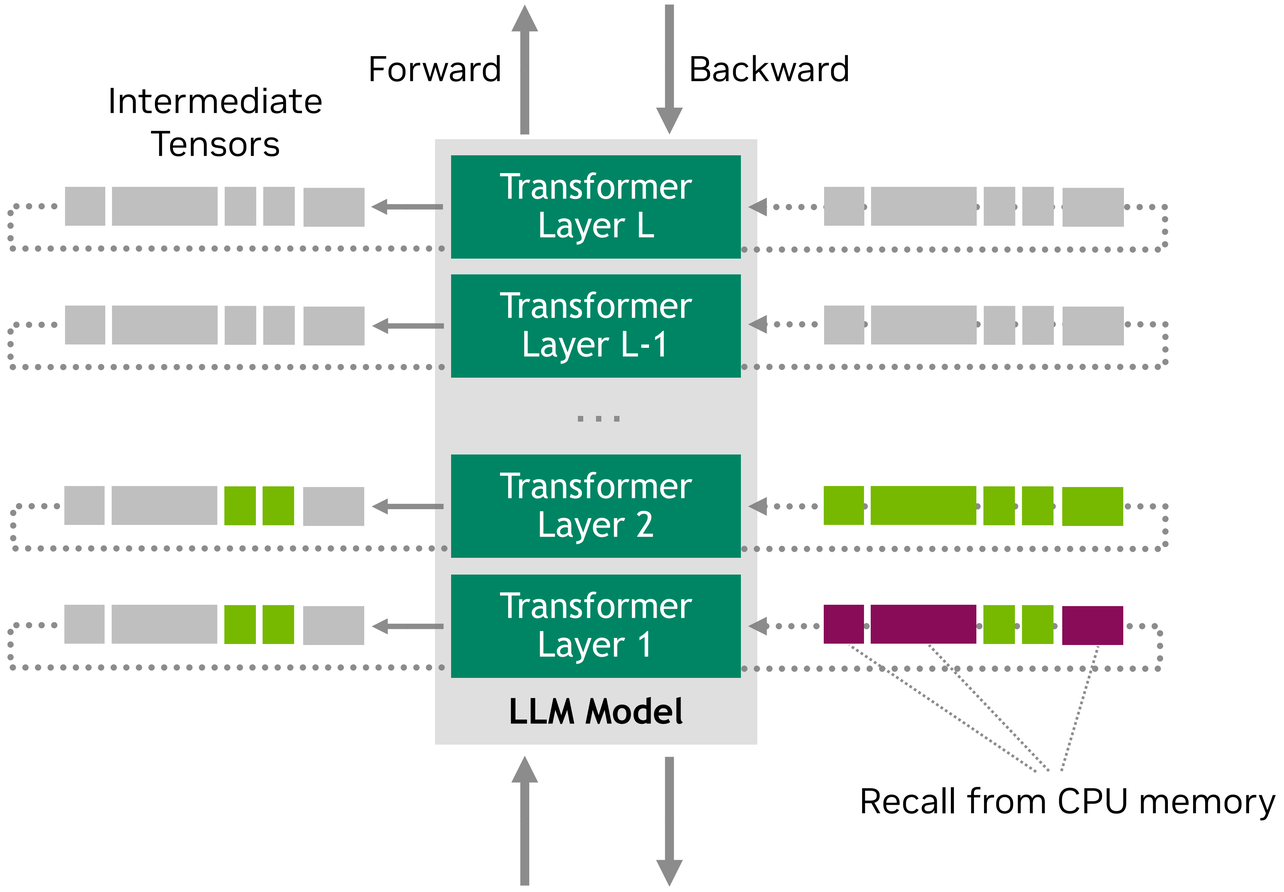
Technique: Asynchronous activation offloading



Supervised Fine Tuning

Asynchronous offloading to CPU memory

Technique: Asynchronous activation offloading



Supervised Fine Tuning

Asynchronous offloading to CPU memory in Megatron LM

- Implementation in Megatron LM will expose runtime parameters for offloading optimizations.

```
--num-experts 2                    …
--num-layers 15
--hidden-size 7680                 --cpu-offload-activations
--ffn-hidden-size 15360            --cpu-offload-asynchronous
--num-attention-heads 120          --cpu-offload-size-threshold 67108864
--seq-length 8192
--max-position-embeddings 8192
--bf16
--use-flash-attn
```

```
self.cpu_offload_context = CPUOffloadContext(args)

...

with self.cpu_offload_context as ctx:
   for index in range(self.num_layers):
      layer = self._get_layer(index)
      hidden_states = layer(hidden_states, **forward_kwargs)
      hidden_states = ctx(hidden_states, self.num_layers)
```

Approx 10B parameter MoE model MegatronLM configuration
for 8 H100 GPU systems

Size per tensor	# tensors	∑ size	% total
< 100 MB	270	0.2 GB	0.1%
100 ~ 499 MB	195	51.7 GB	20.6%
500 ~ 999 MB	60	33.3 GB	13.2%
1000 ~ 1999 MB	135	135.9 GB	54.1%
> 2000 MB	15	30.2 GB	12.0%
TOTAL	**675**	**251.2 GB**	

Temporary tensors in approx. 10B parameter MoE model
(2 experts,15 layers, batch size = 8)

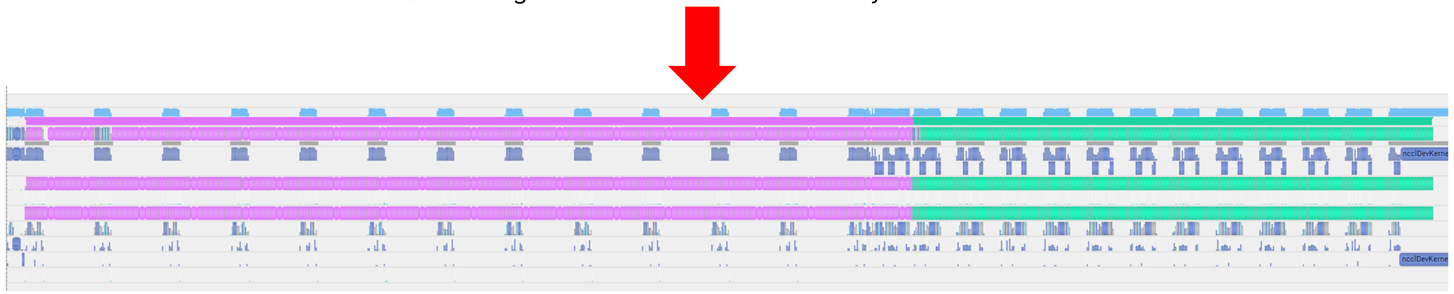
Supervised Fine Tuning

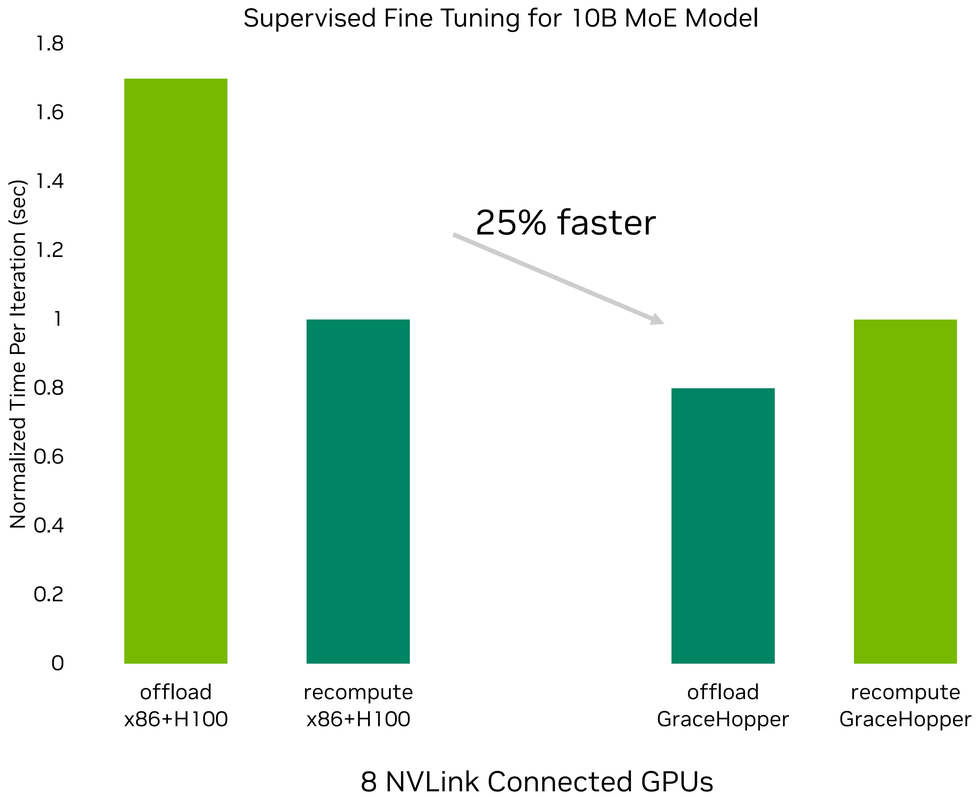
Profiling and Performance

- Activation offloading is more efficient on Grace Hopper compared to x86 platforms.

Offloading on Grace Hopper is hidden behind compute

Offloading on x86+H100 is dominated by data transfer



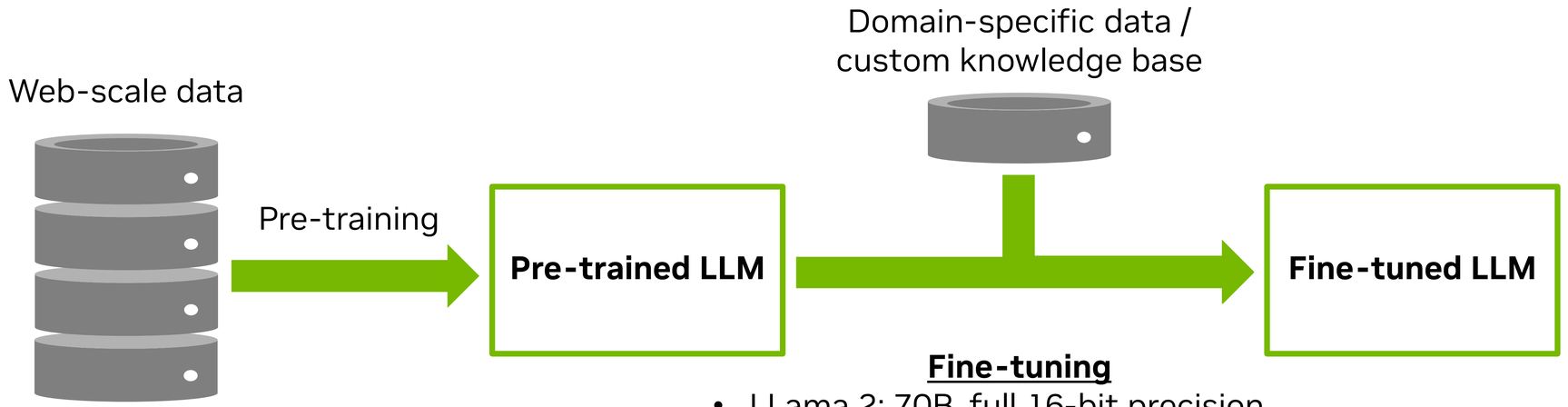


Compute D2H H2D

LoRA Fine Tuning

LoRA Fine Tuning Llama 2 70B

Fine Tuning using Grace Hopper Unified Memory, PyTorch and Huggingface



Fine-tuning

- LLama 2: 70B, full 16-bit precision
  - 140GB parameters only
- PEFT low-rank adaptation: LoRA
  - Reduce trainable parameters
  - Reduce optimizer memory footprint
- SAMSum dataset:
  - messenger-like conversations

LoRA Fine Tuning Llama 2 70B

Fine Tuning using Grace Hopper Unified Memory, PyTorch and Huggingface

- Any given transformer layer only uses fraction of overall memory (80 layers : ~1.25% of memory per layer)
- Enabling single-GPU fine-tuning requires both parameter and activation offloading

16-bit parameter + optimizer state	Size in GB
1 layer w/ LoRA	1.8
1 layer w/o LoRA	7
Full model w/ LoRA	**140**
Full model w/o LoRA	**560**

16-bit, batch size 1, context length 4096 activations	Size in GB
1 layer w/ checkpointing	0.07
1 layer w/o checkpointing	1.4
Full model w/ checkpointing	**6.7**
Full model w/o checkpointing	**112**

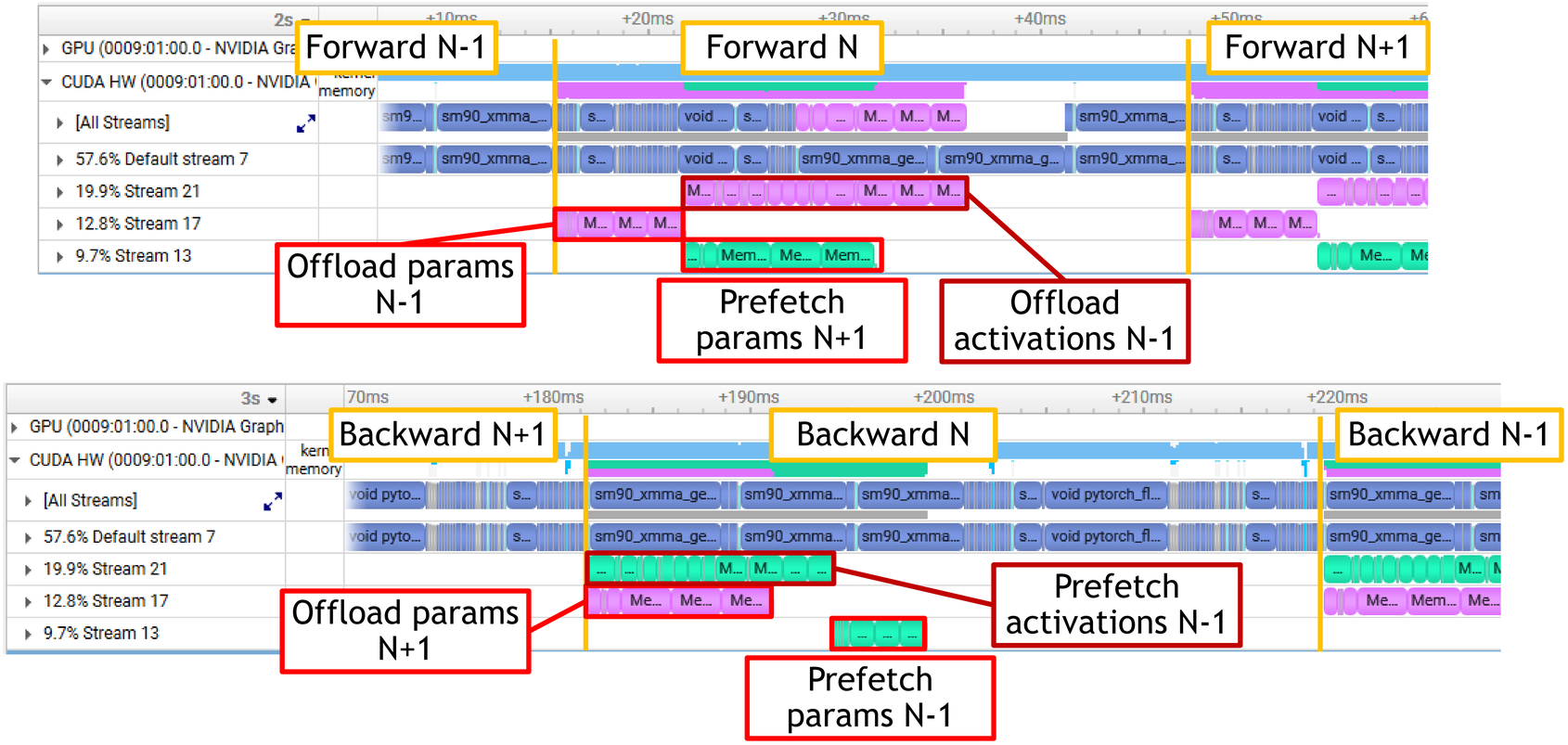
LoRA Fine Tuning Llama 2 70B

Fine Tuning using Grace Hopper Unified Memory, PyTorch and Huggingface

Technique: Asynchronous activation offloading

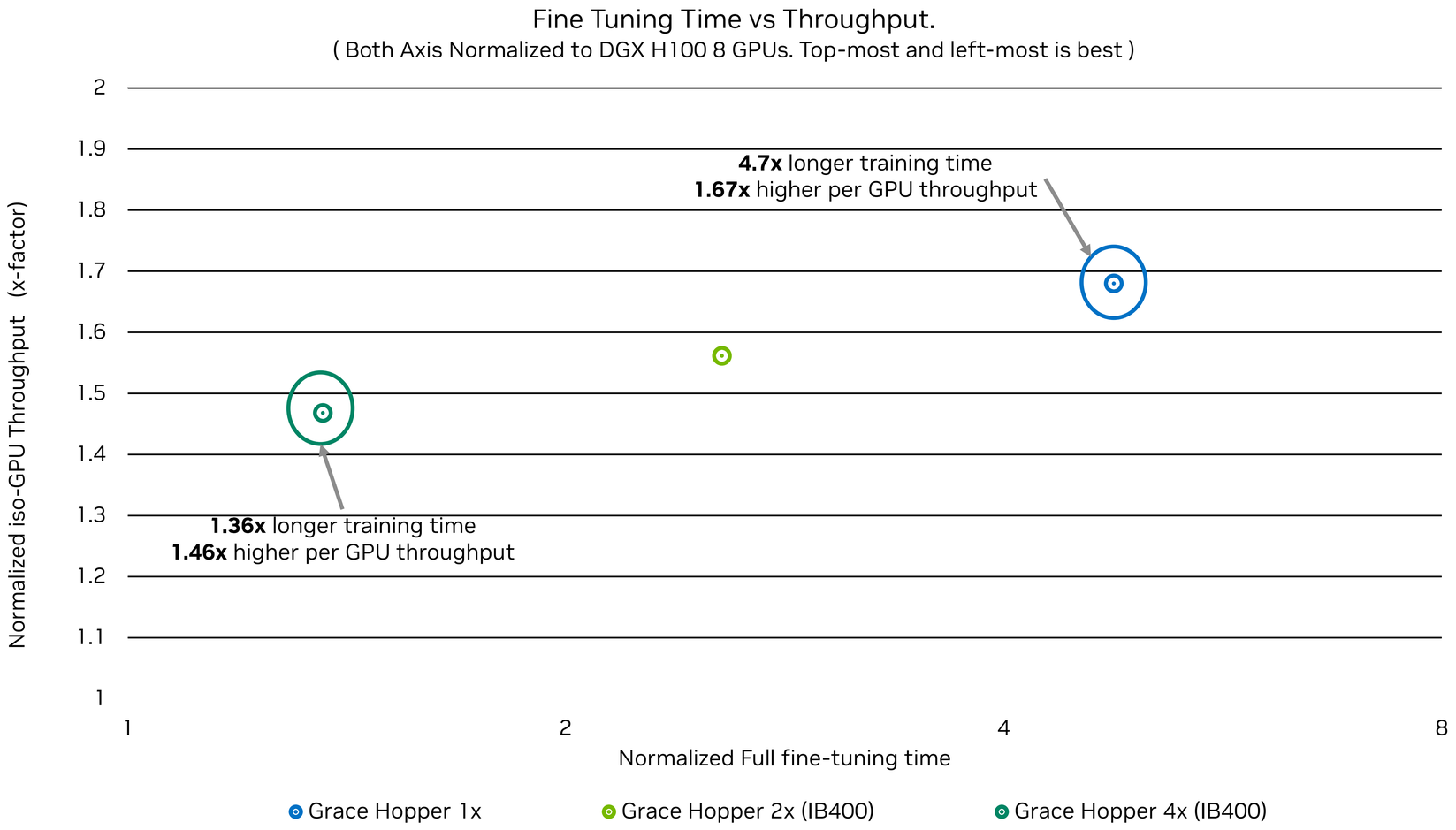
Technique: Asynchronous parameter offloading

- Main strategy to use fast & large CPU memory:
  - Use PyTorch forward/backward hooks for parameter offloading
  - Use PyTorch pack/unpack activation hooks for activation prefetching



LoRA Fine Tuning Llama 2 70B

Fine Tuning using Grace Hopper Unified Memory, PyTorch and Huggingface

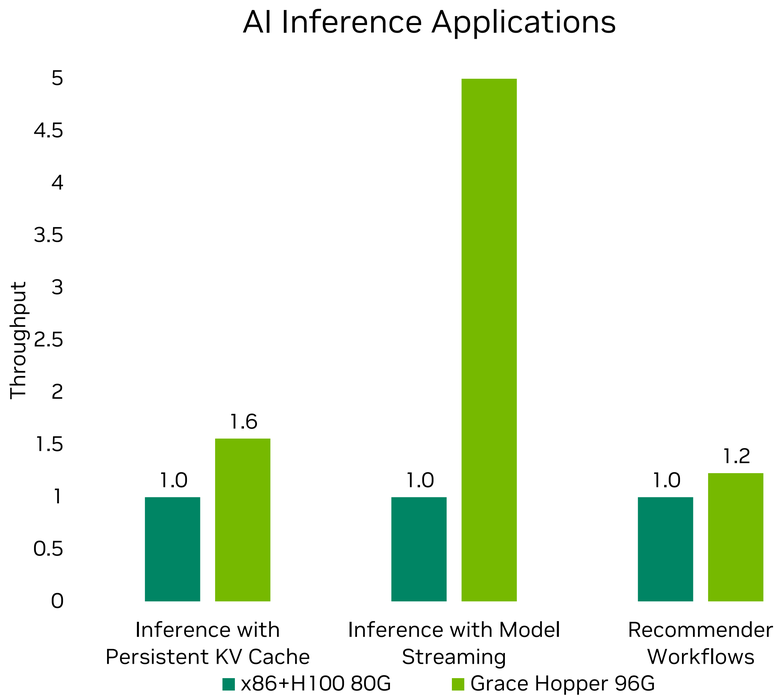


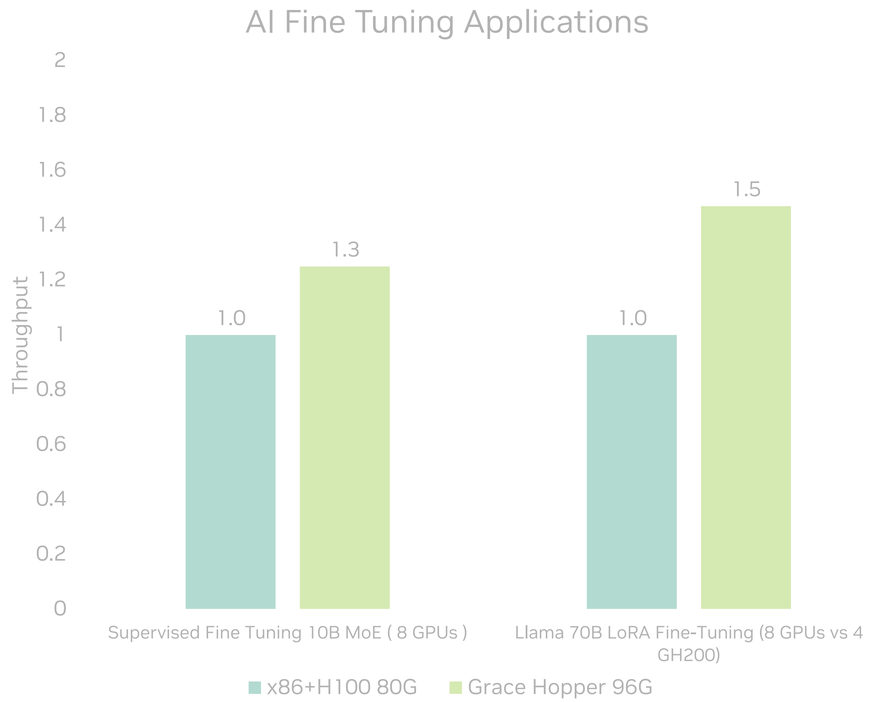
Experiments use 7 epochs, each with 368 batches of (global) size 40
Code: https://github.com/MatthiasKohl/llama-recipes/tree/main

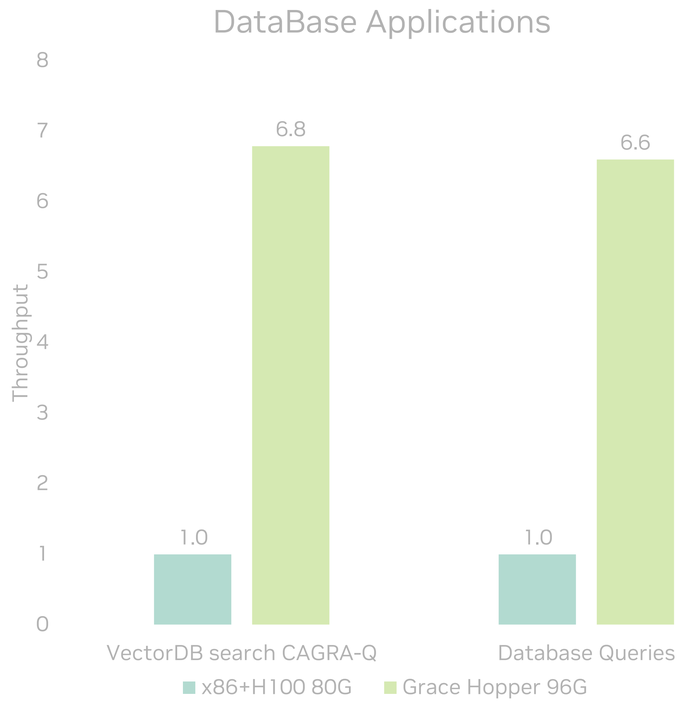
NVIDIA

Grace Hopper Performance sneak peek

Improved GPU utilization for AI applications







GH200: Grace CPU 480GB LPDDR5, H100 96GB HBM3
DGX H100 80GB HBM3

LLM Inference: Persistent KV Cache

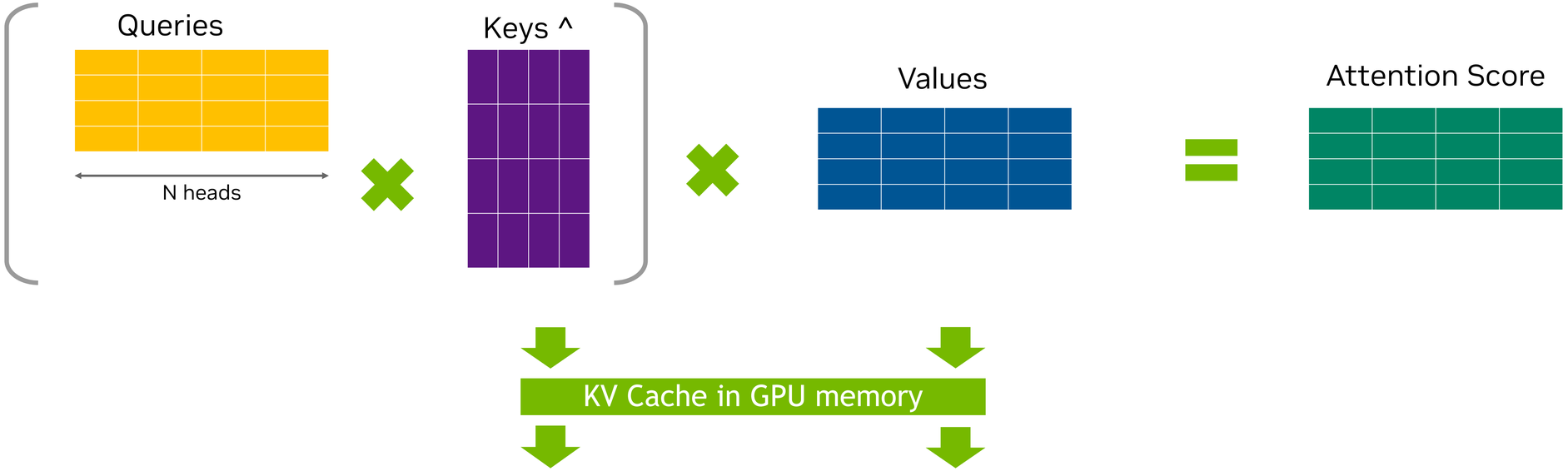
Thanks to: Thor Johnsen (TRT-LLM)

LLM Inference

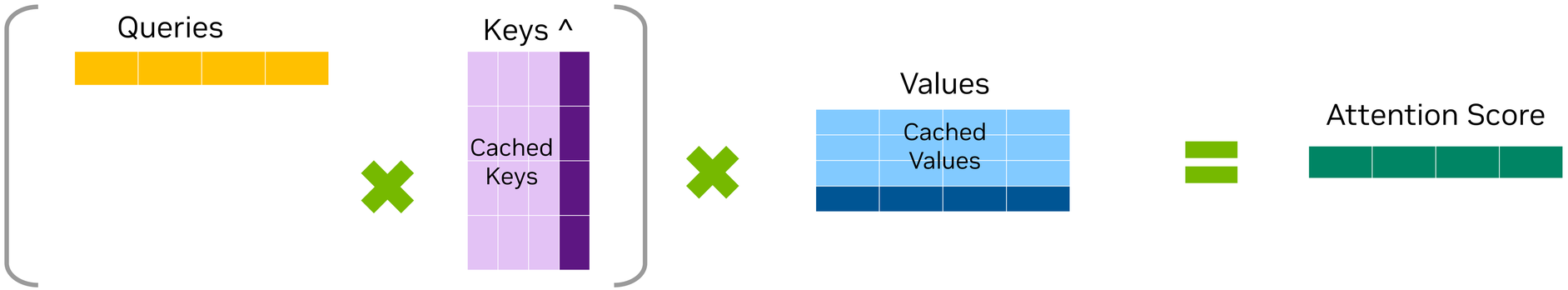
Attention computation

- Inferencing with KV (key-value) cache allows reuse of token computations when generating new tokens.

Step 1: Context Phase



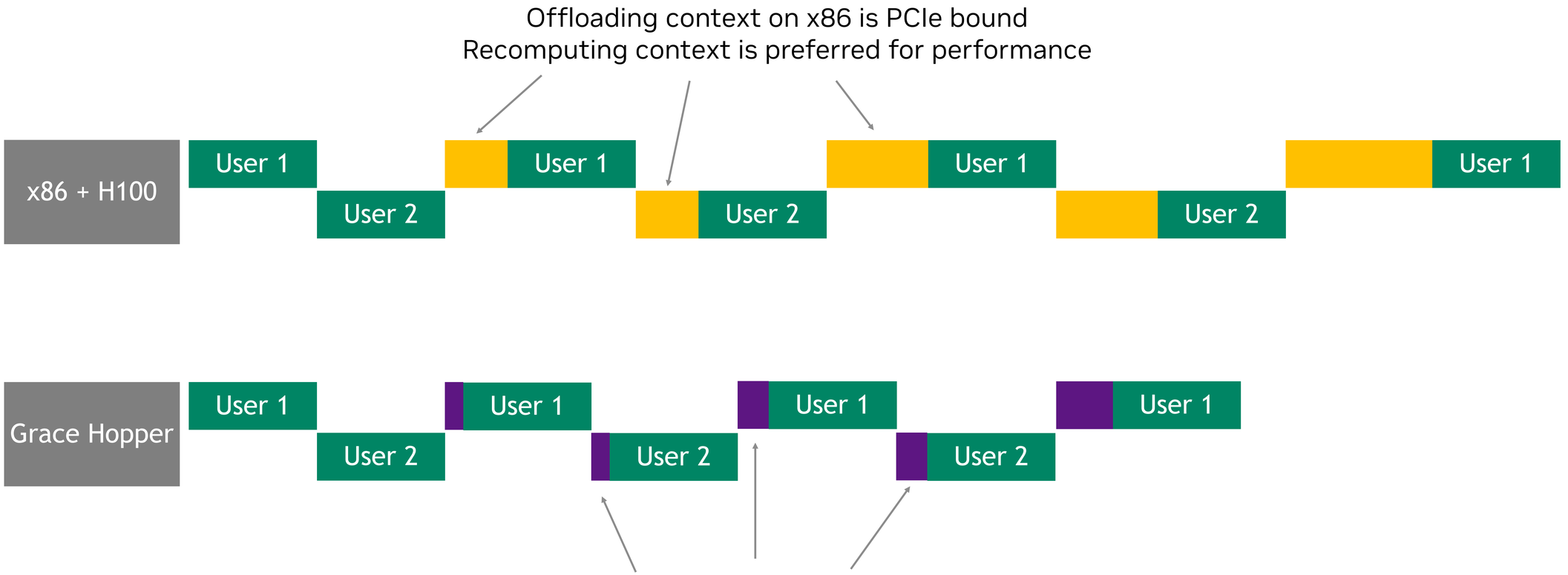
Step N: Generation Phase



LLM Inference

Multi-round dialogue

- An inference server serving multiple users needs to efficiently manage memory and compute
- Resuming idle sessions can force the server to recompute the entire context

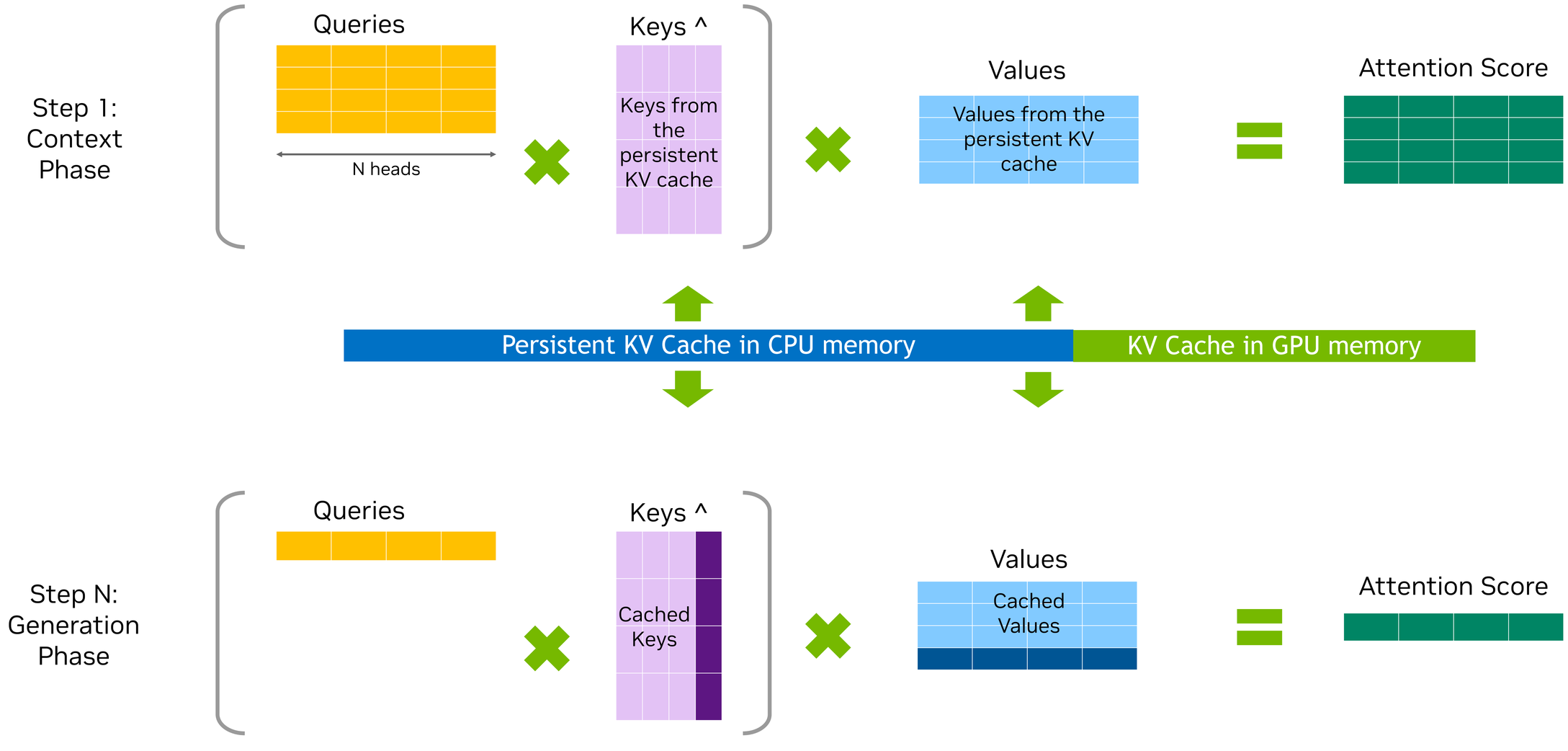


LLM Inference

Attention computation using persistent KV cache

Technique: High-bandwidth direct CPU access

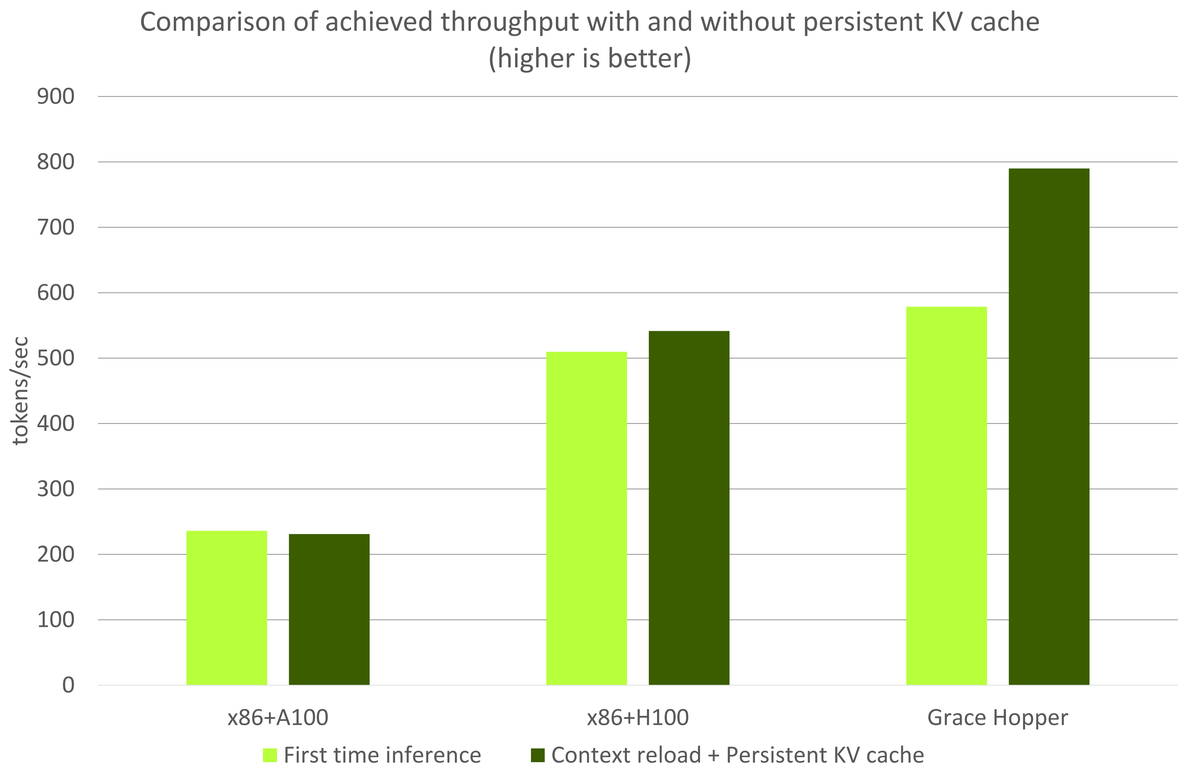
- Persistent KV cache takes it further and keeps computations from previous sequences in CPU memory



LLM Inference

KV cache offloading performance

- Context reloading from the persistent KV cache leads to up to 40% lower first token latency on Grace Hopper compared to x86+H100
- TensorRT-LLM implements KV cache reuse and KV cache offload. Available in TRT-LLM v0.9 release.



LLM Inference: Parameter Offloading

Thanks to: Weiliang Liu, Sahil Modi, Rajnish Aggarwal (TRT-LLM team),
Martin Mehringer (DevTech Compute EMEA)

LLM Inference

Parameters offloading optimization knobs in TensorRT

- With Grace Hopper a part of parameters can be offloaded on Grace Memory.

LLM Inference

Parameters offloading optimization knobs in TensorRT

Technique: Asynchronous parameter offloading

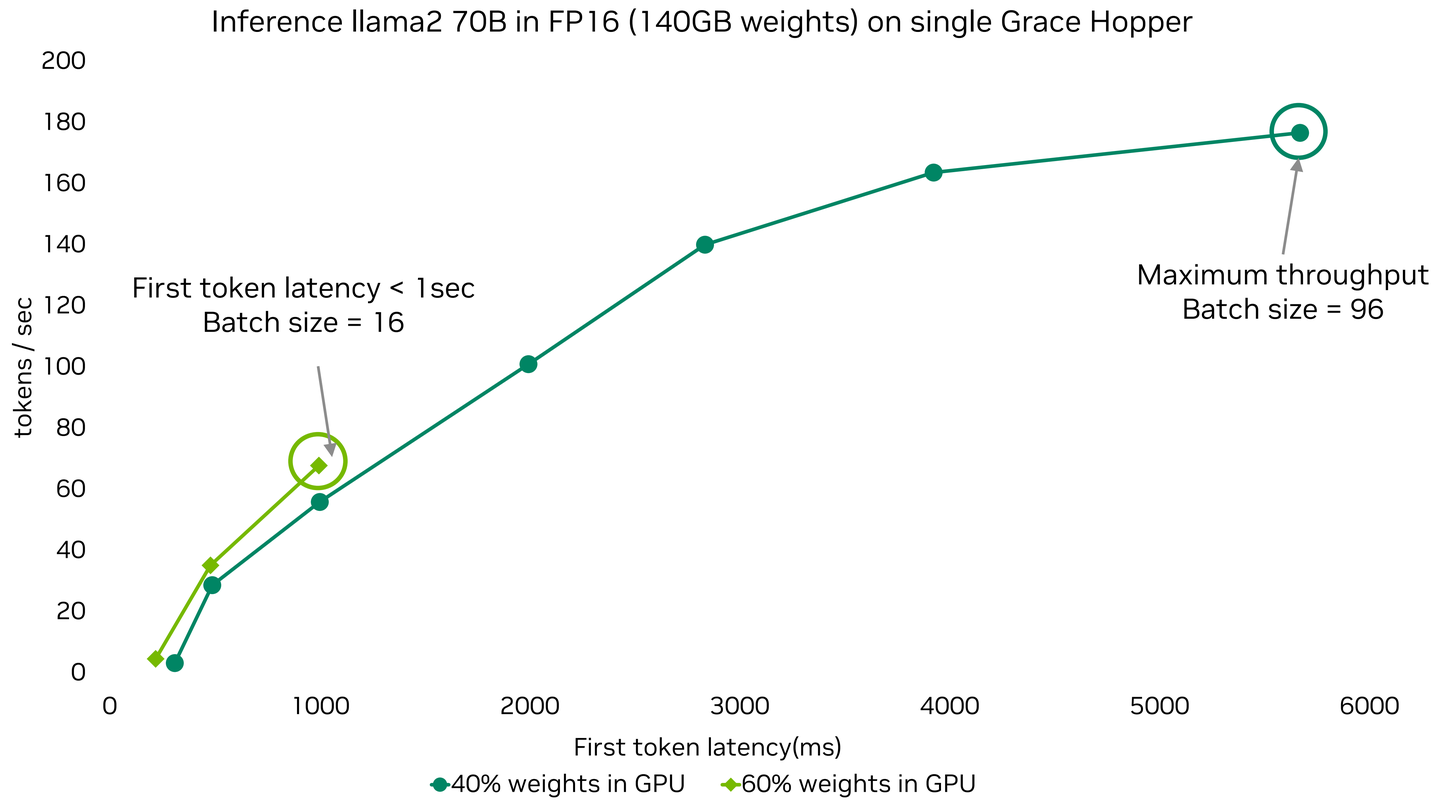
- With Grace Hopper a part of parameters can be offloaded on Grace Memory.
- TRT 10.0 will allow user to build inference engine with offloading enabled.
- Building TRT execution engine:
  - trtexec … --stronglyTyped --enableWeightStreaming --weightStreamingBudget=<budget_megabytes>M

LLM Inference

Improved throughput for Grace Hopper

Technique: Asynchronous parameter offloading

- Parameter streaming has limited value on x86 platforms due to high first-token-latencies and end to end latencies
- On Grace Hopper offloading allows running larger models at reasonable performance.

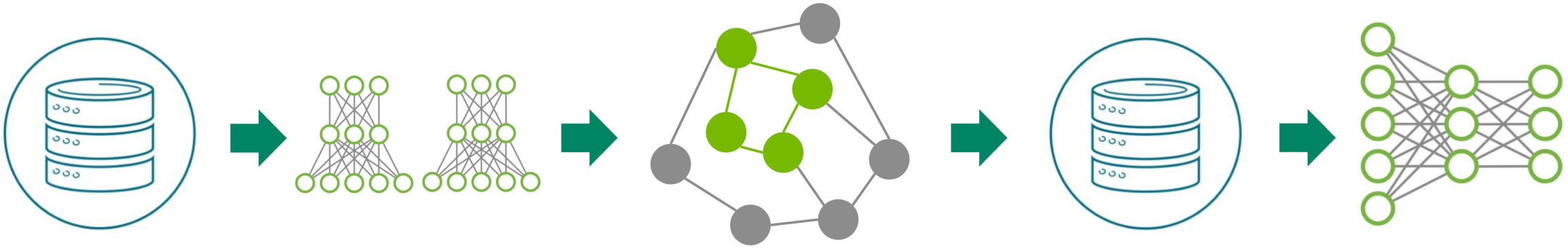


Recommender Workflows

Thanks to: Abdurrahman Yasar, Chirayu Garg (DevTech Compute NALA)

Recommender Workflows

Latency pipeline



Retrieval User Features
Pandas

Two-Tower Model
Onnx runtime

Approximate Nearest Neighbor Graph
Rapids RAFT

Retrieval Item Features
Pandas

DLRM Model
Onnx runtime

- Randomly generate queries.
- Execute each component in the pipeline one by one.
  - A component might spawn other processes or threads.

Github link for the baseline: https://github.com/RedisVentures/Redis-Recsys

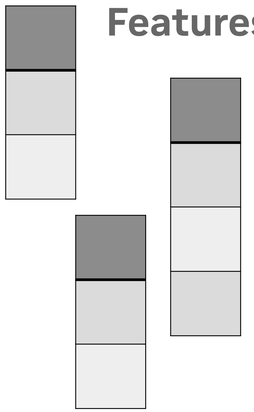
Recommender Workflows

Dataset properties

40 Continuous Features

Age	Consumption Level	...

44 Categorical Features



221k Users

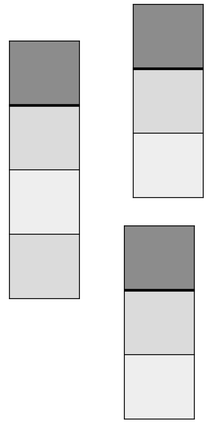
4 Million Interactions

850k Items

20 Continuous Features

Price	Stock	...

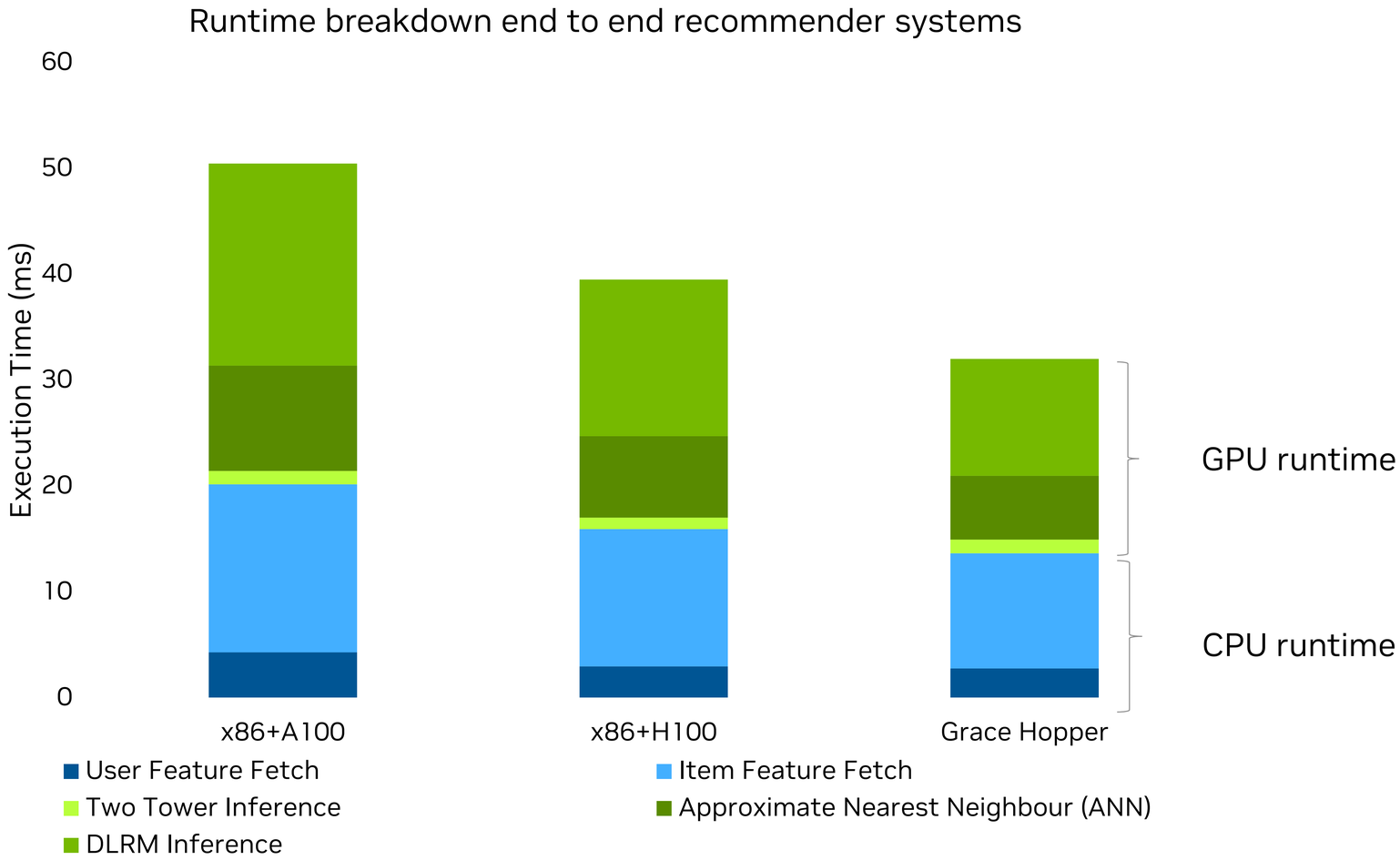
16 Categorical Features



Recommender Workflows

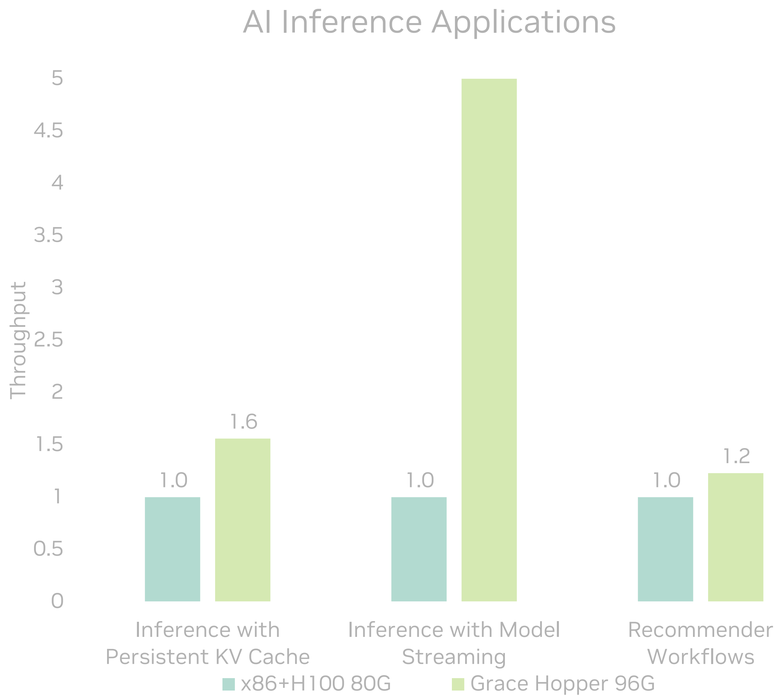
Latency analysis

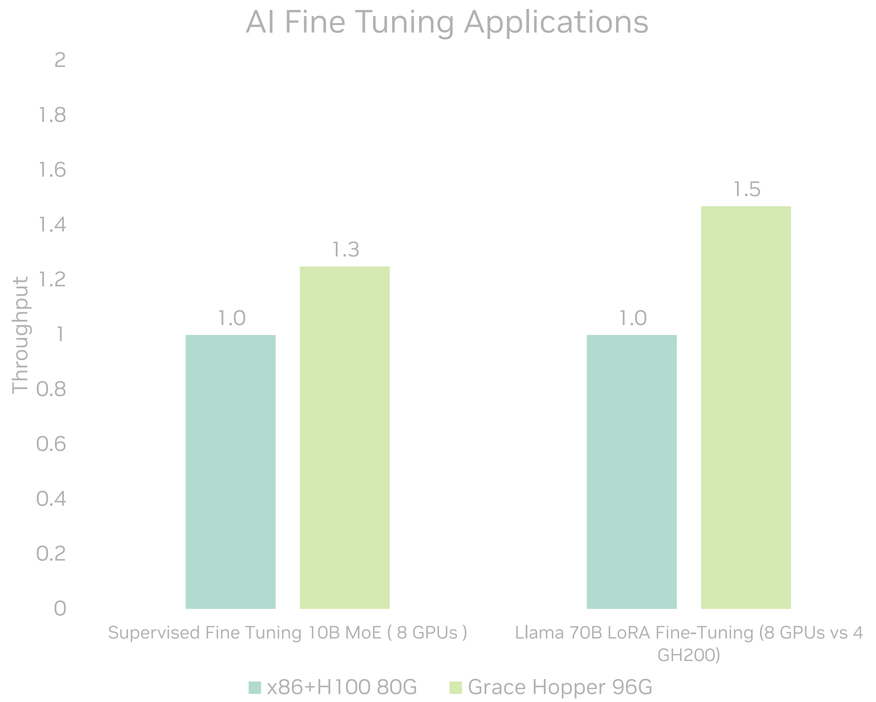
- Overall, Grace Hopper performs **1.23x** better than x86 + H100.
- Grace performs better on CPU tasks that augment recommender DL models.

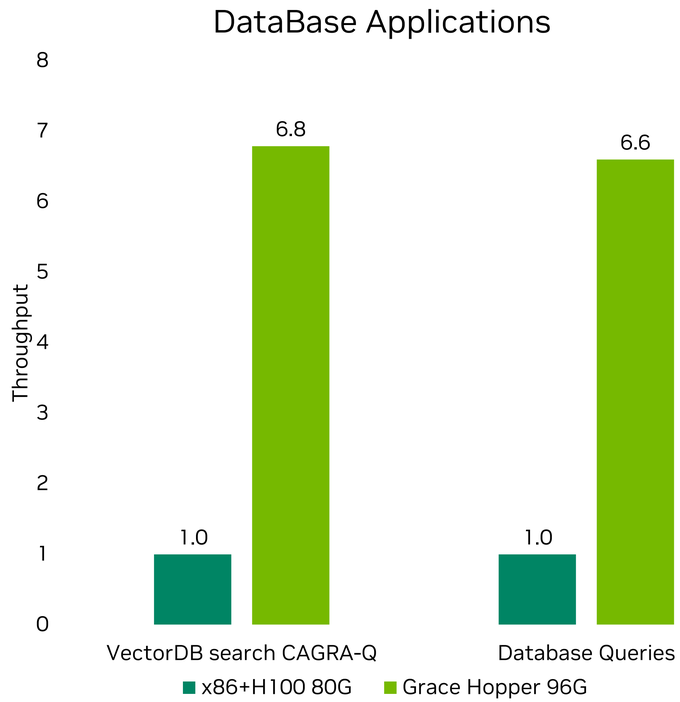


Grace Hopper Performance sneak peek

Improved GPU utilization for AI applications







GH200: Grace CPU 480GB LPDDR5, H100 96GB HBM3
DGX H100 80GB HBM3

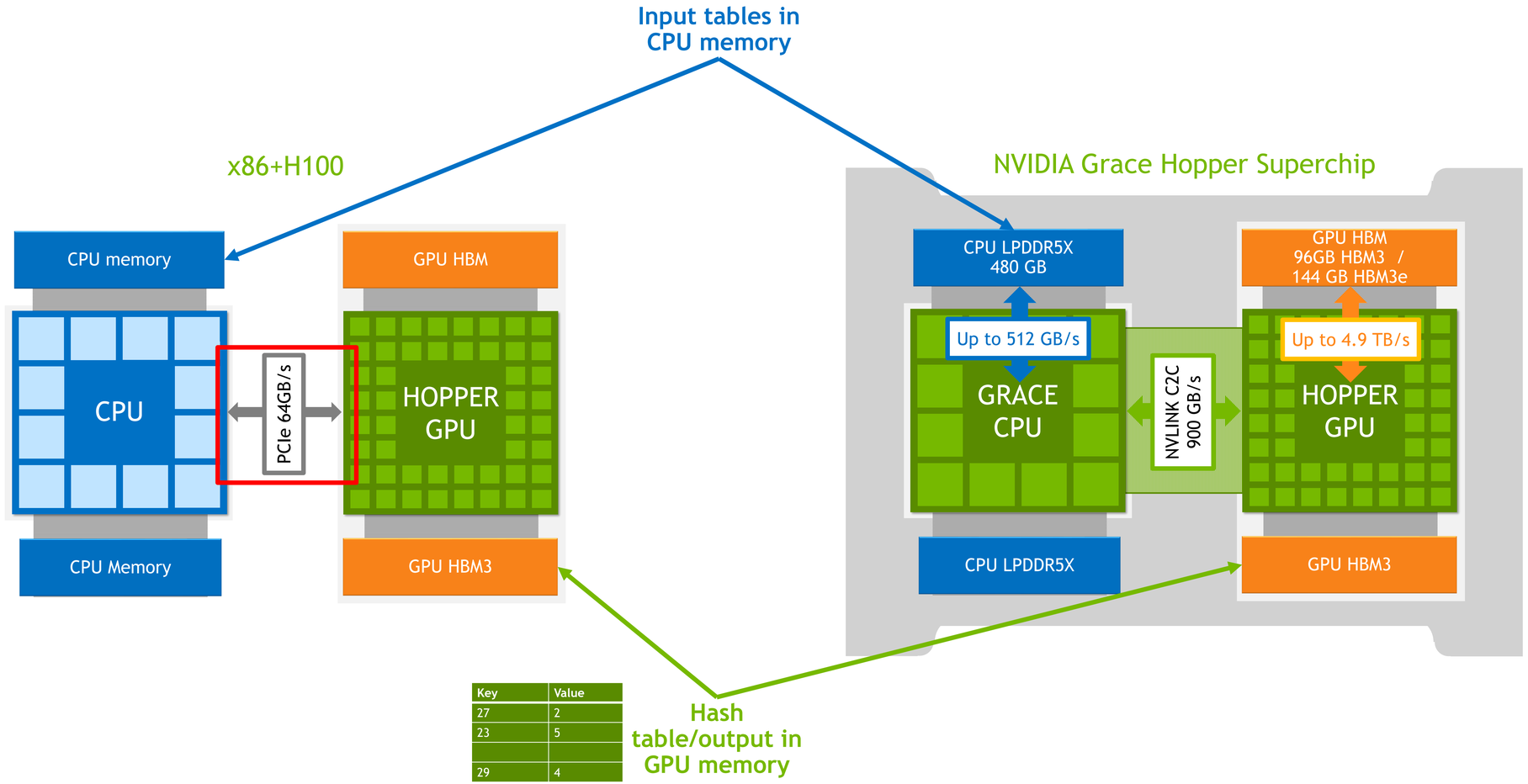
Database Queries

Thanks to: Rui Lan, Eyal Soha, Nikolay Sakharnykh
(DevTech Compute NALA)

Database Queries

Database queries x86 vs Grace Hopper

- PCIe becomes bottleneck on x86 platform.
- On Grace Hopper with NVLink C2C bandwidth can support database queries.



Database Queries

Performance for different database queries

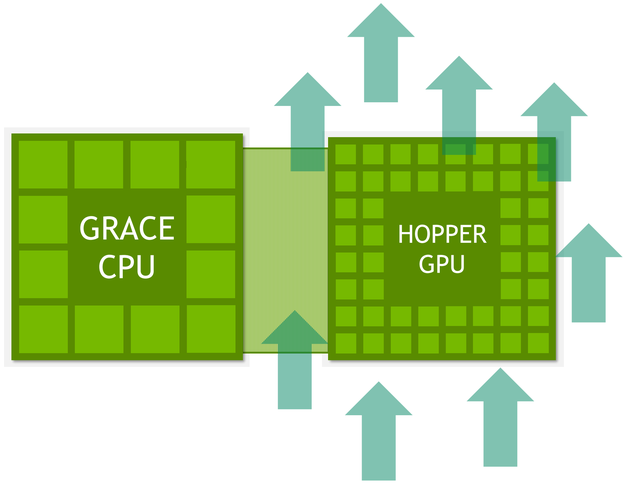
Technique:
High-bandwidth
direct CPU access

- PCIe becomes bottleneck on x86 platform.
- On Grace Hopper with NVLink C2C bandwidth can support database queries.

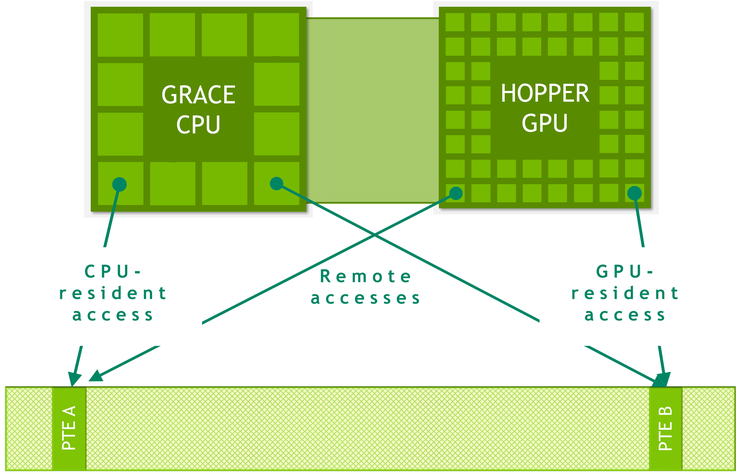
Conclusion

Key Takeaways

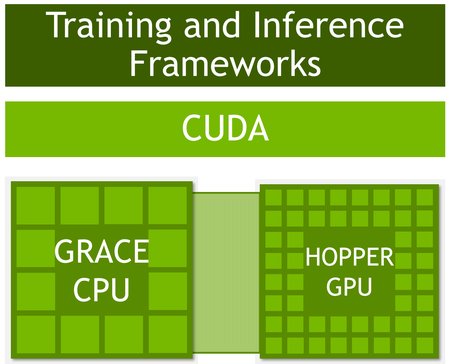
Grace Hopper for AI applications



Higher GPU Throughput
through asynchronous offloading



High Bandwidth Cache for system memory
through Unified Memory



Optimized AI Software Stack
&
Developer Productivity

Core Performance Optimization Techniques at GTC’24

List of presentations

- Introduction to CUDA Programming and Performance Optimization [S62191]
- Advanced Performance Optimization in CUDA [S62192]
- Performance Optimization for Grace CPU Superchip [S62275]
- Grace Hopper Superchip Architecture and Performance Optimizations for Deep Learning Applications [S61159]
- Multi GPU Programming Models for HPC and AI [S61339]
- More Data, Faster: GPU Memory Management Best Practices in Python and C++ [S62550]
- Harnessing Grace Hopper's Capabilities to Accelerate Vector Database Search [S62339]
- From Scratch to Extreme: Boosting Service Throughput by Dozens of Times with Step-by-Step Optimization [S62410]

- Recommended
  - S62550, today at 3PM, by Mark Harris on memory allocators
  - CWE61236, tomorrow at 10AM, by DevTech on Grace Hopper: more in-depth answers to questions about this presentation