PUMP UP THE VOLUME: PROCESSING LARGE DATA ON GPUS WITH FAST INTERCONNECTS

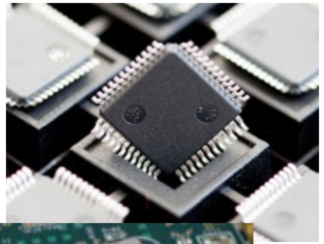
CLEMENS LUTZ ET.AL

Presented by : Syahrial Dahler

BACKGROUND

Introduction of Co Processors

- FPGA (field-programmable gate array)
- ASIC (application-specific integrated circuit)
- GPU (Graphic Processing Unit)



GPU ARCHITECTURE

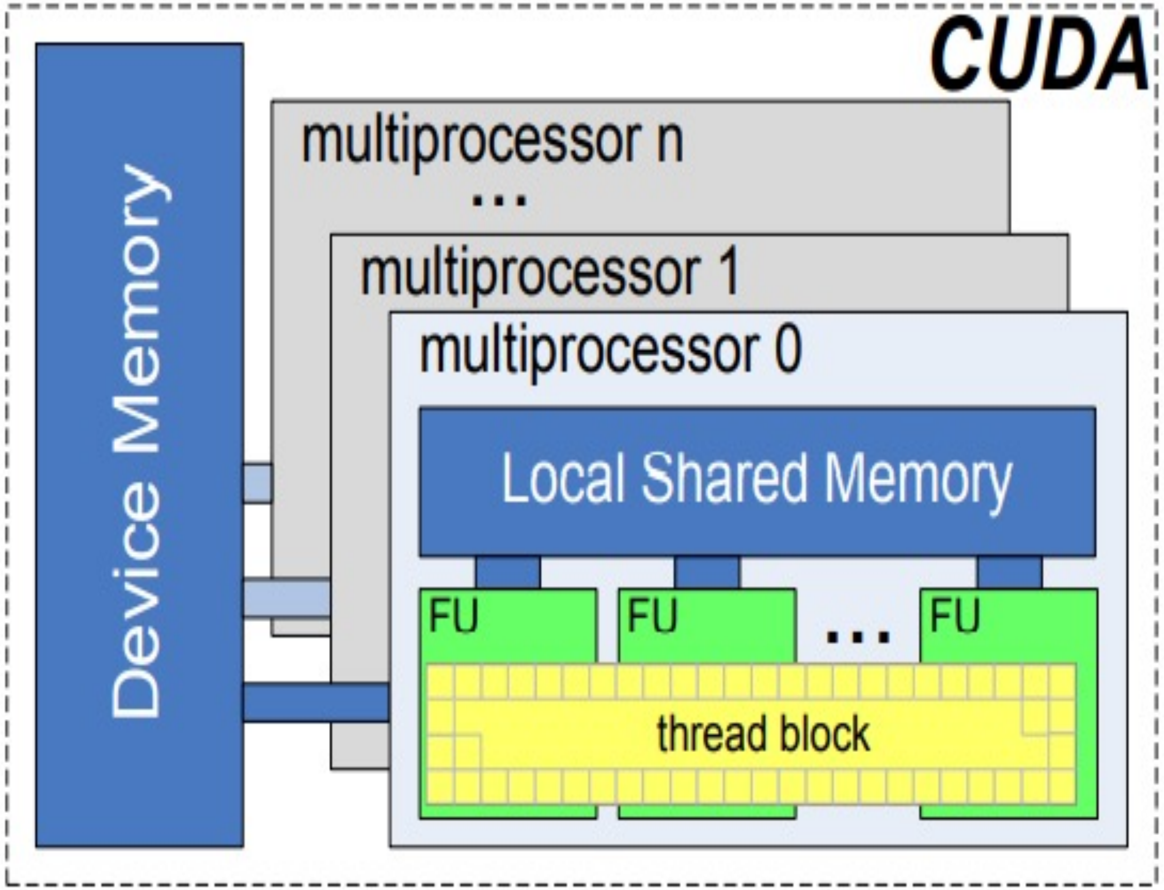
Tesla M2070 Processor:

Streaming Multiprocessors (SM): 14

Streaming Processors on each SM: 32

Total cores = 14 x 32 = 448 cores

Each Streaming Multiprocessor supports 1024 threads.



Compute unified device architecture

ADVANTAGES OF GPU

Parallel Processing → Faster result

GPU IN MODERN DATABASES

CHALLENGES OF USING GPU FOR DATABASE APPLICATION

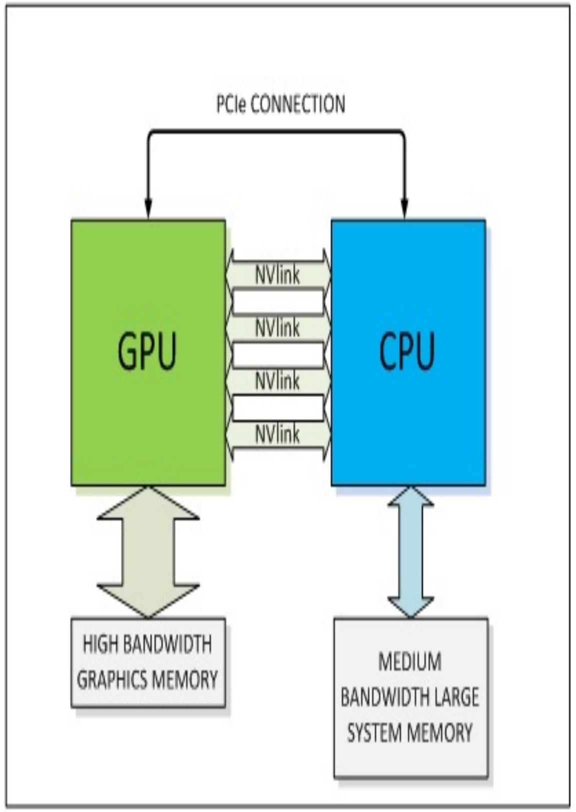
Transfer Bottleneck

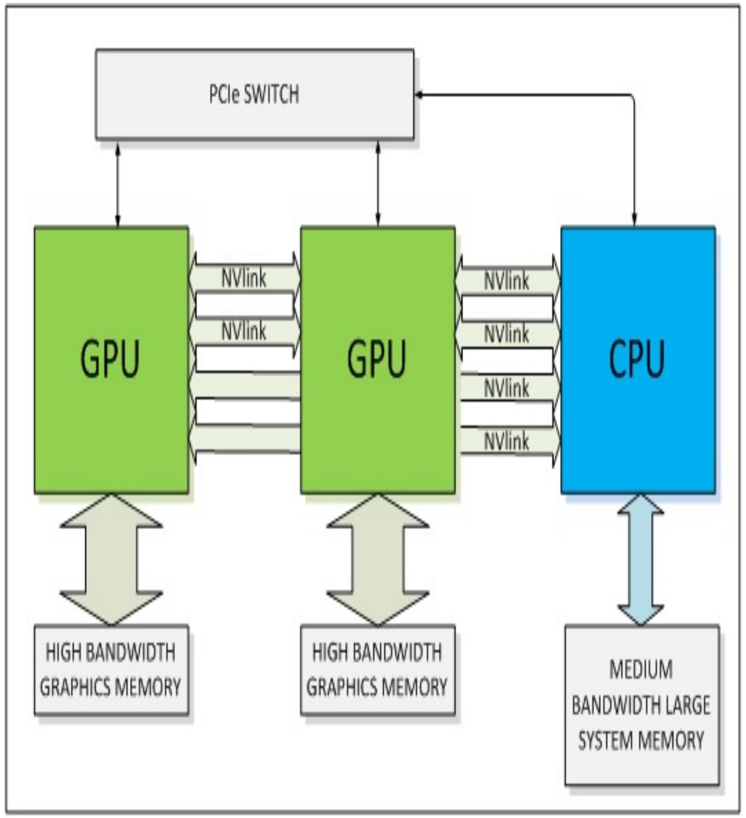
- Low Interconnect Bandwidth
- Small GPU memory capacity
- Coarse grain cooperation of CPU and GPU
- **How to access data in main memory from GPU?**

FAST INTERCONNECT

Faster interconnects help to remedy transfer bottleneck issues

NVLink 2.0

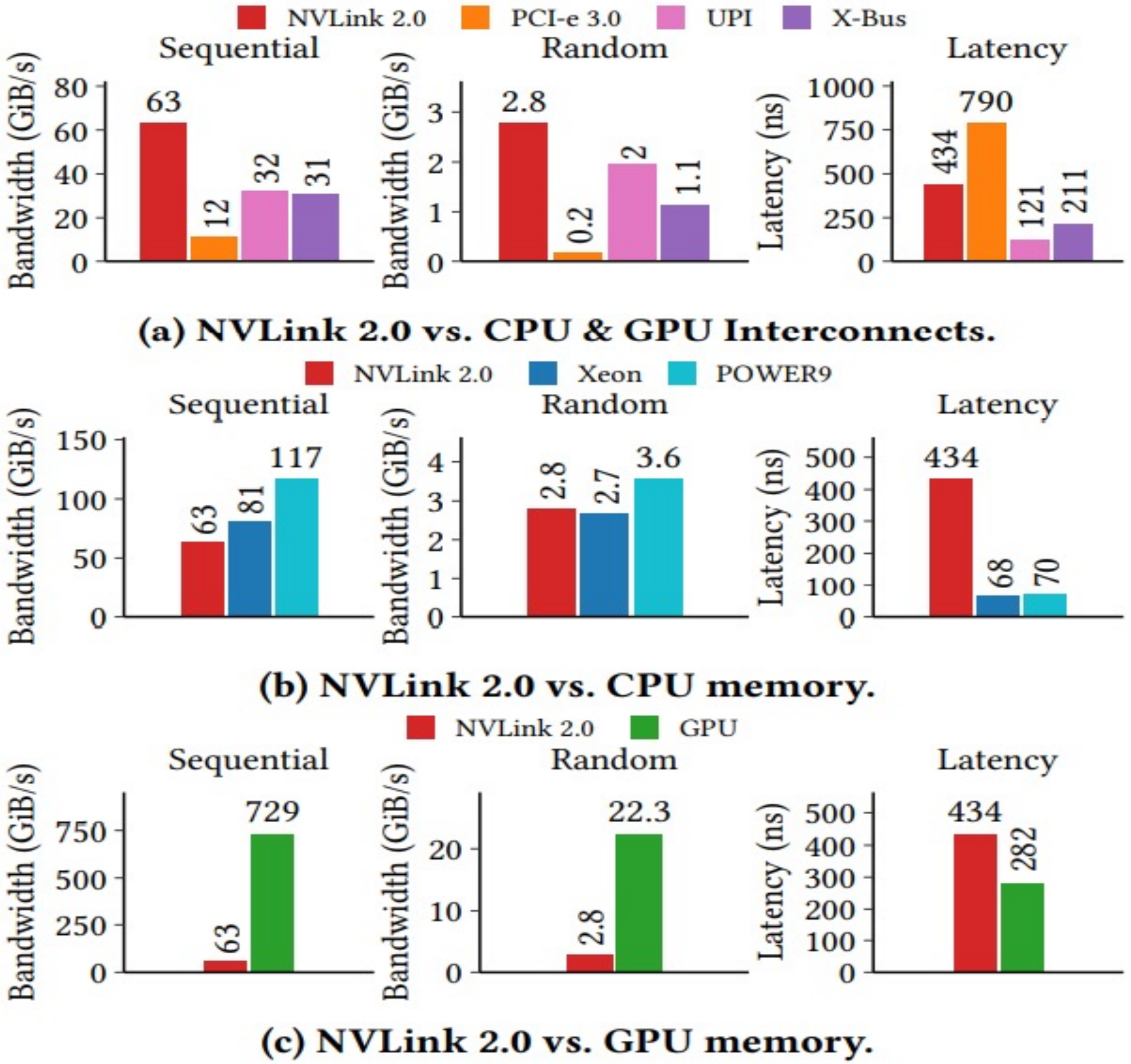




ANALYSIS OF A FAST INTERCONNECT

NVLink 2.0 improves the GPU's interconnect performance

(data transfer)



CHALLENGES DESPITE FAST CONNECTS (NV LINK) FOR QUERY PROCESSING

- Out-of-core GPU join operator must perform both data access and computation efficiently
- Join CPU and GPU requires effective cooperation. Locality and synchronization cost

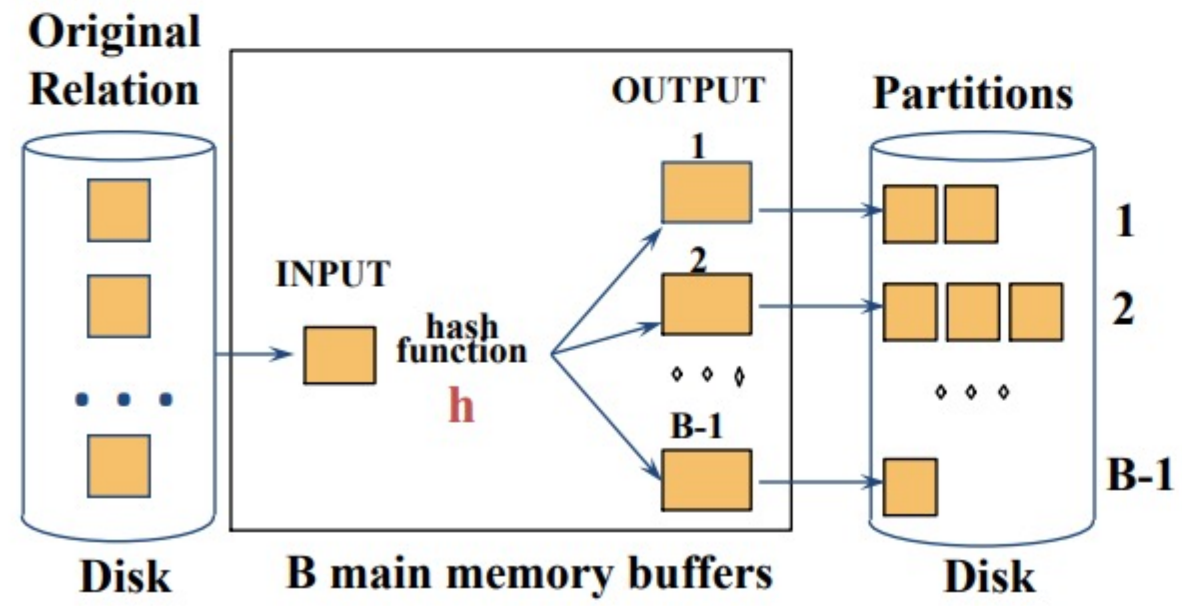
- Increase build side→ increase NP –HJ → spill Hash table to CPU memory → more irregular access to CPU memory (inefficient)

GOAL OF THE PAPER

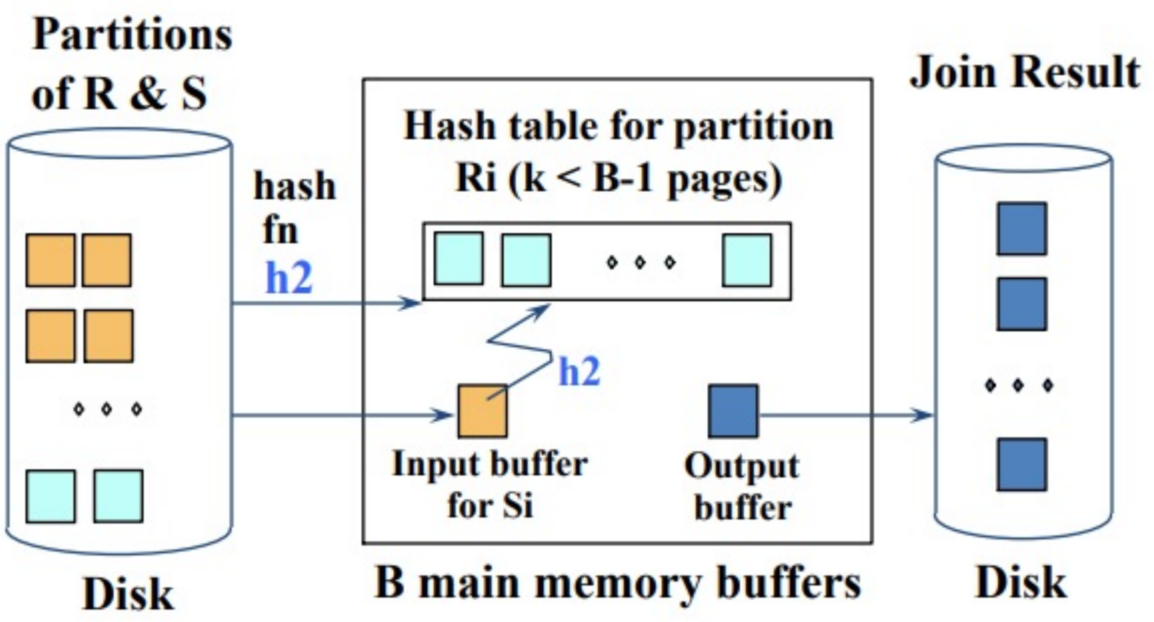
"Scale up GPU-accelerated data management to arbitrary data volumes"

Hash-Join

Partition both relations using hash funtion h: R tuples in partition *i* will only match S tuples in partition *i*



Read in a partition of R, hash it using h2 (<> h!). Scan matching partition of S, probe hash table for matches



NO PARTITION HASH JOIN (R⋈S)

To take advantage of multi core processing

Build

Scan relation R and create a hash table on join key.

Probe

For each tuple in S, look up its join key in hash table for R. If a match is found, output combined tuple.

DATA TRANSFER BETWEEN CPU AND GPU

Push => CPU push data to GPU

Pull => GPU pull data from CPU

Table 1: An overview of GPU transfer methods.

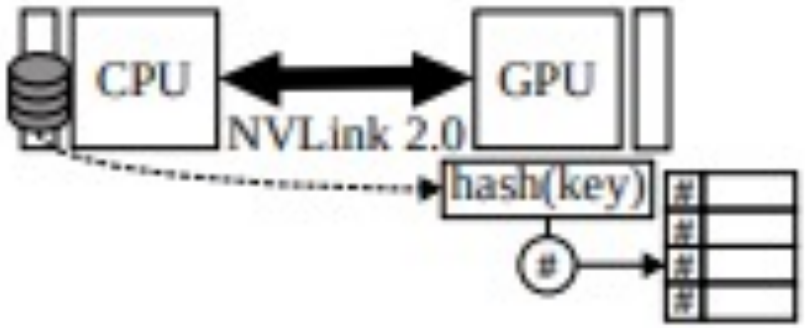
Method	Semantics	Level	Granularity	Memory
Pageable Copy				
Staged Copy				Pageable
Dynamic Pinning	Push	SW	Chunk	
Pinned Copy				Pinned
UM Prefetch				Unified
UM Migration		OS	Page	Unified
Zero-Copy	Pull	HW	Byte	Pinned
Coherence				Pageable

Coherence : GPU can directly access any CPU memory during execution (because of NVLink)

SCALING GPU HASH JOIN : SCALING PROBE SIZE

1. Build hash table on GPU by pulling R tuples on demand from CPU

2. Using Coherence transfer



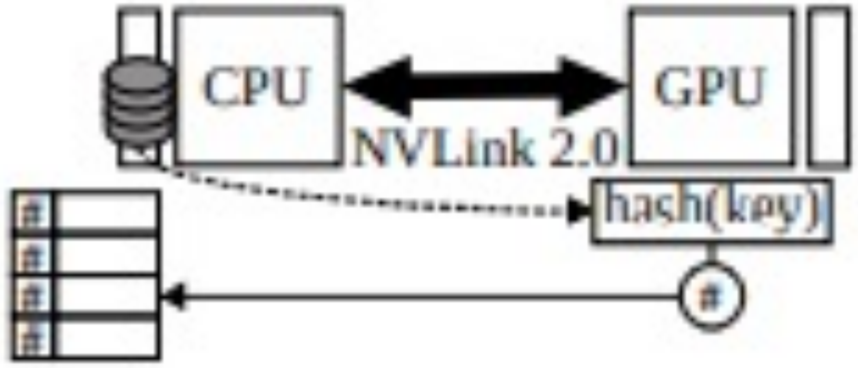
(b) Data in CPU memory and hash table in GPU memory.

Baseline: data is copied into GPU memory to build hash table

SCALING GPU HASH JOIN
: SCALING BUILD SIZE

1. Hash Table is stored in CPU memory

2. No longer constrained by the GPU's memory capacity



(a) Data and hash table in CPU memory.

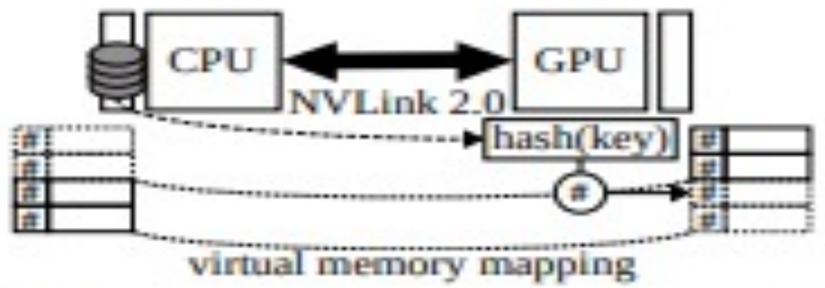
SCALING GPU HASH JOIN : OPTIMIZE HASH TABLE PLACEMENT

Since GPU is much faster than CPU:

1. Place Hash Table on GPU memory and then spill to CPU memory

2. It is done by using Hybrid Hash table

3. Hybrid hash table uses virtual memory to abstract the physical location of memory page



(b) Data in CPU memory and hash table spills from GPU memory into CPU memory.

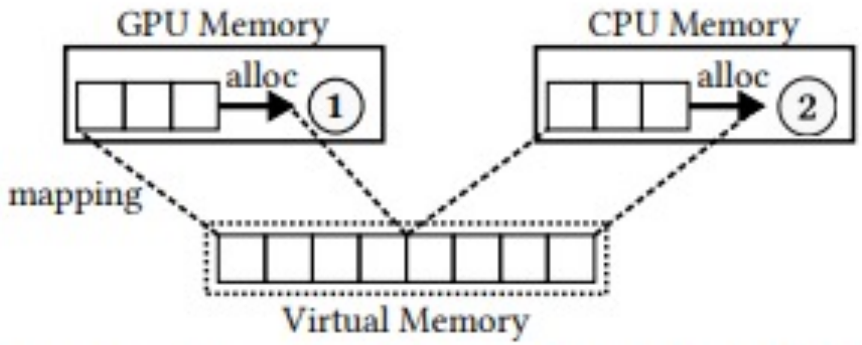


Figure 8: Allocating the hybrid hash table.

SCALING-UP USING CPU AND GPU : TASK SCHEDULING

To solve **load imbalance** issue

1. Adapt the CPU-oriented, **morsel-driven** approach

2. Give each processor the right amount of work to minimize execution skew by considering the increased latency of scheduling work on a GPU, and the higher processing rate of the GPU

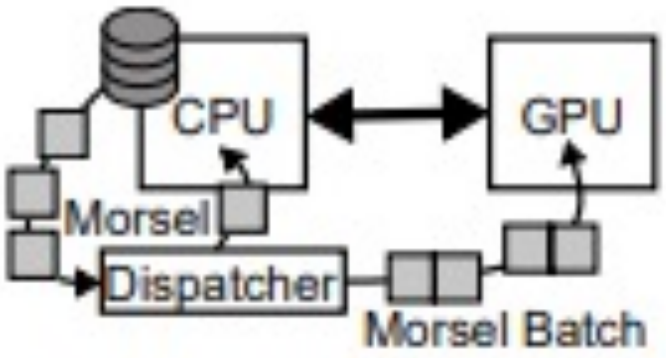
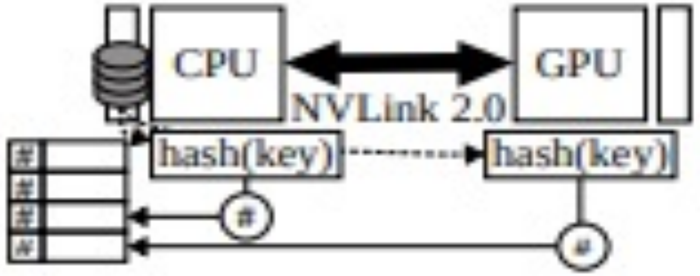


Figure 10: Dynamically scheduling tasks to CPU and GPU processors.

SCALING-UP USING CPU AND GPU : HETEROGENEOUS HASH TABLE PLACEMENT

A. CPU and GPU processing a join using a globally shared hash table (Het strategy) Same as scaling build size



(a) Cooperatively process join on CPU and GPU with hash table in CPU memory.

SCALING-UP USING CPU AND GPU : HETEROGENEOUS HASH TABLE PLACEMENT

Processors are fastest when accessing their local memories

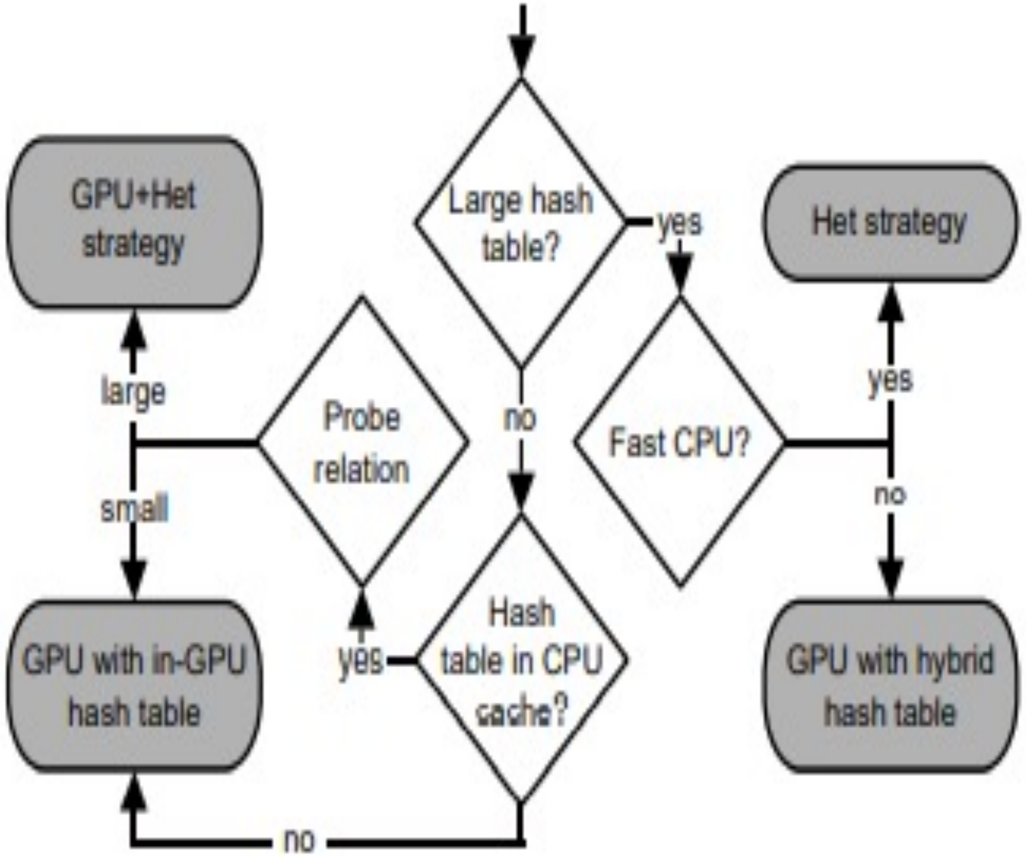


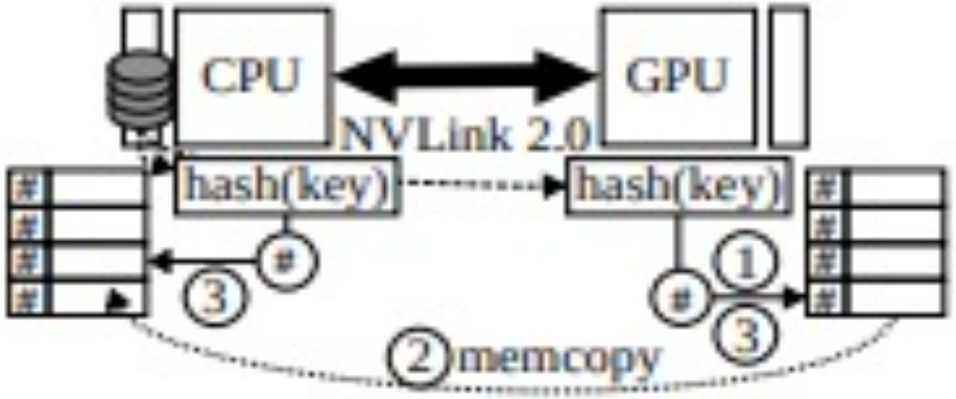
Figure 11: Hash table placement decision.

SCALING-UP USING CPU AND GPU : HETEROGENEOUS HASH TABLE PLACEMENT (WHEN HASH IS SMALL)

1. GPU build hash table in local memory

2. Copy to all other processors

3. Execute the probe phase on all processors using our heterogeneous scheduling strategy.



(b) Build hash table on GPU, copy the hash table to processor-local memories, and then cooperatively probe on CPU and GPU.

MULTI GPU HASH TABLE PLACEMENT

Advantages of multi-GPU

1. Using only GPUs avoids computational skew

2. Distributing large hash tables within GPU memory frees CPU memory bandwidth for loading the base relations

3. interleaving the hash table over multiple GPUs utilizes the full bi-directional bandwidth of fast interconnects, as opposed to the mostly uni-directional traffic of the Het strategy

EXPERIMENT: WORKLOADS

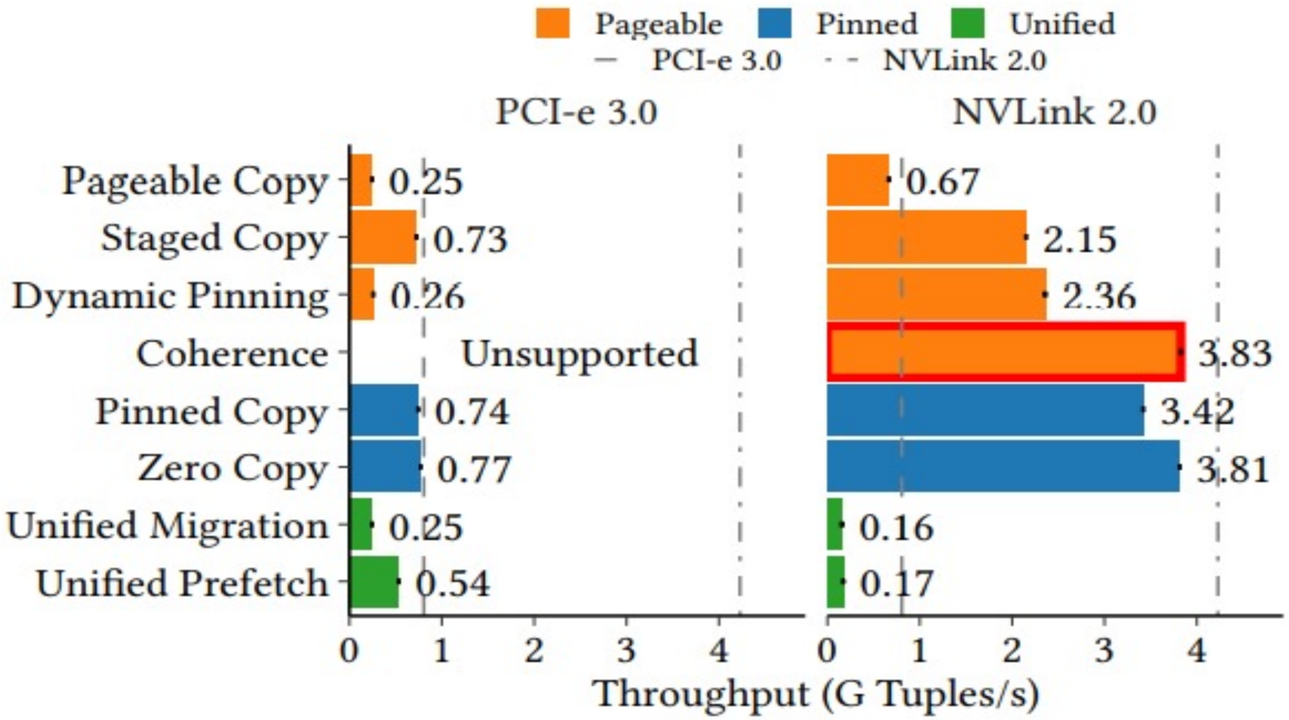
Table 2: Workload Overview.

Property	A (from [10])	B	C (from [54])
key / payload	8 / 8 bytes	8 / 8 bytes	4 / 4 bytes
cardinality of R	2^{27} tuples	2^{18} tuples	$1024 \cdot 10^6$ tuples
cardinality of S	2^{31} tuples	2^{31} tuples	$1024 \cdot 10^6$ tuples
total size of R	2 GiB	4 MiB	7.6 GiB
total size of S	32 GiB	32 GiB	7.6 GiB

EXPERIMENT RESULT (NVLINK VS OTHERS)

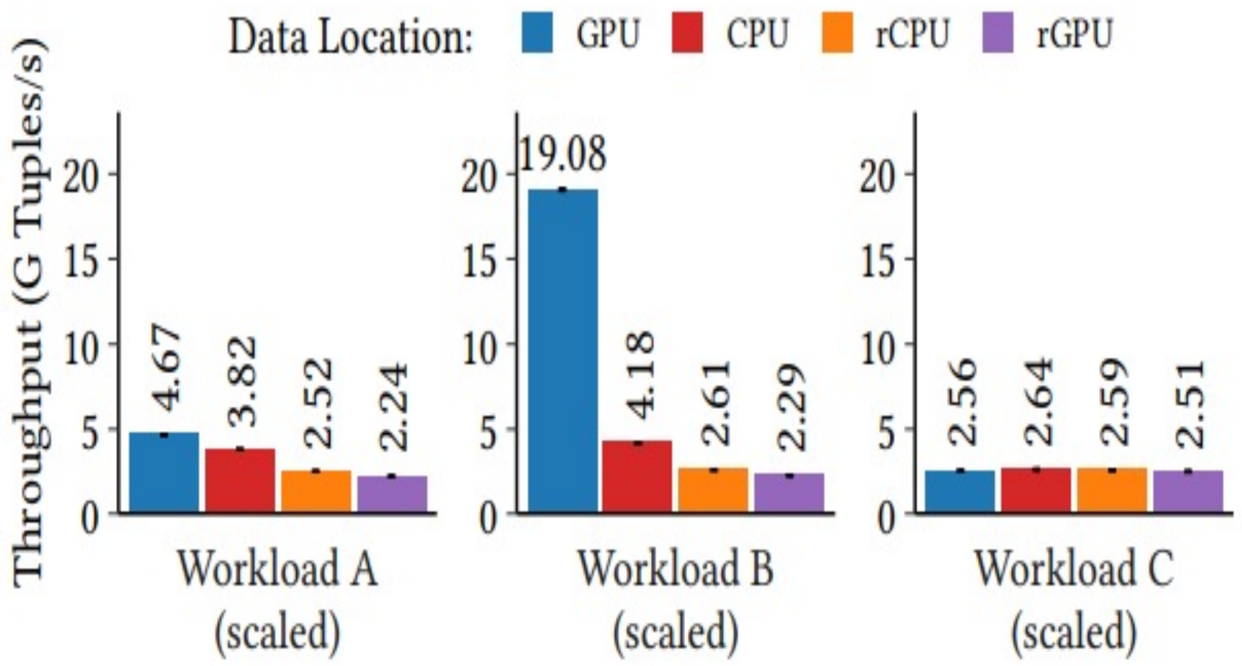
NVLink throughput is higher than PCI-e 3.0

Coherence produces the highest throughput



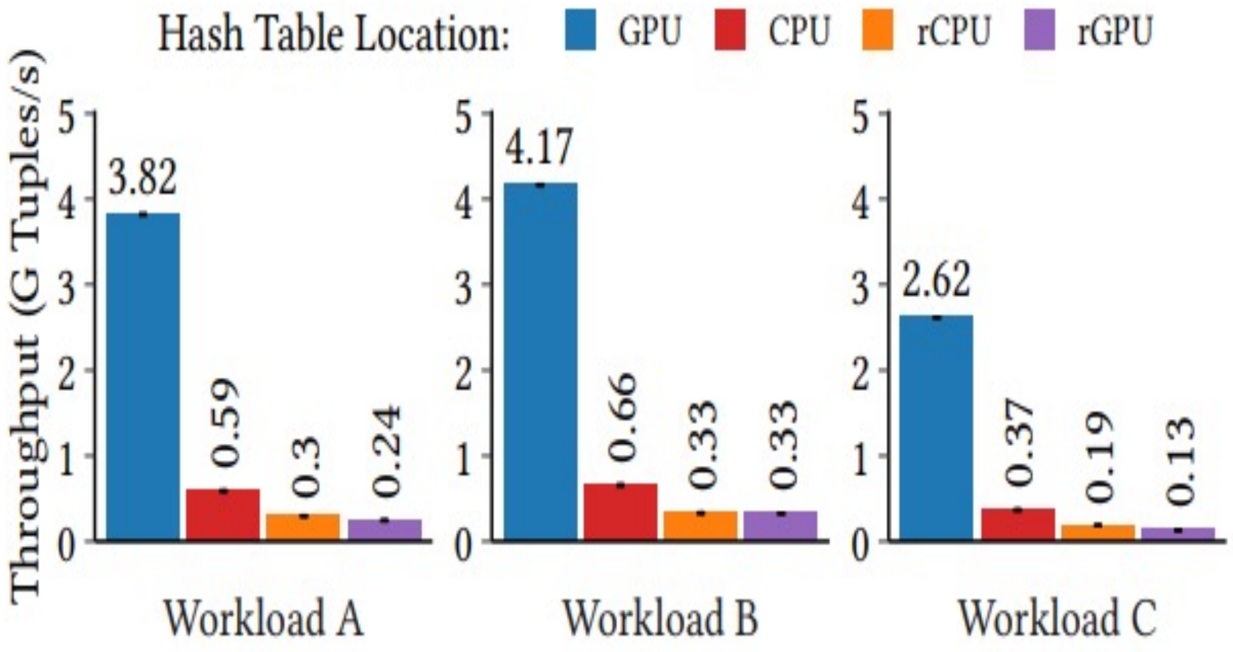
EXPERIMENT RESULT (DATA LOCATION)

Performance best when data in 1 GPU memory



EXPERIMENT RESULT (HASHTABLE LOCATION)

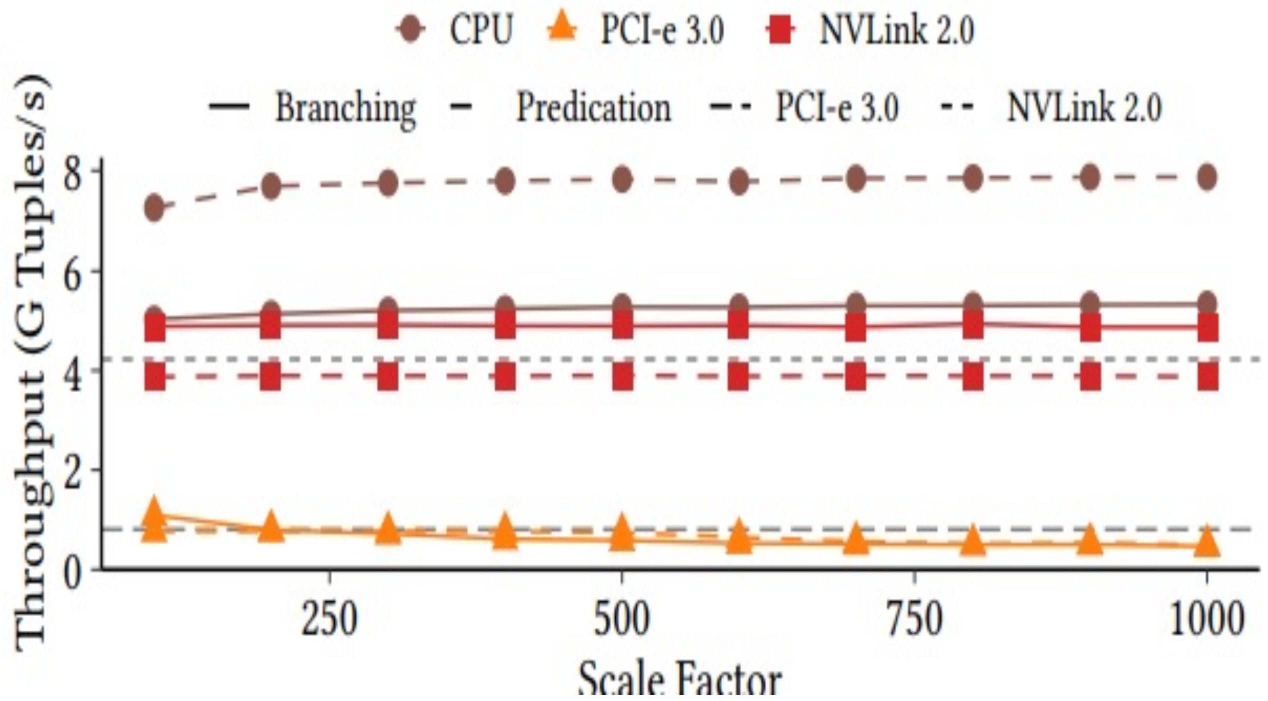
Performance best when hash table in 1 GPU memory



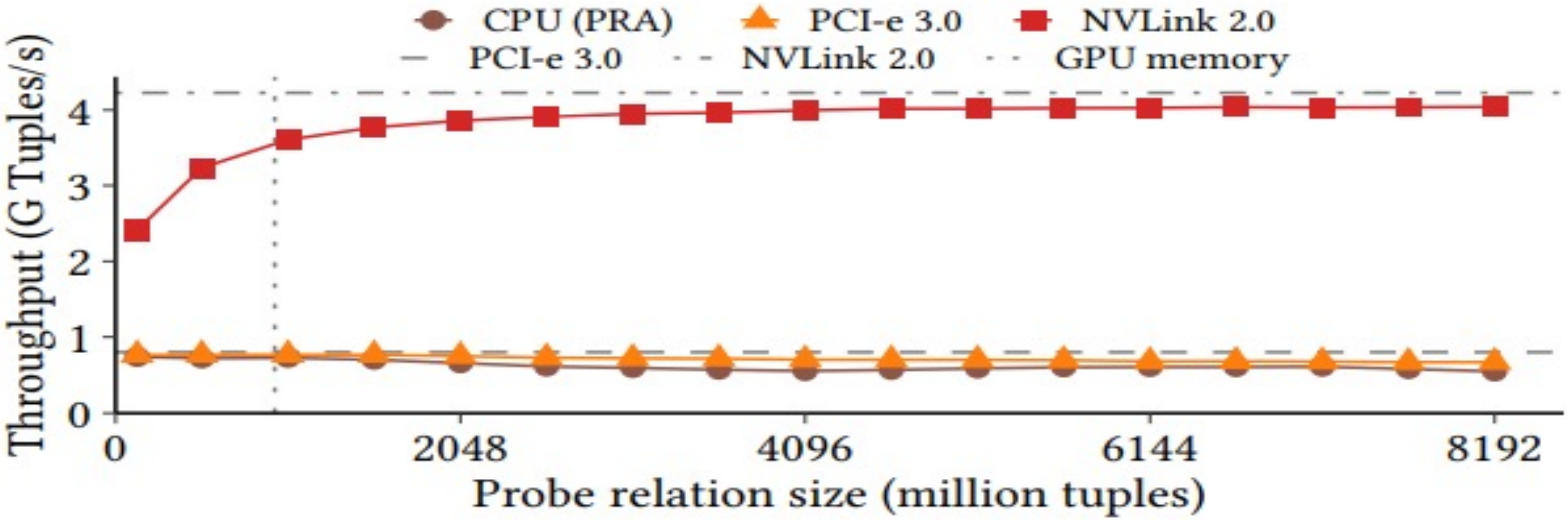
EXPERIMENT RESULT (SCALING DATA SIZE)

Interconnects. The CPU achieves the highest throughput, and outperforms NVLink 2.0

Branching vs. Predication. Branching performs better than predication on the GPU with NVLink 2.0.



EXPERIMENT RESULT (SCALING PROBE SIZE)



The throughput of NVLink 2.0 is the fastest

EXPERIMENT RESULT (SCALING BUILD SIZE)

NV Link provides best through put

NVLink 2.0 with Hybrid Hash Table degrades gracefully

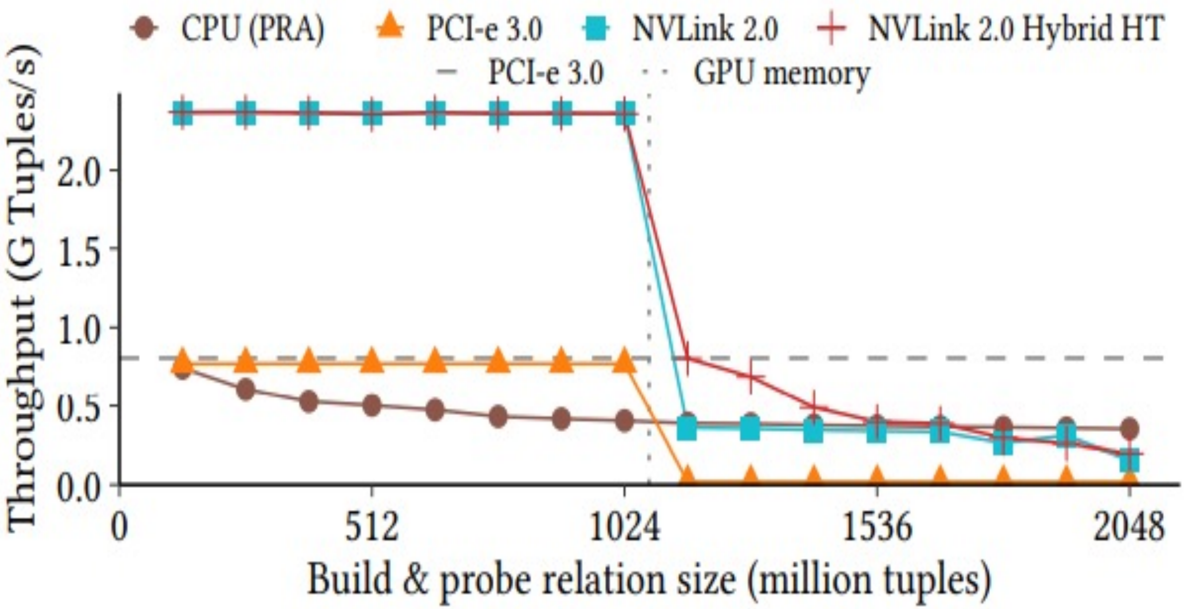


Figure 17: Scaling the build-side relation.

EXPERIMENT RESULT (BUILD TO PROBE RATIO)

The build phase takes 71% of the time

For larger ratios, the build-side takes up a smaller proportion of time

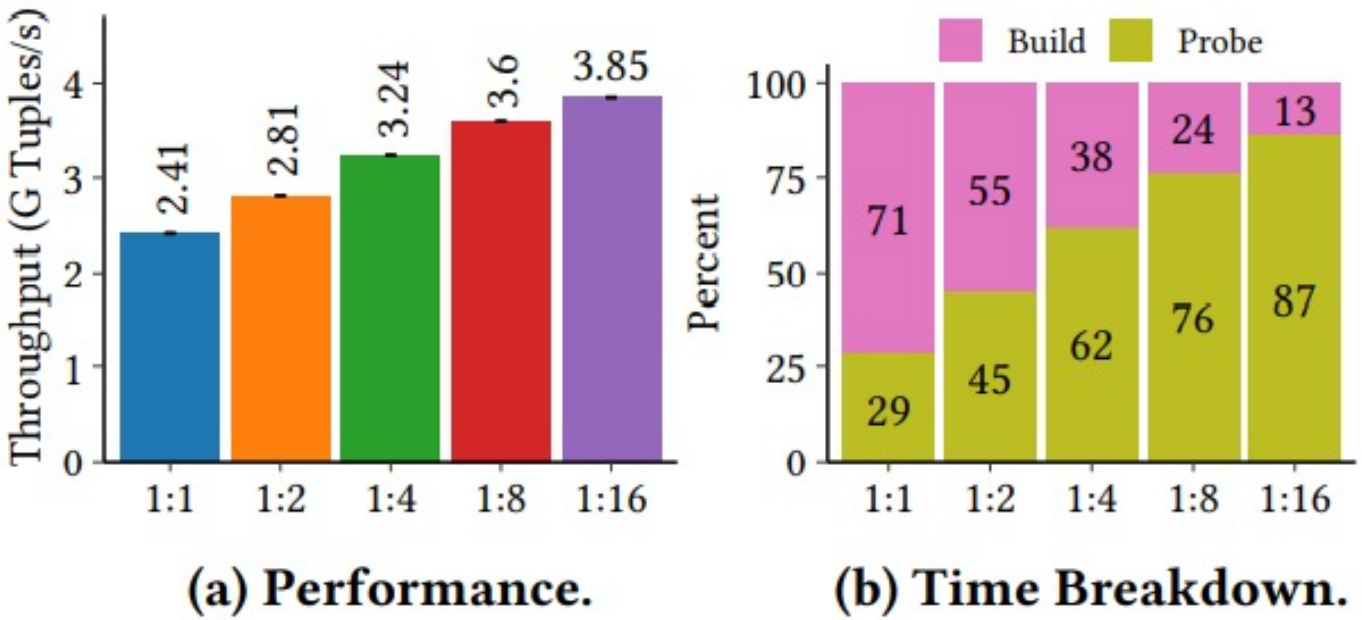


Figure 18: Different build-to-probe ratios on NVLink.

EXPERIMENT RESULT (BUILD DATA SKEW)

Higher skew leads to a higher throughput

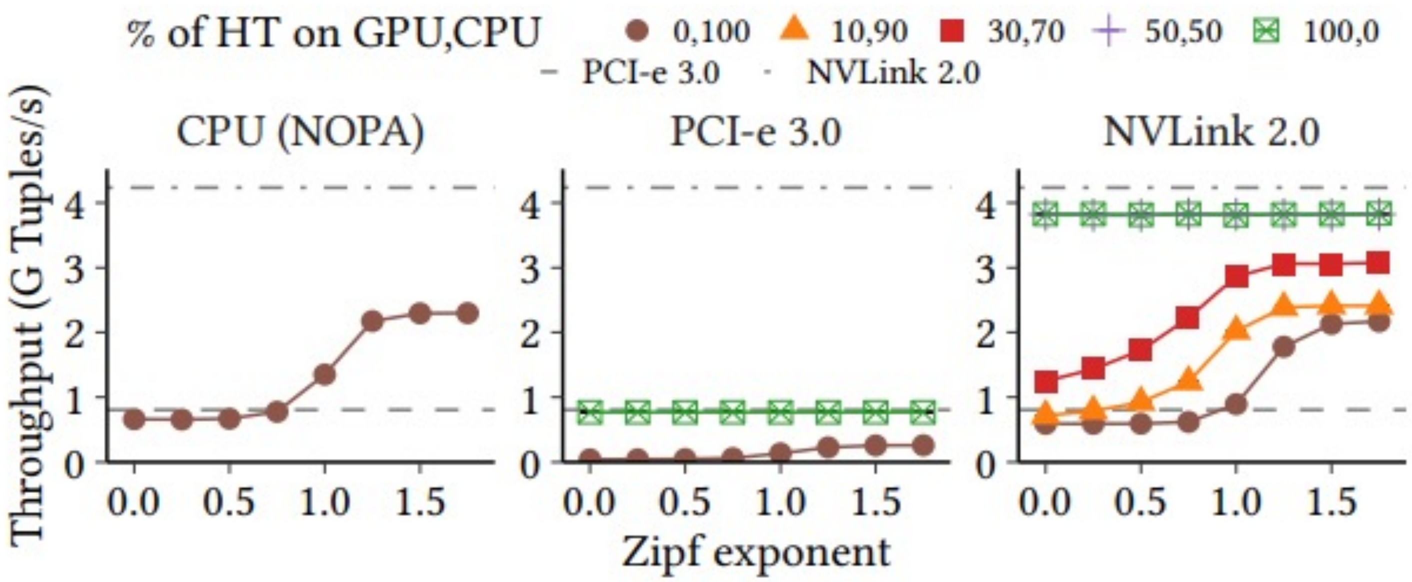


Figure 19: Join performance when the probe relation follows a Zipf distribution.

EXPERIMENT RESULT (JOIN SELECTIVITY)

Join throughput decreases with higher selectivity

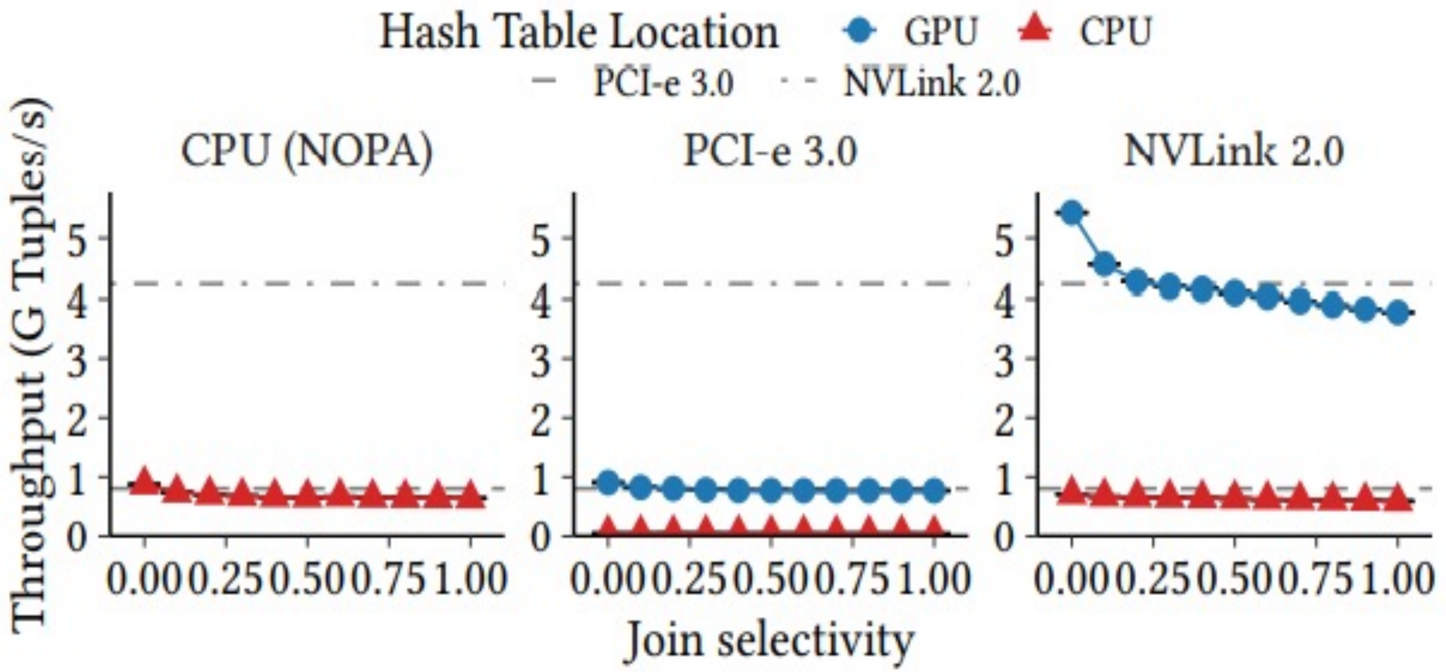


Figure 20: The effect of join selectivity on throughput.

EXPERIMENT RESULT (CPU GPU CO PROCESSING SCALE UP)

1. Using a GPU always achieves the same or better throughput than the CPU-only strategy, and never decreases throughput.

2. GPU-only strategy achieves the best throughput for most of our workloads.

(a) Performance.

(b) Time per join phase in workload C (scaled).

INSIGHTS

- GPUs have high-bandwidth access to CPU memory
- GPUs can efficiently process large, out-of-core data
- GPUs are able to operate on out-of-core data structures, but should use GPU memory if possible
- Scaling-up co-processors with CPU + GPU makes performance more robust.
- Due to cache-coherence, memory pinning is no longer necessary to achieve high transfer bandwidth.
- Fair performance comparisons between GPUs vs. CPUs have become practical.

CONCLUSION

With fast interconnects, GPU acceleration becomes an attractive scale-up alternative that promises large speedups for databases.

THANK YOU