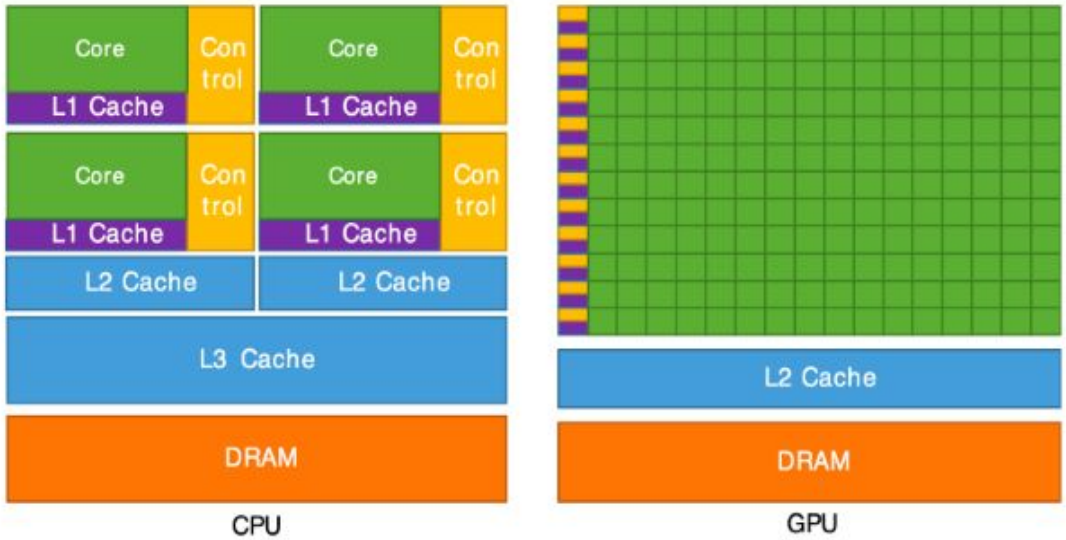
Pump Up the Volume: Processing Large Data on GPUs with Fast Interconnects

Team members: Xiao Zhang
Chen Zhu
Ran Wei
Xingjian Zhang

What is a GPU? Graphics Processing Unit



DRAM: main memory.
L1/L2/L3 Cache: high speed cache.

Control: control unit.
Core: computation unit

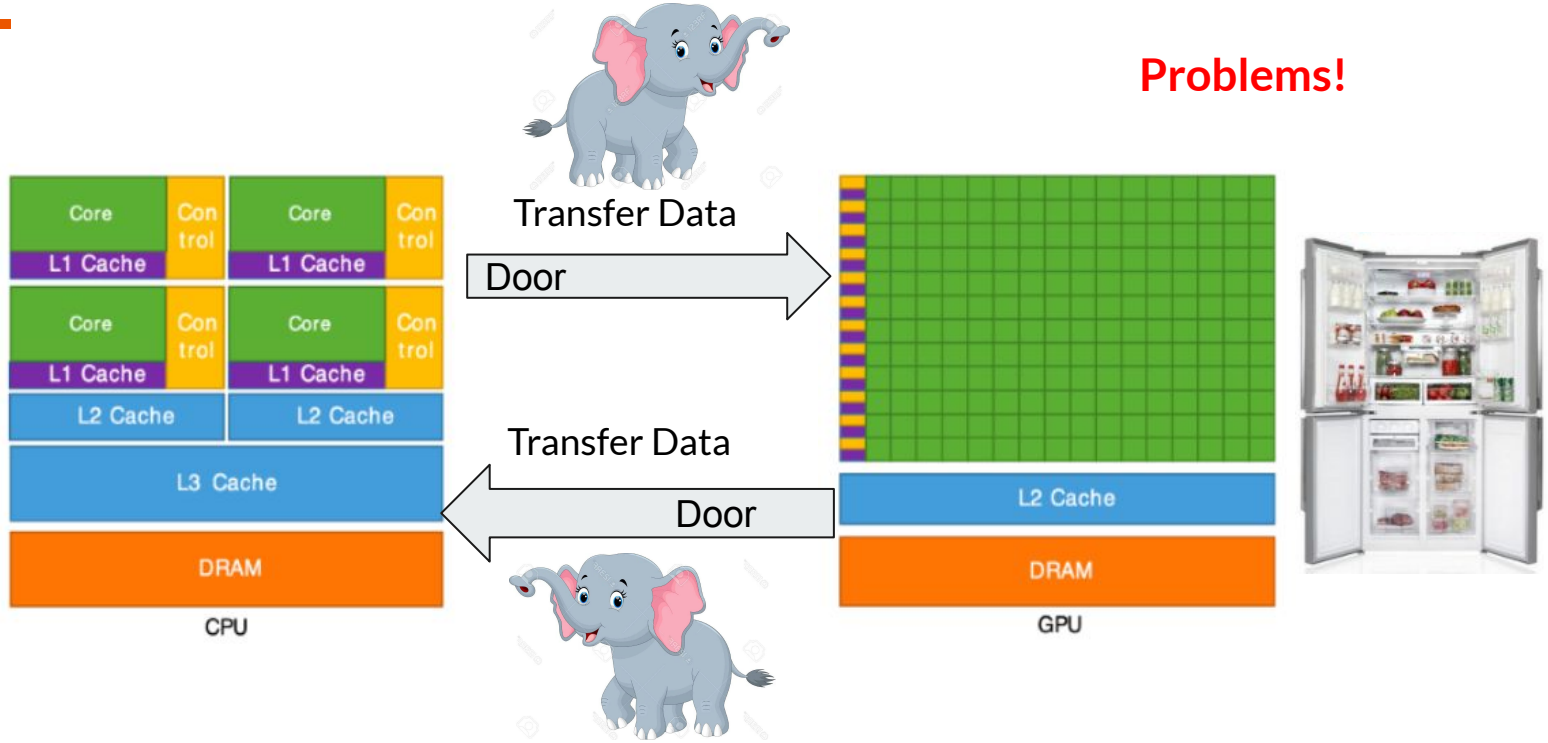
What can we see from the right picture?

https://cvw.cac.cornell.edu/GPUarch/gpu_characteristics

How to apply GPU in database area?

Acceleration! But how?

How to apply GPU in database area?

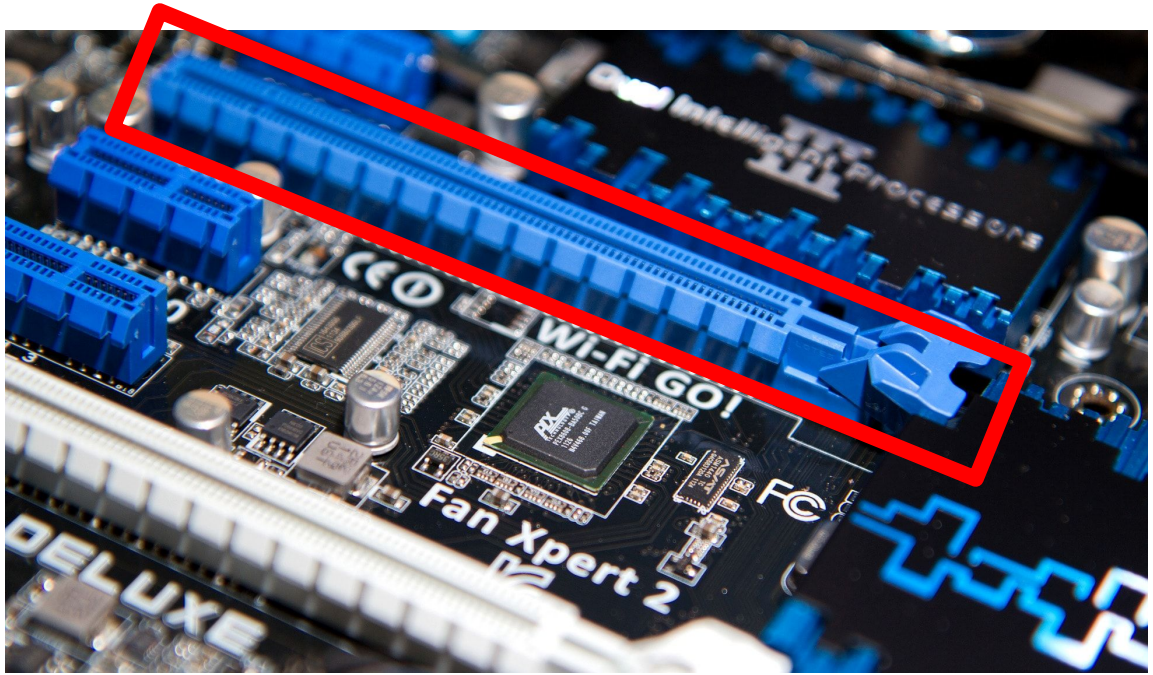


Three fundamental limitations

1. Low interconnect bandwidth
2. Small GPU memory capacity
3. Coarse-grained cooperation of CPU and GPU

Let’s see how we can break these limitations with the new technology.

State-of-art data channel: PCIe 3.0



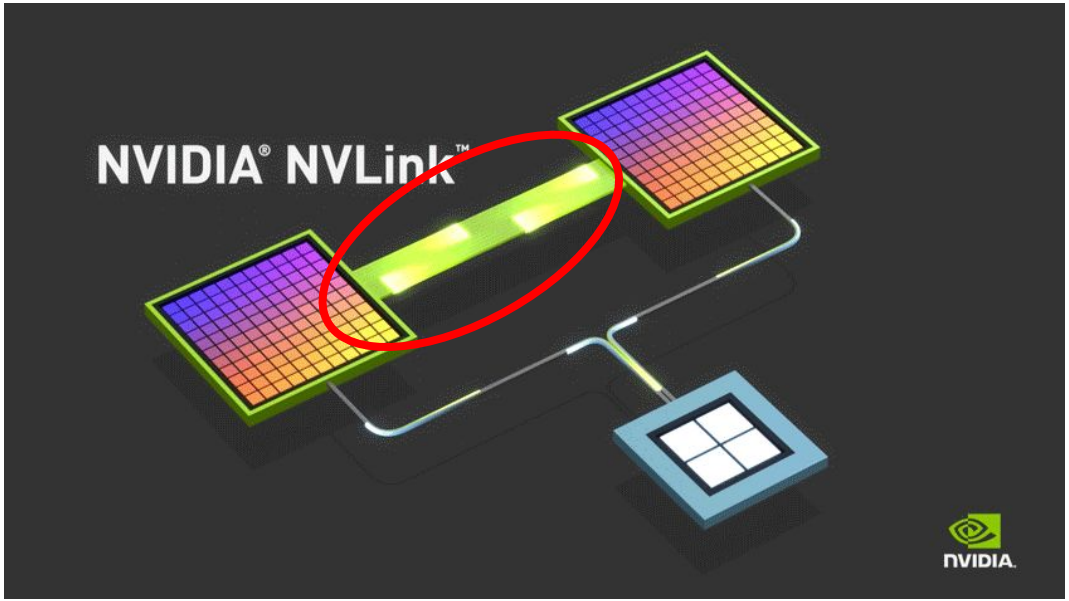
General usage.

16GB/s

Limitation 1:
Low interconnect bandwidth

New technology: NVLink 2.0

Higher Speed! Lower Latency! Higher Bandwidth!

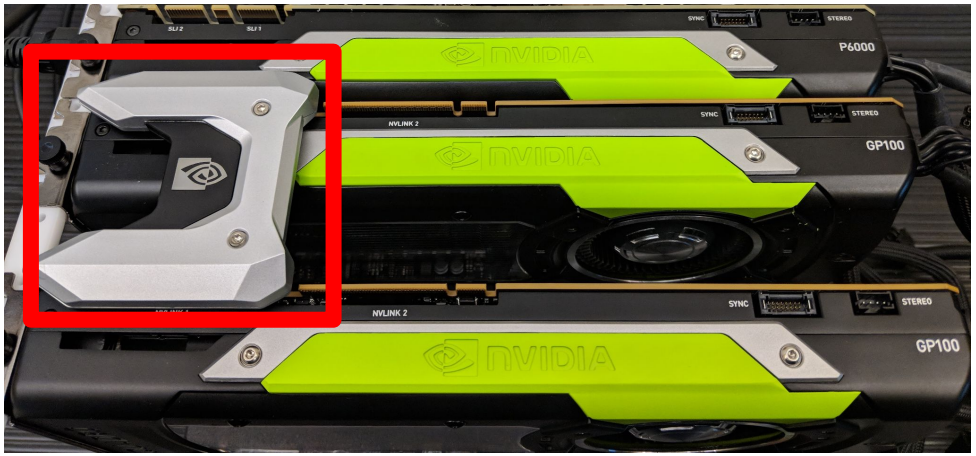


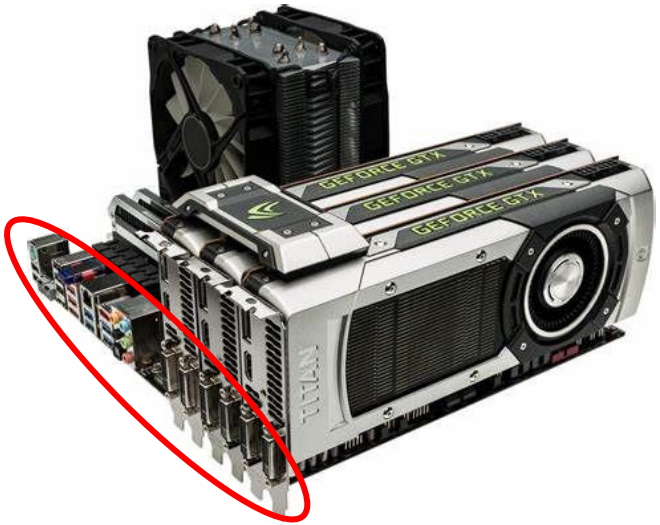
Special designed for GPU

75GB/s

Limitation 1?
Low interconnect bandwidth?

NVLink 2.0





Interconnection between GPUs !

Interconnection between GPU and CPU is still PCIe 3.0

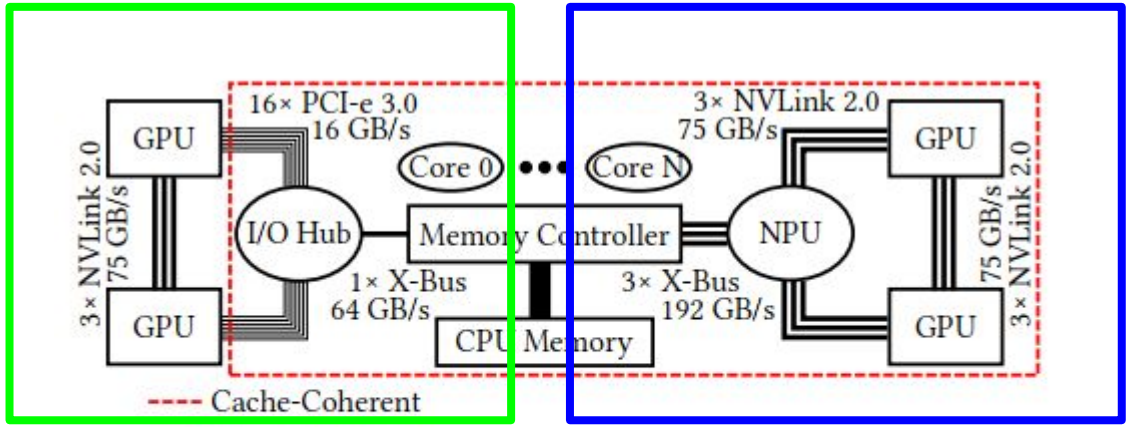
How could interconnection help us with the capacity problem?

1. ~~Low interconnect bandwidth~~ ——SOLVED
2. Small GPU memory capacity ?

Cache Coherence

NVLink 2.0

PCIe 3.0



Cache-Coherence: Data consistency.

~~**Limitation 2: Memory Capacity.**~~

Three fundamental limitations

1. **Low interconnect bandwidth**
2. **Small GPU memory capacity**
3. **Coarse-grained cooperation of CPU and GPU**

Coming up with NVLink 2.0 against PCIe

Let's see exactly how can NVLink 2.0 improves everything above.

Fast Interconnect

In previous discussion, we see three main advantages of fast interconnect:

Higher Bandwidth & Lower Latency

Cache Coherence

Performance Comparison of Interconnects

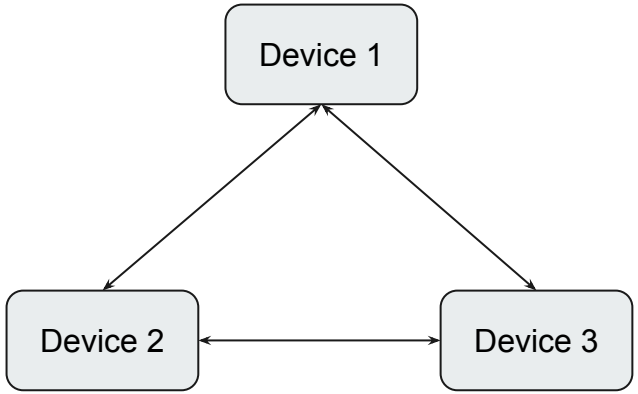
Fast Interconnect: NVLink 2.0

GPU Interconnect: PCI-e 3.0

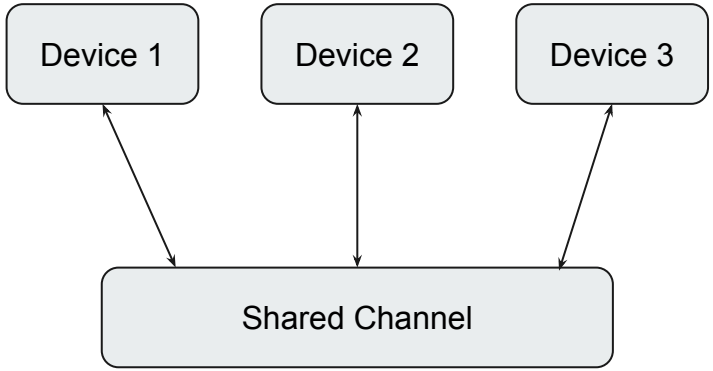
CPU Interconnect: Intel Ultra Path Interconnect (Point-to-Point Interconnect)

IBM X-Bus (Bus Interconnect)

Point-to-Point Interconnect *vs* Bus Interconnect

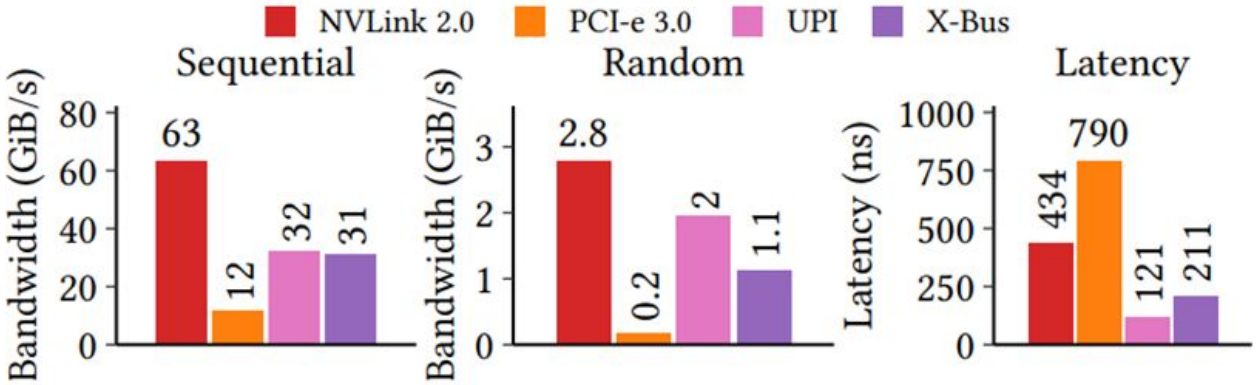


Point-to-Point Interconnect



Bus Interconnect

Performance Comparison of Interconnects



(a) NVLink 2.0 vs. CPU & GPU Interconnects.

Fast Interconnect

In previous discussion, we see three main advantages of fast interconnect:

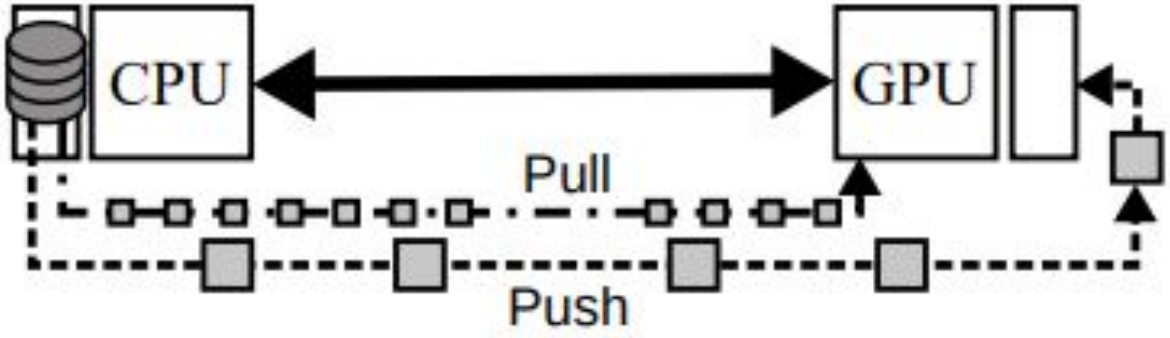
Higher Bandwidth & Lower Latency

Cache Coherence

Limitation 1: Low interconnect bandwidth (Solved)

Limitation 2: Small GPU memory capacity

Data Transfer between CPU and GPU



8 Methods of Data Transmission

Method	Semantics	Level	Granularity	Memory
Pageable Copy Staged Copy Dynamic Pinning	Push	SW	Chunk	Pageable
Pinned Copy				Pinned
UM Prefetch				Unified
UM Migration	Pull	OS	Page	Unified
Zero-Copy		SW	Byte	Pinned
Coherence		HW		Pageable

Data Transfer between CPU and GPU

Zero-copy

Using Unified Virtual Addressing (UVA)

Avoid any explicit copying of data

Still require additional software

Coherence

New approach enabled by NVLink 2.0

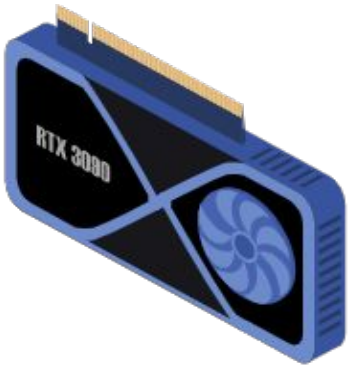
Based on Hardware Address Translation (HAT)

Totally managed in hardware

Limitation 2: Small GPU memory capacity

Scaling GPU Hash Joins to Arbitrary Data Sizes

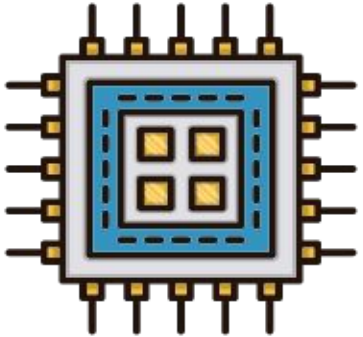
GPU



Smaller memory

VS

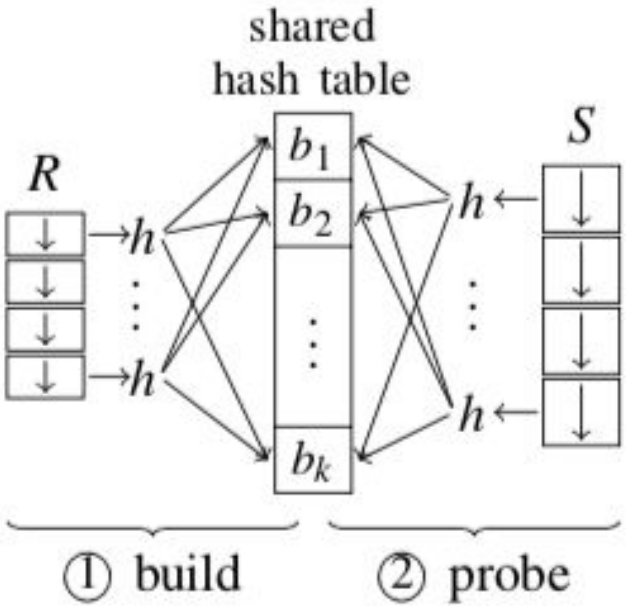
CPU



Larger memory

https://www.iconfont.cn/search/index?searchType=icon&q=GPU

Example: Hash Joins

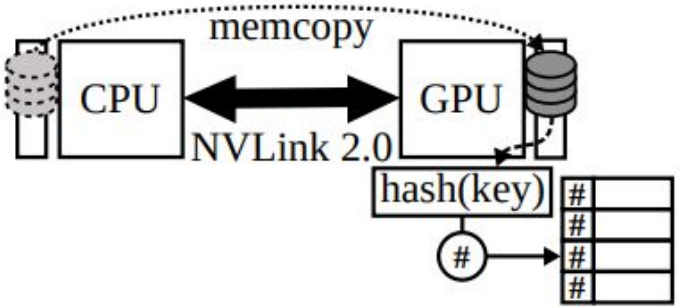


https://nan01ab.github.io/2019/03/Hash-Joins.html

Scaling the Probe Side to Any Data Size

Strategy 1. Baseline approach

Relations: GPU
Hash tables: GPU



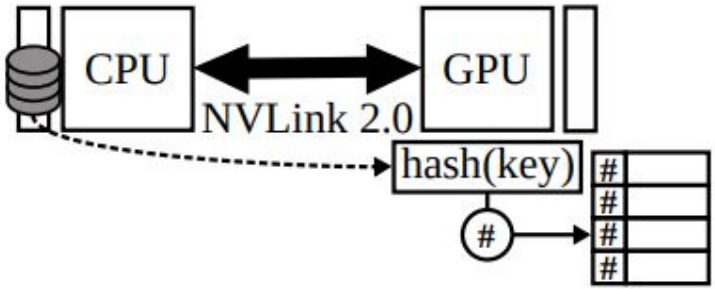
Problem?

😢 not enough GPU memory for the **whole relations**

Scaling the Probe Side to Any Data Size

Strategy 2. Probe-side scalable approach: **large relations**

Relations: CPU
Hash tables: GPU



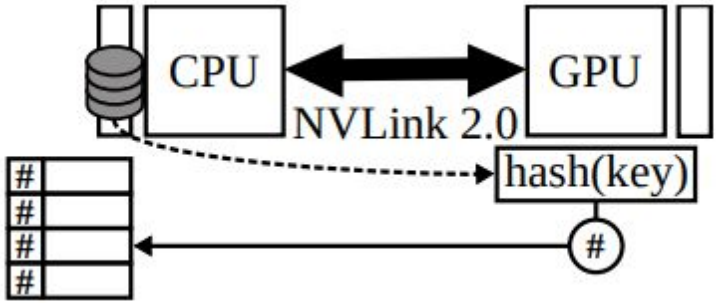
Problem?

😢 not enough GPU memory for the **hash table**

Scaling the Build Side to Any Data Size

Strategy 3. Build-side scalable approach: **large relations & large hash tables**

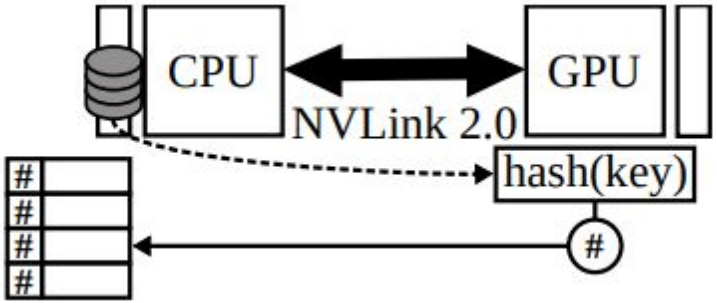
Relations: CPU
Hash tables: CPU



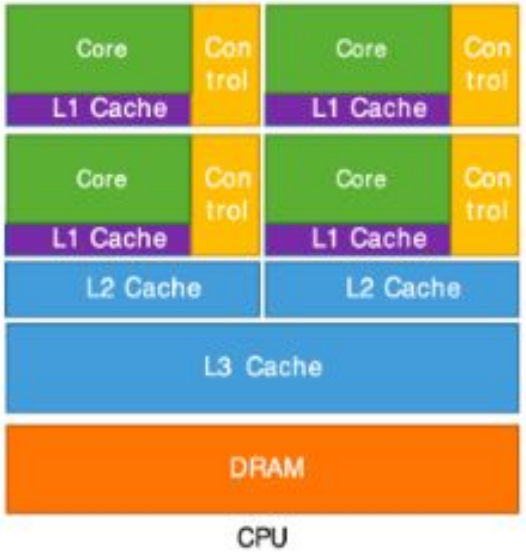
Problem?

Scaling the Build Side to Any Data Size

Strategy 3. Build-side scalable approach: **large relations & large hash tables**



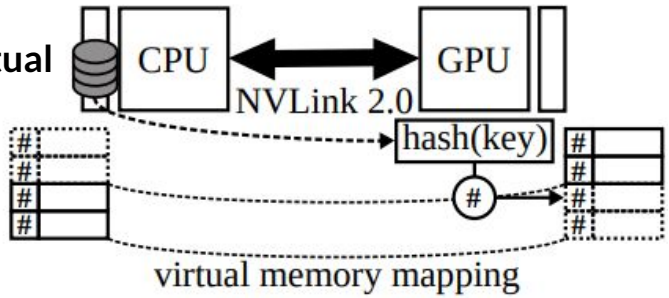
😢 **Problem: random accesses** in CPU memory is very slow



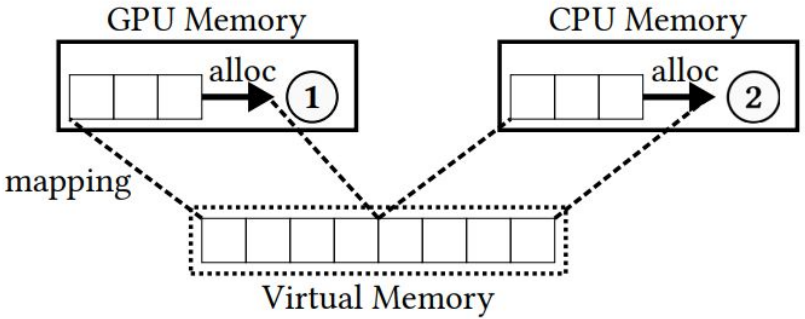
Hash Table Placement

Strategy 4. Hybrid hash table: **large relations & large hash tables**

Relations: CPU
Hash tables: Virtual



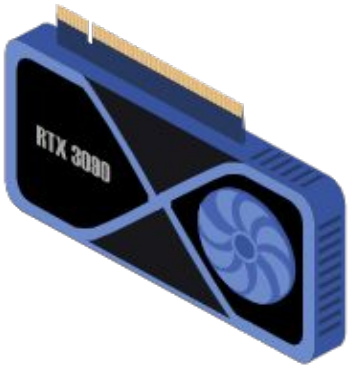
Allocating strategy:



- **Amortize the harm of large hash tables**

Scaling-up using CPU and GPU

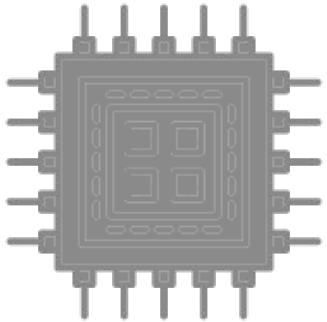
GPU



working

VS

CPU

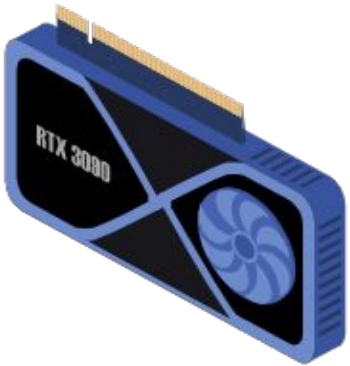


unused

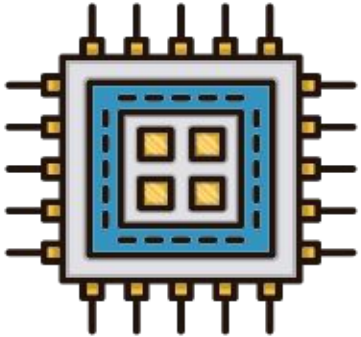
Scaling-up using CPU and GPU

GPU

CPU



co-process



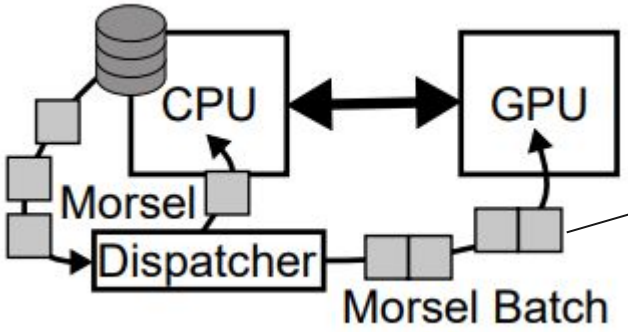
working

working

Task Scheduling

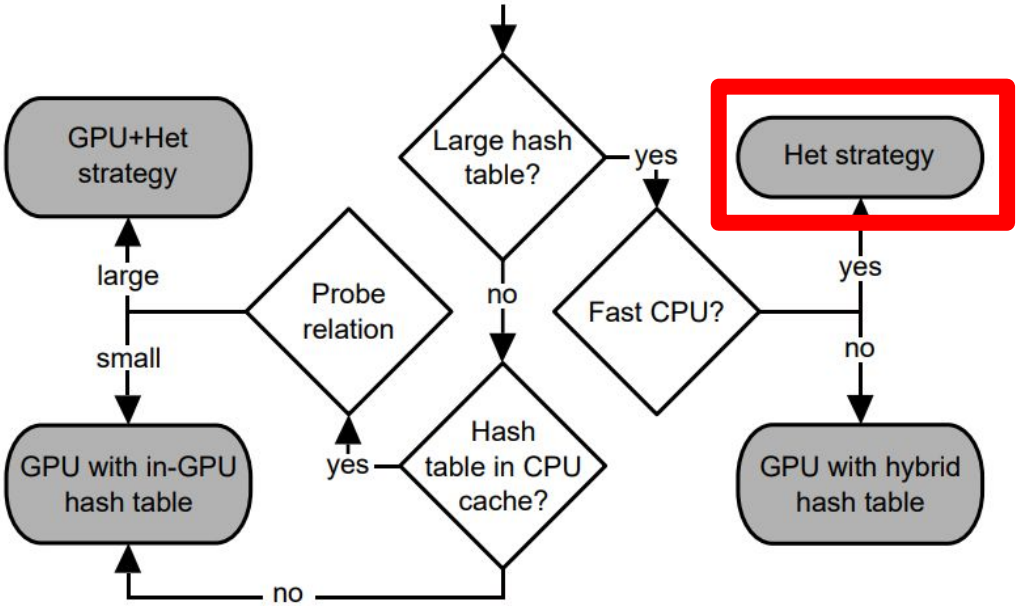
changing throughput ⇒ load imbalances

CPUs and GPUs
- actively request for tuples
- work at the same time



Morsel Batches for GPU
- High latency of scheduling work
- Higher processing rate of GPU

Heterogenous Hash Table Placement

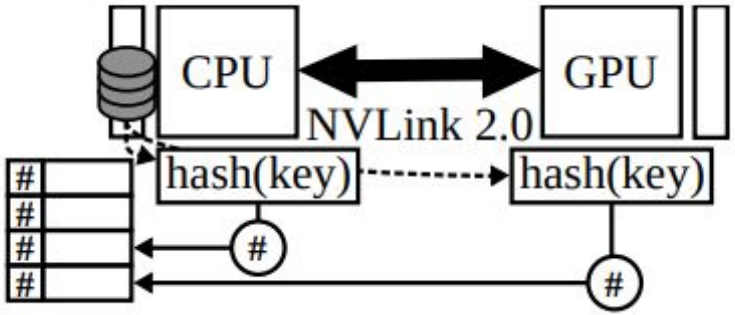


Relations: CPU
Hash tables: CPU
Co-process: Yes

Heterogenous Hash Table Placement

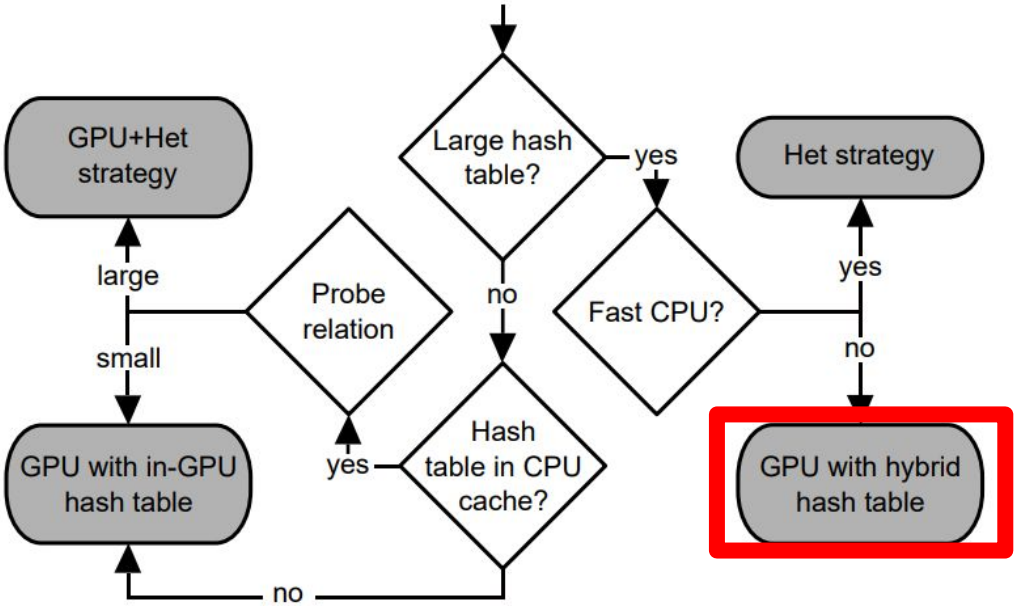
Strategy 5. Het strategy: both relations and hash tables are in CPU memory

Relations: CPU
Hash tables: CPU
Co-process: Yes



always speed up

Heterogenous Hash Table Placement



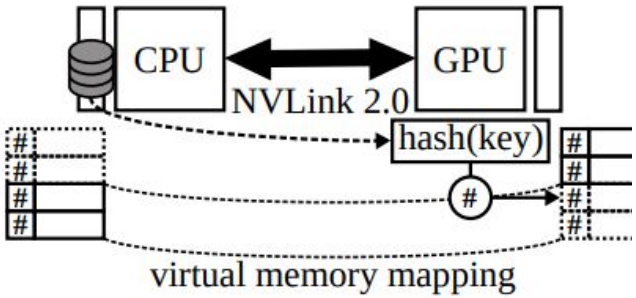
Relations: CPU
Hash tables: CPU
Co-process: Yes

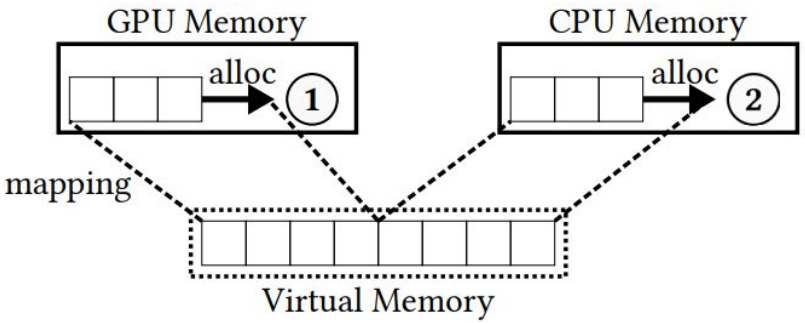
Relations: CPU
Hash tables: CPU
Co-process: No

Heterogenous Hash Table Placement

Strategy 4. Hybrid hash table: relations are in CPU memory and hash tables are in virtual memory

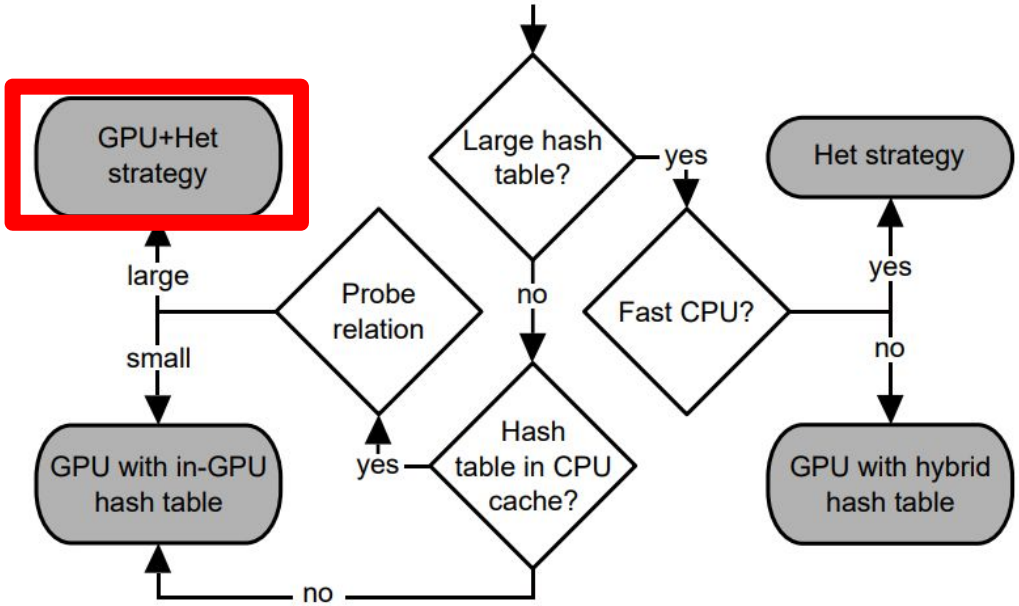
Relations: CPU
Hash tables: CPU
Co-process: No





Heterogenous Hash Table Placement

Relations: CPU
Hash tables: GPU
Co-process: Yes



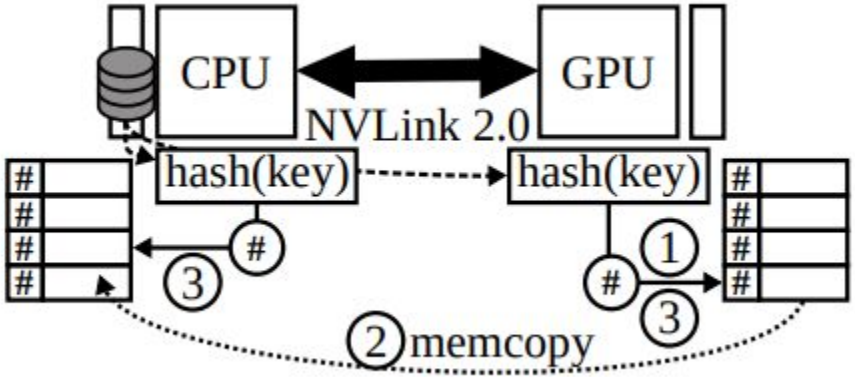
Relations: CPU
Hash tables: CPU
Co-process: Yes

Relations: CPU
Hash tables: CPU
Co-process: No

Heterogenous Hash Table Placement

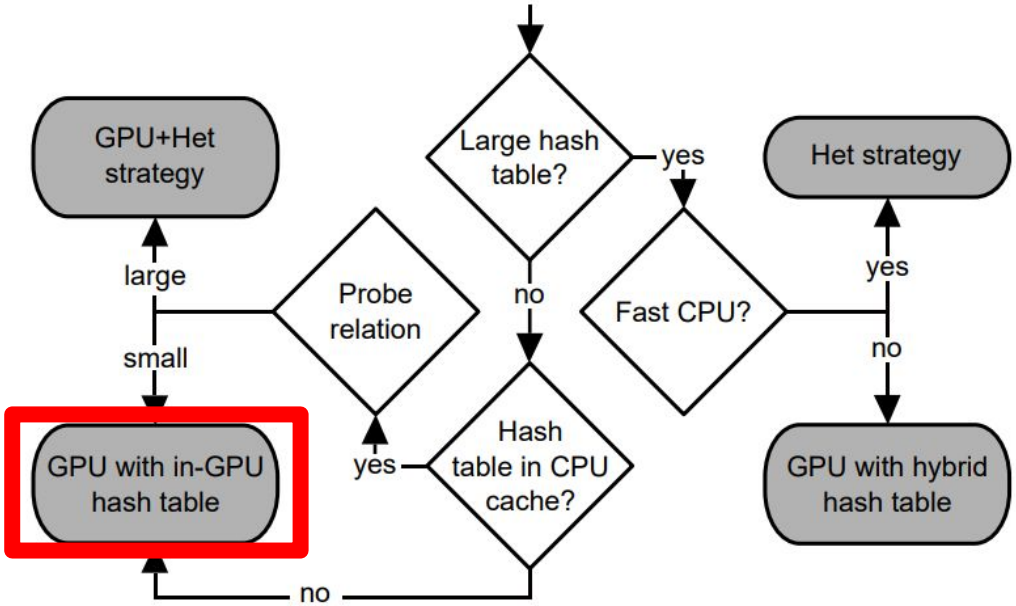
Strategy 6. GPU + Het strategy: relations are in CPU memory and a copy of hash tables is in every memory.

Relations: CPU
Hash tables: GPU
Co-process: Yes



Heterogenous Hash Table Placement

Relations: CPU
Hash tables: GPU
Co-process: Yes



Relations: CPU
Hash tables: CPU
Co-process: Yes

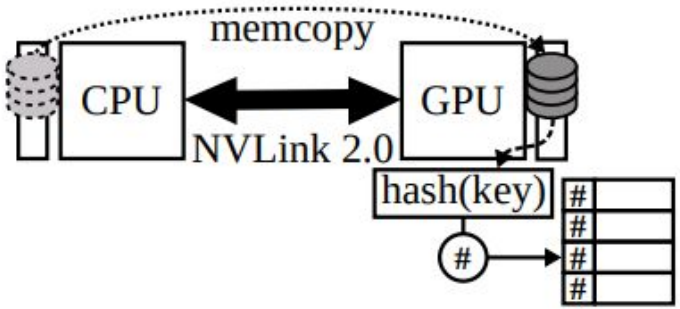
Relations: GPU
Hash tables: GPU
Co-process: No

Relations: CPU
Hash tables: CPU
Co-process: No

Heterogenous Hash Table Placement

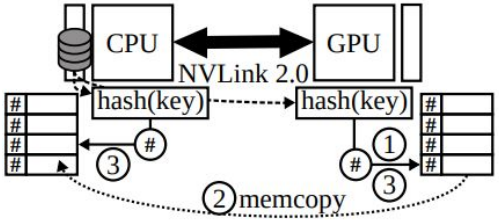
Strategy 1. Baseline approach: both relations and hash tables are in GPU memory

Relations: GPU
Hash tables: GPU
Co-process: No



Multi-GPU Hash Table Placement

Small hash table -> strategy 6. GPU + Het strategy

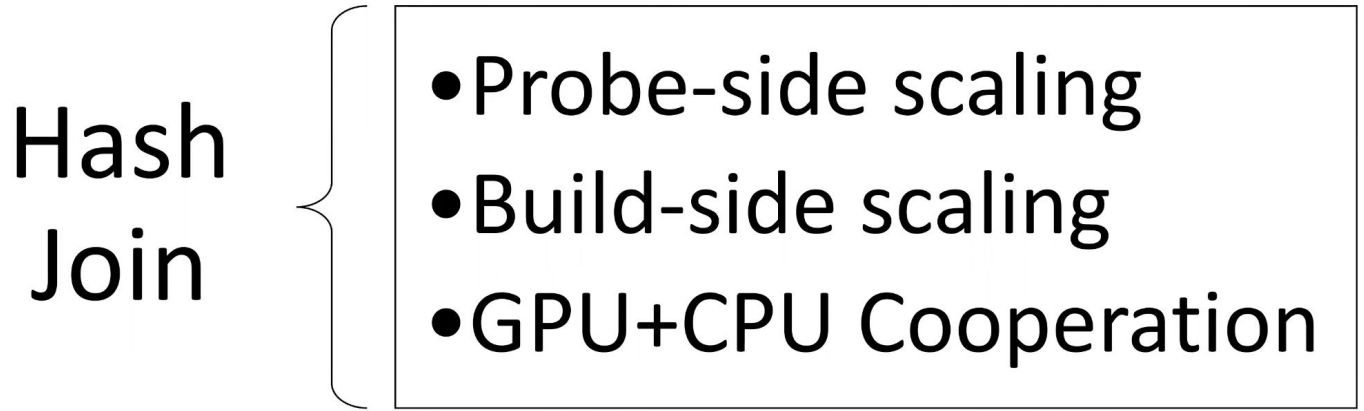


Large hash table -> strategy 7. Hybrid hash table in GPUs (interleave the pages over all GPUs)

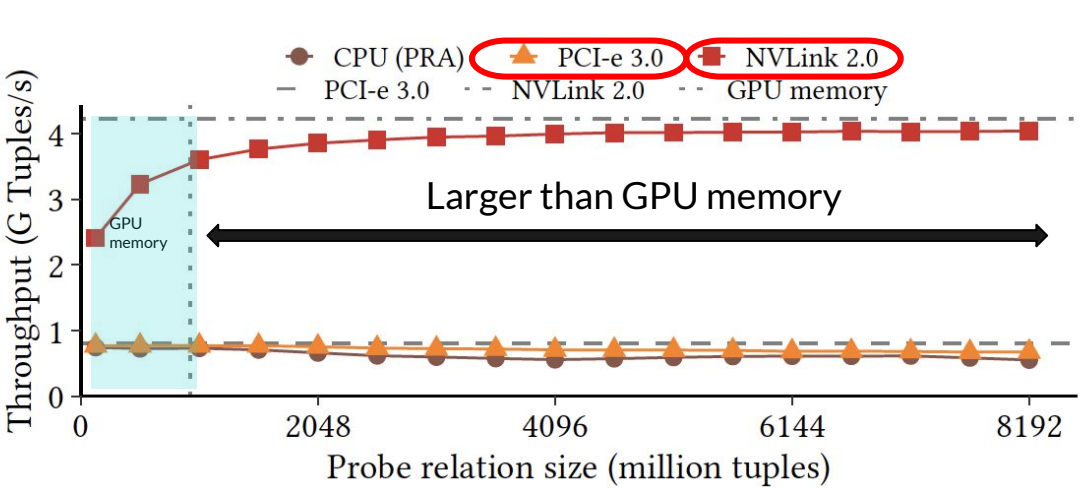
😄 Advantages of only using multiple GPUs:

- Avoid computational skew
- Free CPU memory bandwidth
- Utilize the full bi-directional bandwidth of fast interconnects

Evaluation


Hash Join
•Probe-side scaling
•Build-side scaling
•GPU+CPU Cooperation

Probe-Side Scaling



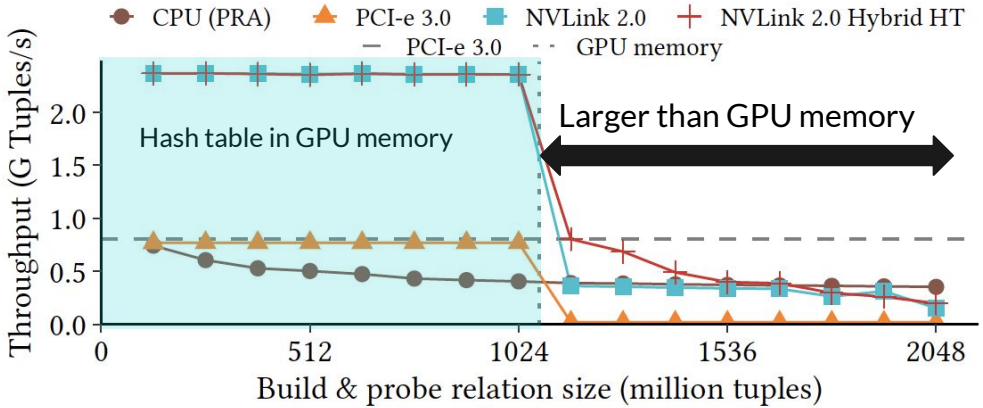
NVLink 2.0

NVLink 2.0

Baseline of the CPU

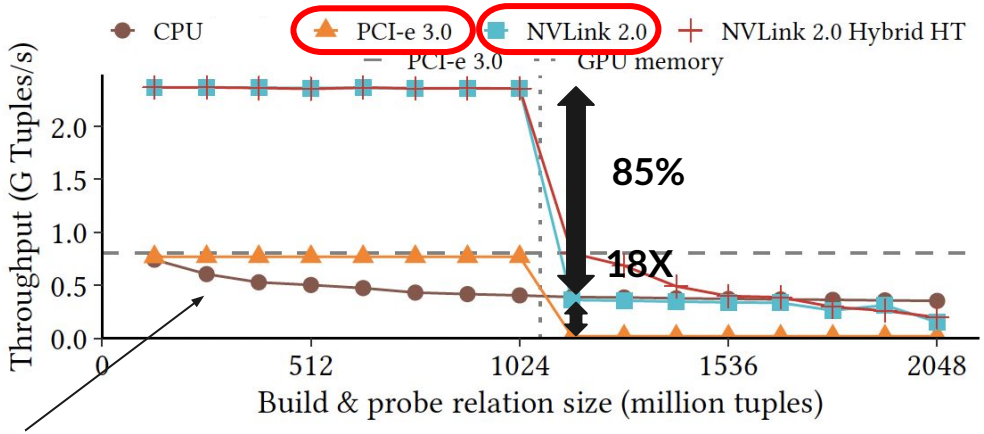
- Up to 2 GiB ⋈ 122 GiB

Build-Side Scaling



- Up to 30 ⋈ 30 GiB with a 30 GIB has table = 90 Gib

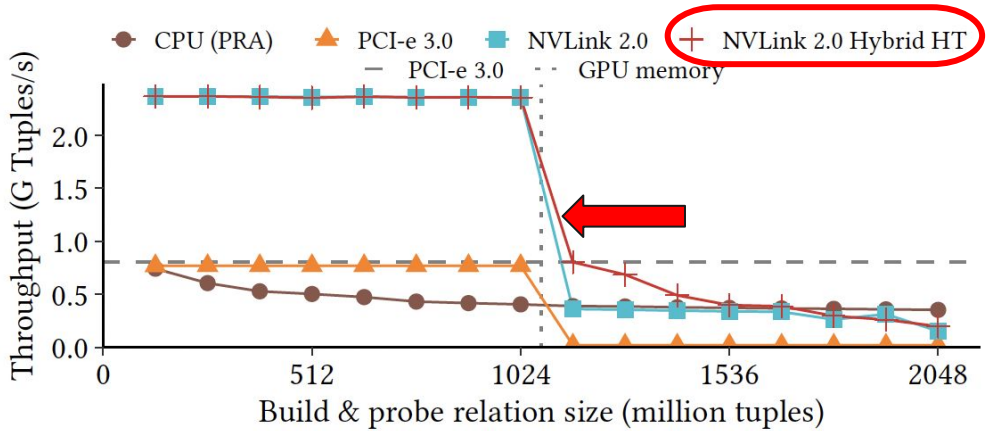
Build-Side Scaling



CPU

- Up to 30 ⋈ 30 GiB with a 30 GIB has table = 90 Gib

Build-Side Scaling



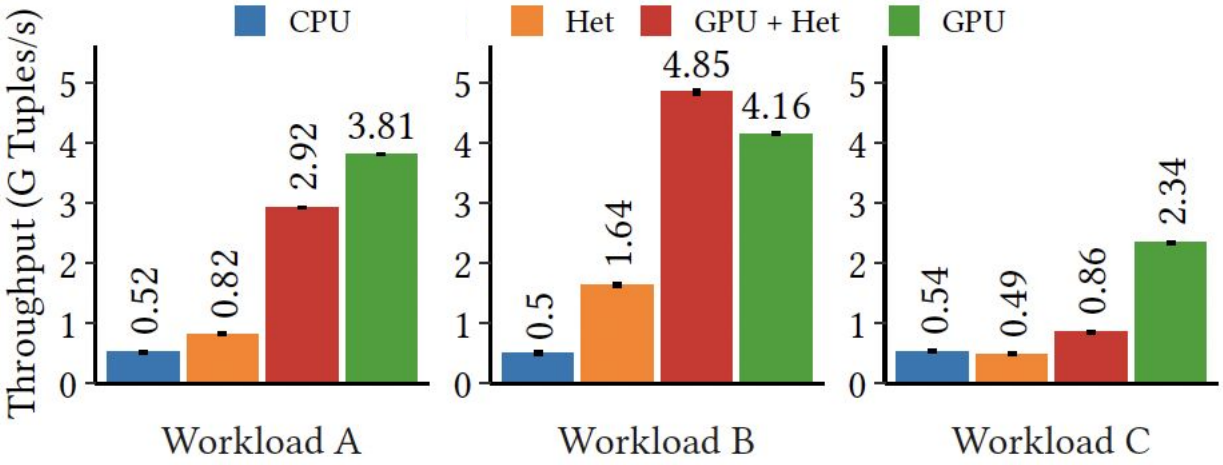
- Up to 30 ⋈ 30 GiB with a 30 GIB has table = 90 Gib

GPUs are able to operate on large, out-of-core data structures

- Hybrid hash table spills to CPU memory

Cooperative Co-processing

GPU+CPU Cooperation



Workload A: 2 GiB ⋈ 32GiB
Workload B: 4 MiB ⋈ 32GiB
Workload C: 7.8 GiB ⋈ 7.8GiB

- Heterogeneous → GPU & CPU share hash table in CPU memory
- GPU + Het → GPU builds hash table in GPU memory, then copy it to CPU memory

Conclusion:

Explore **in which ways** fast interconnects **benefit database**:

- Out-of-core **data sets**
- Out-of-core **data structures**
- **Fine-grained** Cooperative co-processing