S9557 EFFECTIVE, SCALABLE MULTI-GPU JOINS

Tim Kaldewey, Nikolay Sakharnykh and Jiri Kraus, March 20th 2019

RECAP JOINS

Joins are implicit in a business question

Business question

Counts the number of orders in a given quarter of a given year in which at least one lineitem was received by the customer later than its committed date. The query lists the count of such orders for each order priority sorted in ascending priority order

SQL

```
select
  o_orderpriority,
  count(o_orderkey) as order_count,
from
  orders
where
  o_orderdate >= date '[DATE]' and
  o_orderdate < date '[DATE]' + interval '3' month and
  exists (select * from lineitem
          where l_orderkey = o_orderkey and
          l_commitdate < l_receiptdate)
group by
  o_orderpriority,
order by
  o_orderpriority;
```

Database Operators

aggregate

predicate (filter)

join
predicate (filter)

aggregate

sort

TPC-H SCHEMA

part (p_)

PARTKEY
NAME
MFGR
CATEGORY
BRAND
…

supplier (s_)

SUPPKEY
NAME
ADDRESS
CITY
NATIONKEY
…

lineitem (l_)

ORDERKEY
LINENUMBER
PARTKEY
SUPPKEY
COMMITDATE
RECEIPTDATE
…
…

order (o_)

ORDERKEY
CUSTKEY
ORDERDATE
ORDPRIORITY
ORDERSTATUS
…

customer (c_)

CUSTKEY
NAME
ADDRESS
CITY
…

nation (n_)

NATIONKEY
NAME
…

RELATIONAL JOIN

Lineitem[1]

l_orderkey
23
14
56
11
39
27
23

Foreign Key

⋈

Order[2]

o_orderkey	o_orderpriority
11	1
23	5
27	2
29	4

Primary Key   Payload

=

Join Results

o_orderkey	o_orderpriority
23	5
11	1
27	2
23	5

[1] after applying predicate “l_commitdate < l_receiptdate”
[2] after applying predicates “o_orderdate >= date '[DATE]’ **and** o_orderdate < date '[DATE]' + interval '3' month”

HASH JOIN

Lineitem[1]

l_orderkey
23
14
56
11
39
27
23

Foreign Key
= Probe inputs

⋈

Order[2]

o_orderkey	o_orderpriority
11	1
23	5
27	2
29	4

Primary Key   Payload

Build hash table

=

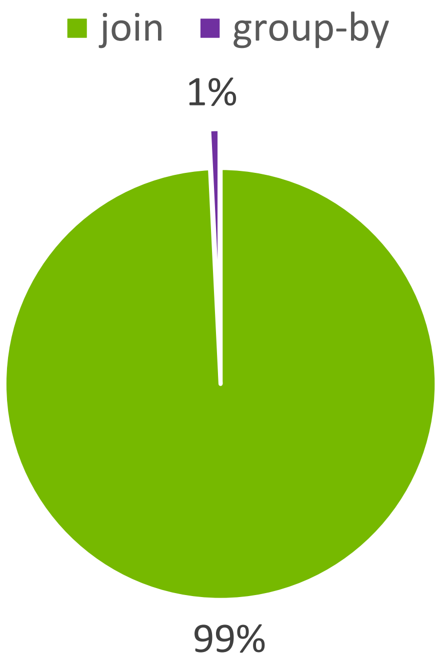
Join Results

o_orderkey	o_orderpriority
23	5
11	1
27	2
23	5

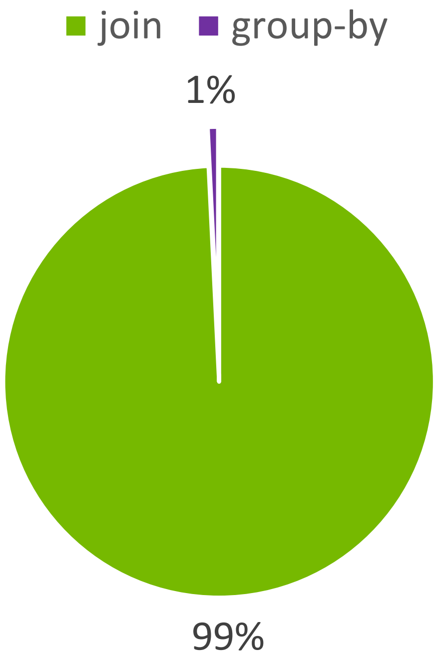
[1] after applying predicate “l_commitdate < l_receiptdate”
[2] after applying predicates “o_orderdate >= date '[DATE]’ **and** o_orderdate < date '[DATE]' + interval '3' month”

JOINS & E2E PERFORMANCE

CPU TPC-H Q4 execution breakdown



GPU TPC-H Q4 execution breakdown

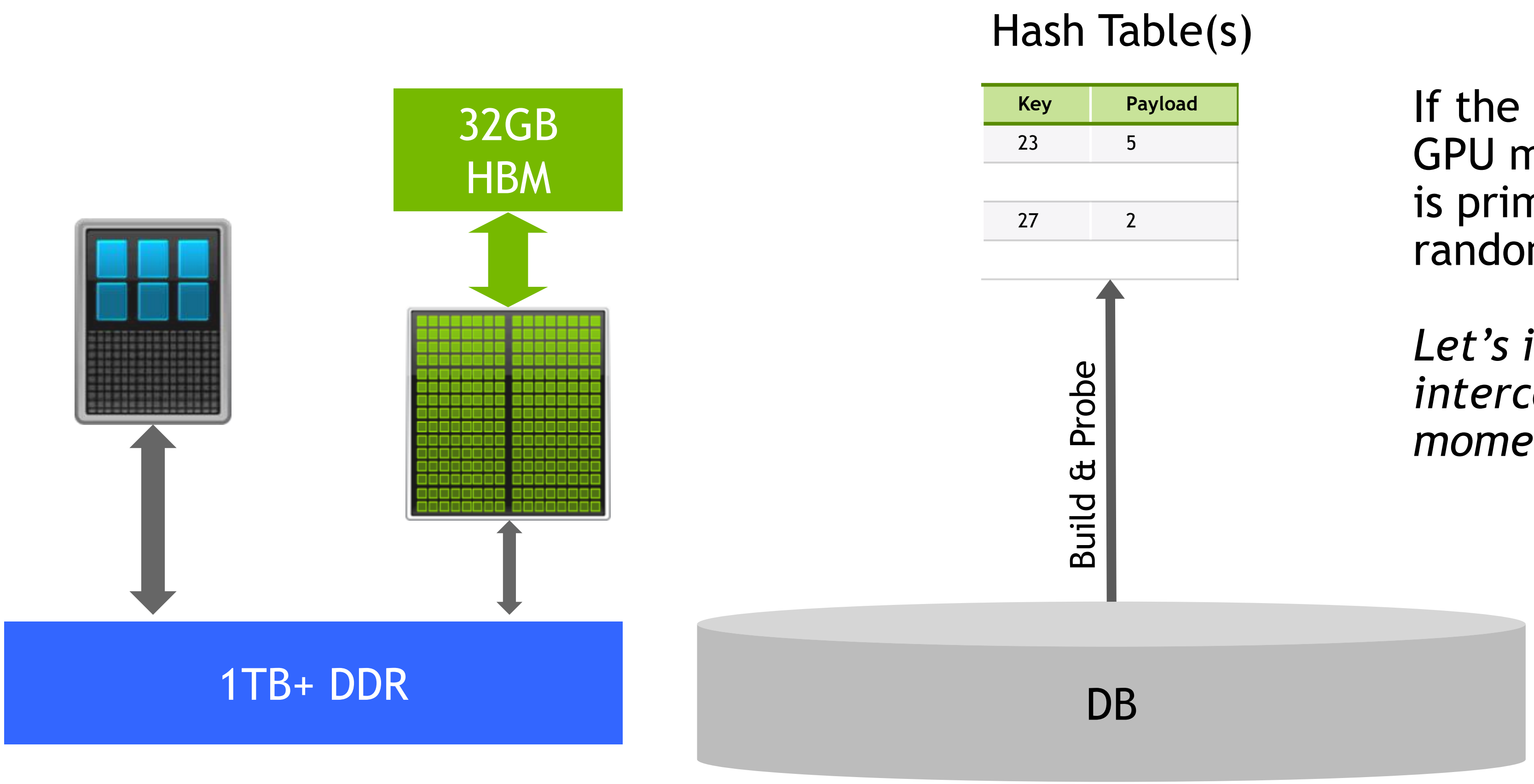


18/22 TPC-H Queries involve **Joins** and are the longest running ones[1]

[1] c.f. recently published TPC-H results at http://www.tpc.org/tpch/results/tpch_last_ten_results.asp

IMPLEMENTING GPU JOINS

In Heterogeneous Systems



If the hash table fits in GPU memory, performance is primarily bound by random memory access.[1]

Let's ignore CPU-GPU interconnect for a moment.

[1] *c.f. "How to Get the Most out of GPU Accelerated Database Operators", GTC Silicon Valley 2018, Session ID S8289*

PERFORMANCE

	Peak memory bandwidth[1]	Random 8B access[1]
High-end CPU (6-channel DDR4)	120 GB/s	6GB/s
NVIDIA Tesla V100	900 GB/s	60GB/s

10x

[1] *c.f. "How to Get the Most out of GPU Accelerated Database Operators", GTC Silicon Valley 2018, Session ID S8289*
http://on-demand-gtc.gputechconf.com/gtc-quicklink/ar9zi75

PERFORMANCE VS. CAPACITY

	Peak memory bandwidth[1]	Random 8B access[1]	Memory capacity
High-end CPU (6-channel DDR4)	120 GB/s	6GB/s	1 TB+
NVIDIA Tesla V100	900 GB/s	60GB/s	32GB

1/32

[1] *c.f.“How to Get the Most out of GPU Accelerated Database Operators”, GTC Silicon Valley 2018, Session ID S8289*
http://on-demand-gtc.gputechconf.com/gtc-quicklink/ar9zi75

PERFORMANCE VS. CAPACITY

	Peak memory bandwidth[1]	Random 8B access[1]	Memory capacity
High-end CPU (6-channel DDR4)	120 GB/s	6GB/s	1 TB+
NVIDIA Tesla V100	900 GB/s	60GB/s	32GB
NVIDIA DGX-2 (16x V100)	16 x 900 GB/s	16x 60GB/s	512 GB

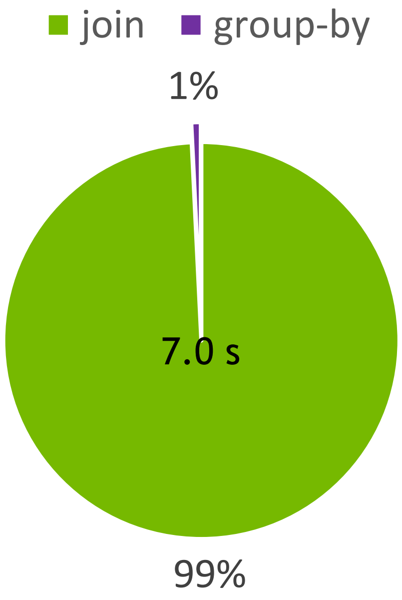
1/2

[1] *c.f."How to Get the Most out of GPU Accelerated Database Operators", GTC Silicon Valley 2018, Session ID S8289*
http://on-demand-gtc.gputechconf.com/gtc-quicklink/ar9zi75

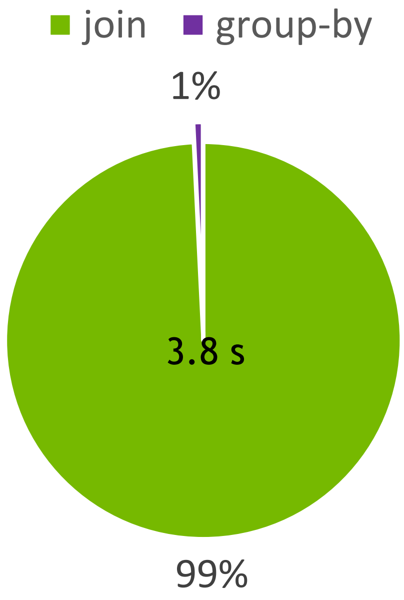
IS A SINGLE V100 FAST/LARGE ENOUGH?

TPC-H query 4 @SF1000 = 1000GB data warehouse

GPU execution breakdown



GPU execution breakdown, compressed data



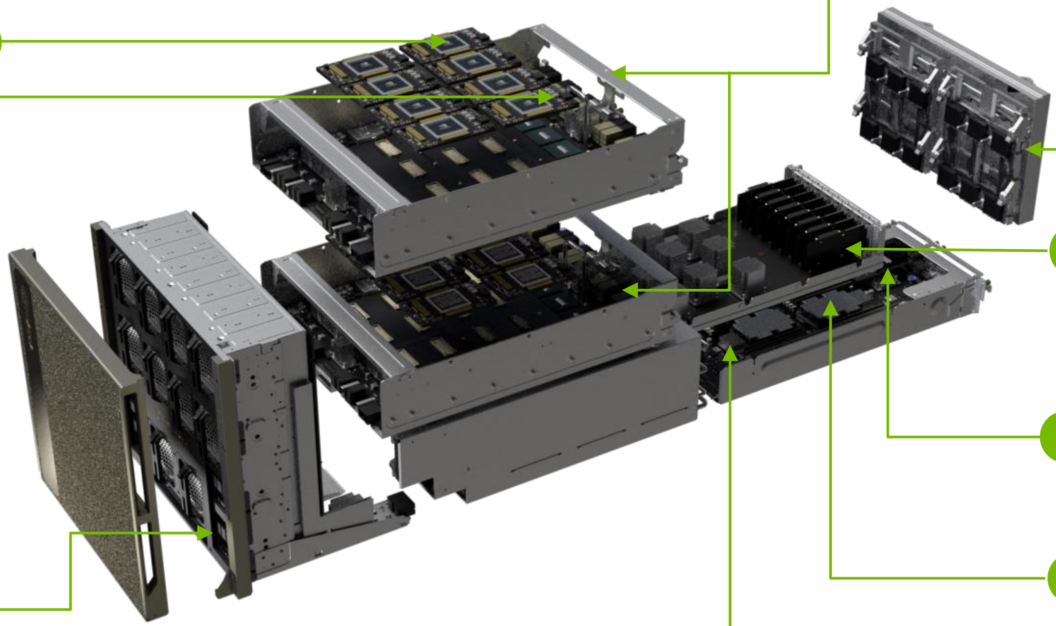
Hash table sizes

Query	SF1K	SF3K	SF10K
Q4	1.5 GB	4.5 GB	15 GB
Q18	21 GB	63 GB	210 GB
Q21	10.5 GB	31.5 GB	105 GB

For further speedup or > SF 1000 need to to distribute hash table across multiple GPUs

DESIGNED TO TRAIN THE PREVIOUSLY IMPOSSIBLE

NVIDIA DGX-2



1 **NVIDIA Tesla V100 32GB**

2 **Two GPU Boards**
8 V100 32GB GPUs per board
6 NVSwitches per board
512GB Total HBM2 Memory
interconnected by
Plane Card

3 **Twelve NVSwitches**
2.4 TB/sec bi-section bandwidth

4 **Eight EDR Infiniband/100 GigE**
1600 Gb/sec Total
Bi-directional Bandwidth

5 **Two Intel Xeon Platinum CPUs**

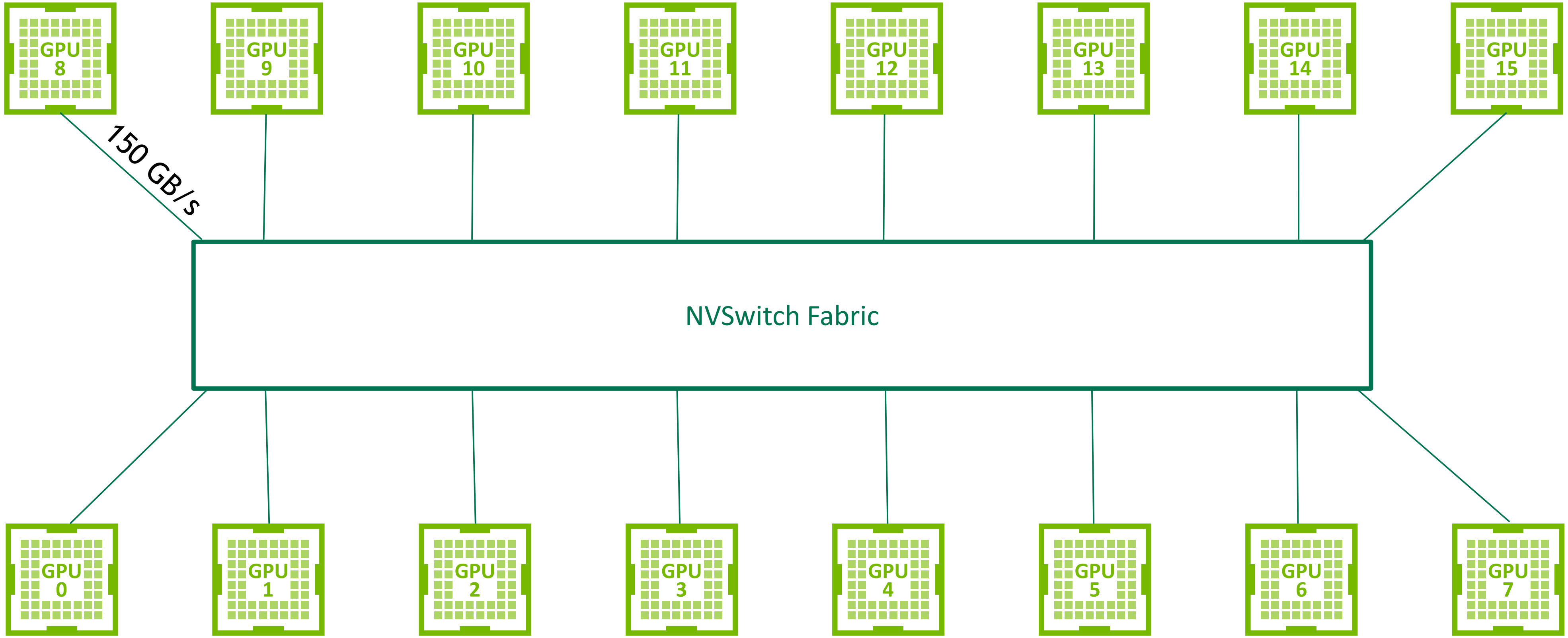
6 **1.5 TB System Memory**

7 **30 TB NVME SSDs**
Internal Storage

8 **Two High-Speed Ethernet**
10/25/40/100 GigE

POTENTIAL DGX-2 IMPLEMENTATION

Use 2.4TB/s bisection BW to exchange FT chunks



SCALING OF INNER JOIN

DISCLAIMER

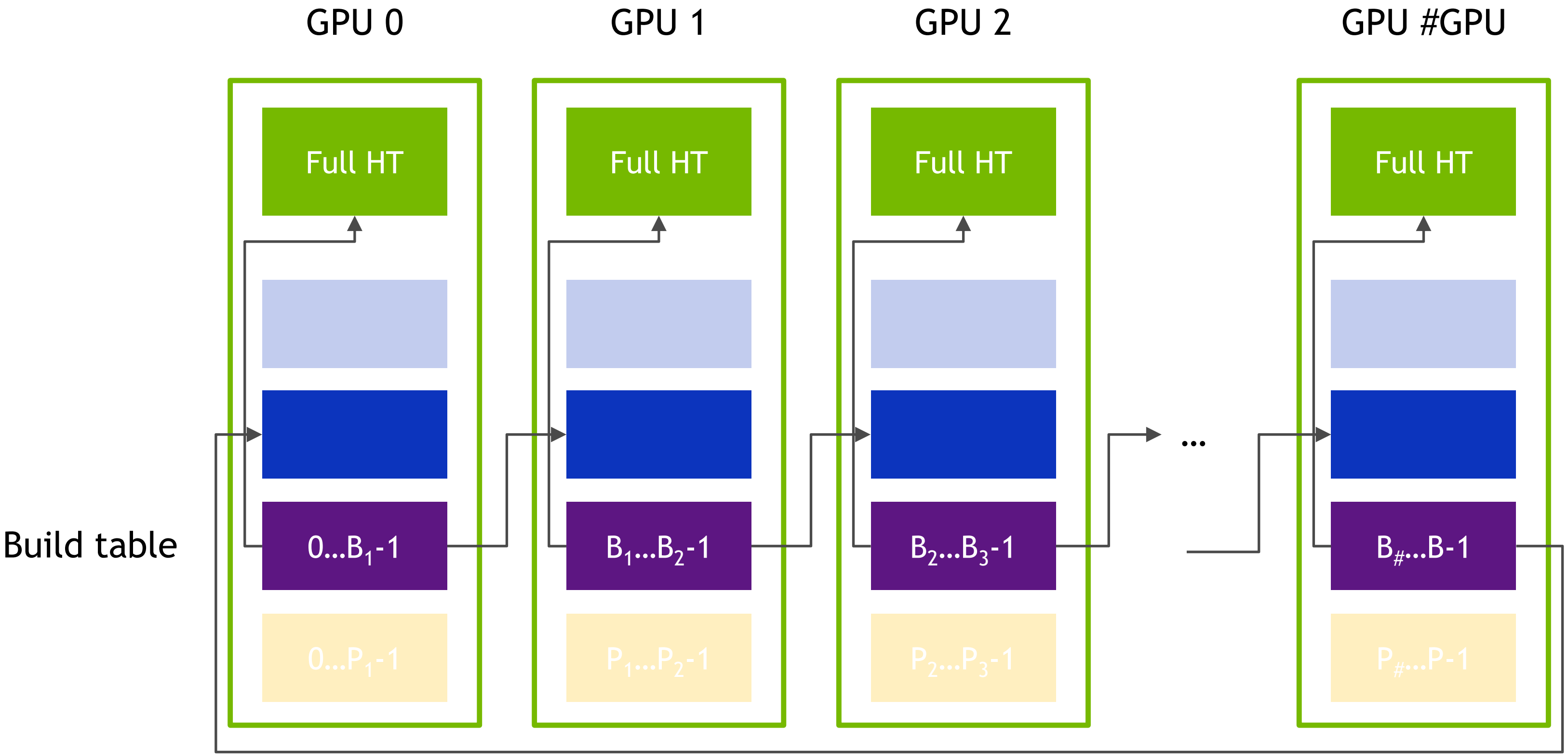
This investigation is ongoing

For a production system some additional aspects need to be considered:

- Data Skew
- Cardinality estimation
- Query optimizer

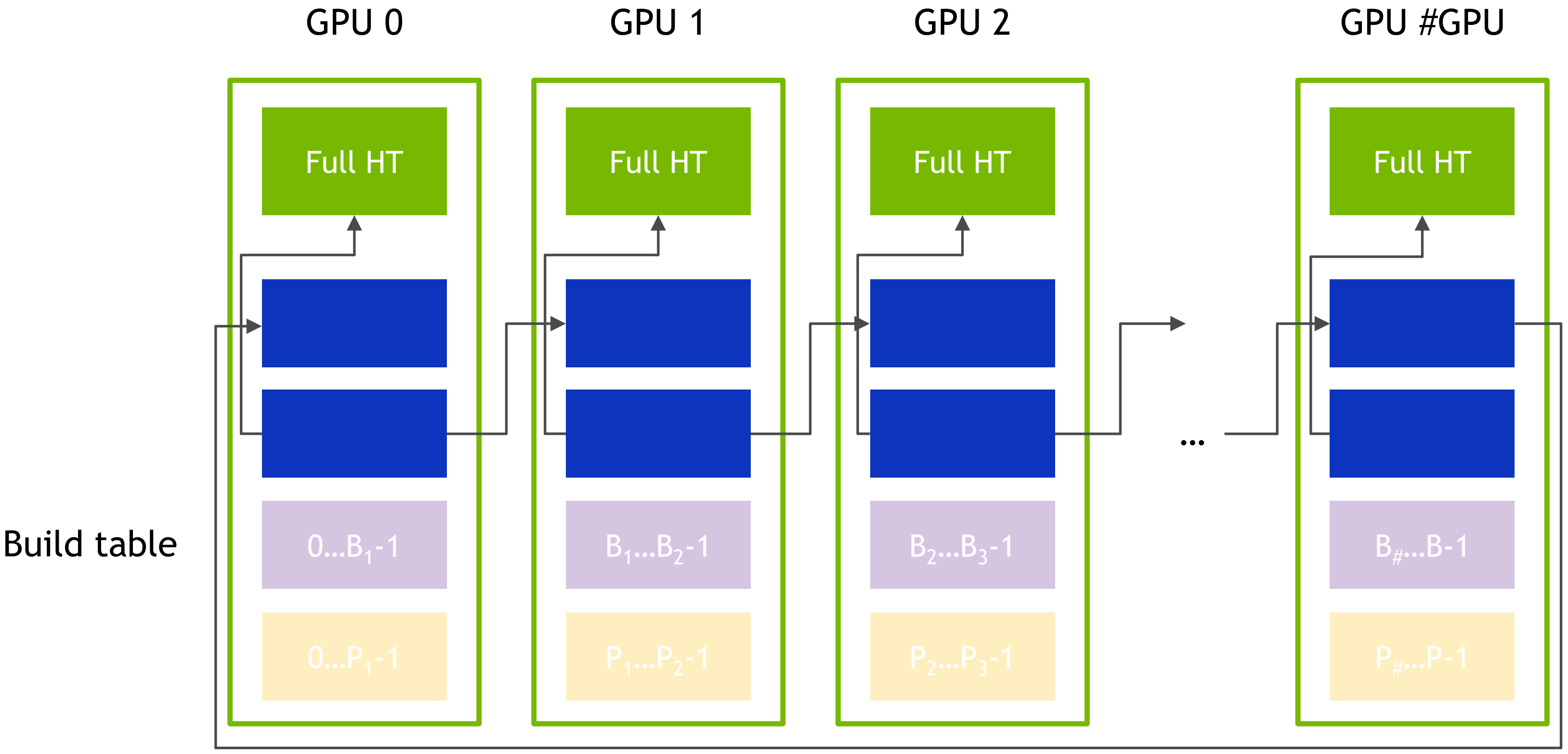
SCALING OF INNER JOIN

redundant build of replicated HT (step 0)



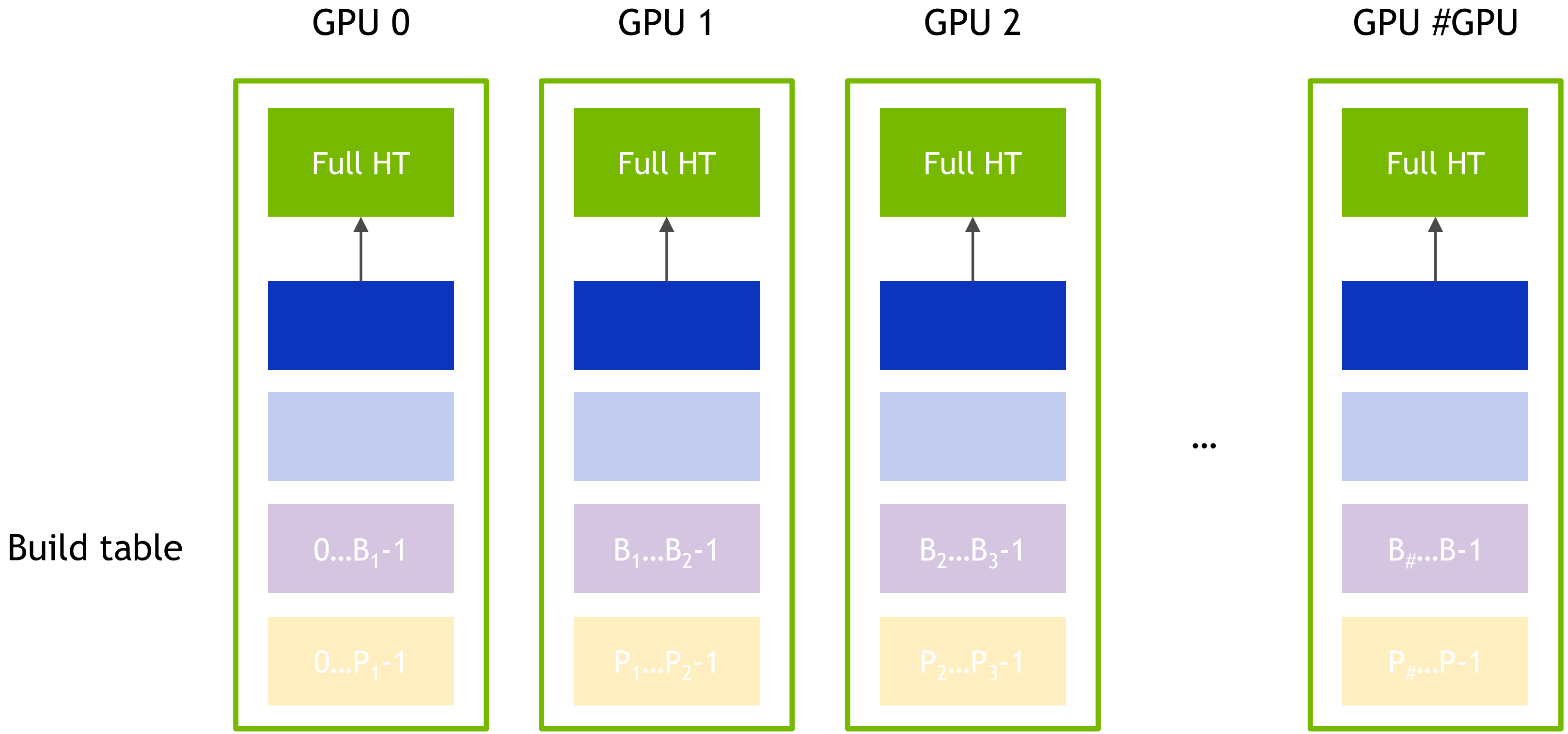
SCALING OF INNER JOIN

redundant build of replicated HT (step 1..#GPU-1)



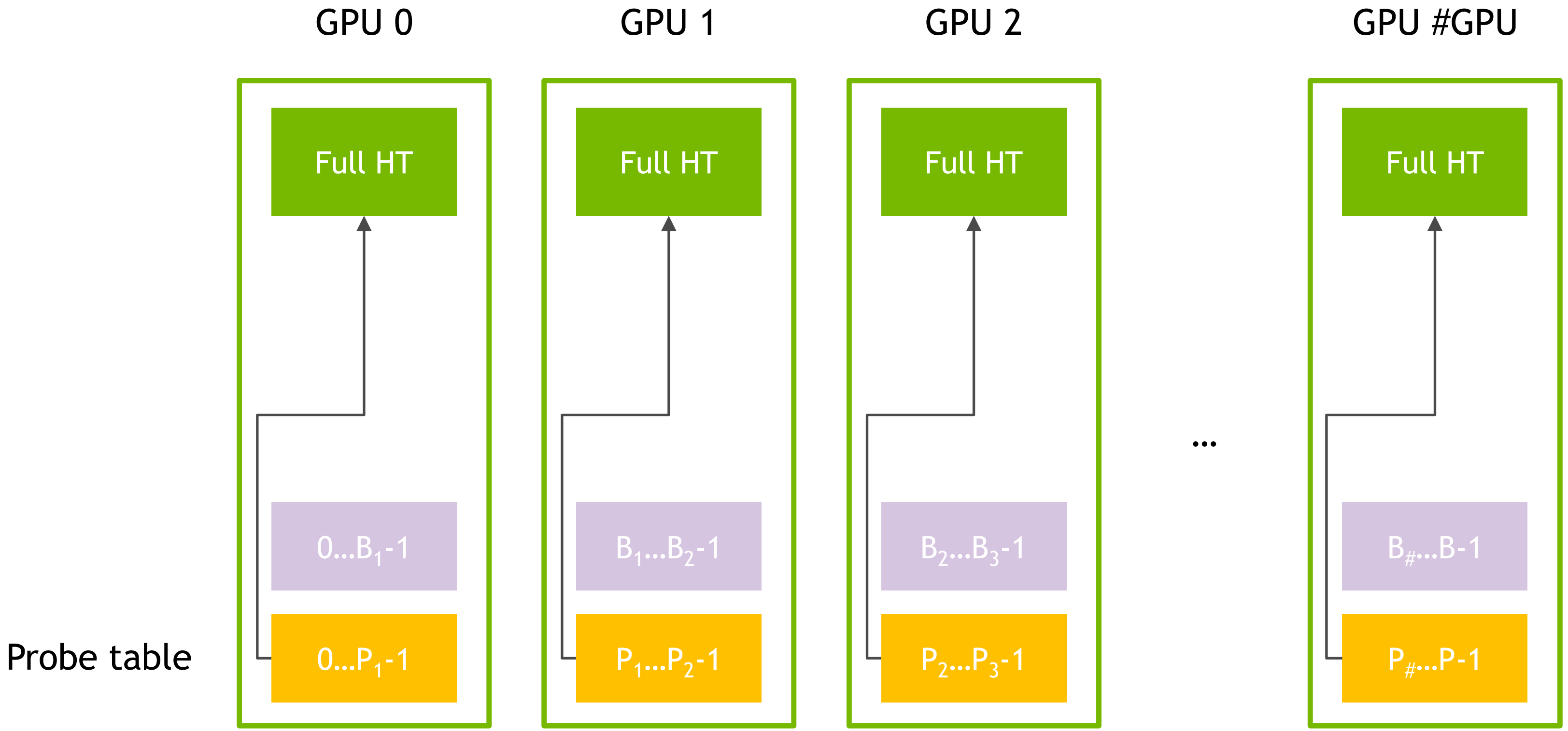
SCALING OF INNER JOIN

redundant build of replicated HT (step #GPU)



SCALING OF INNER JOIN

parallel probe of replicated HT



SCALING OF INNER JOIN

Benchmark Problem

randomly generated 8 bytes keys

build table size = probe table size = 335544320 rows (worst case for HT creation fitting in the memory of a single GPU: 2x 2.5GiB for tables, 2x10GiB for HT + staging buffers (for strong scaling experiment))

HT occupancy = 50%

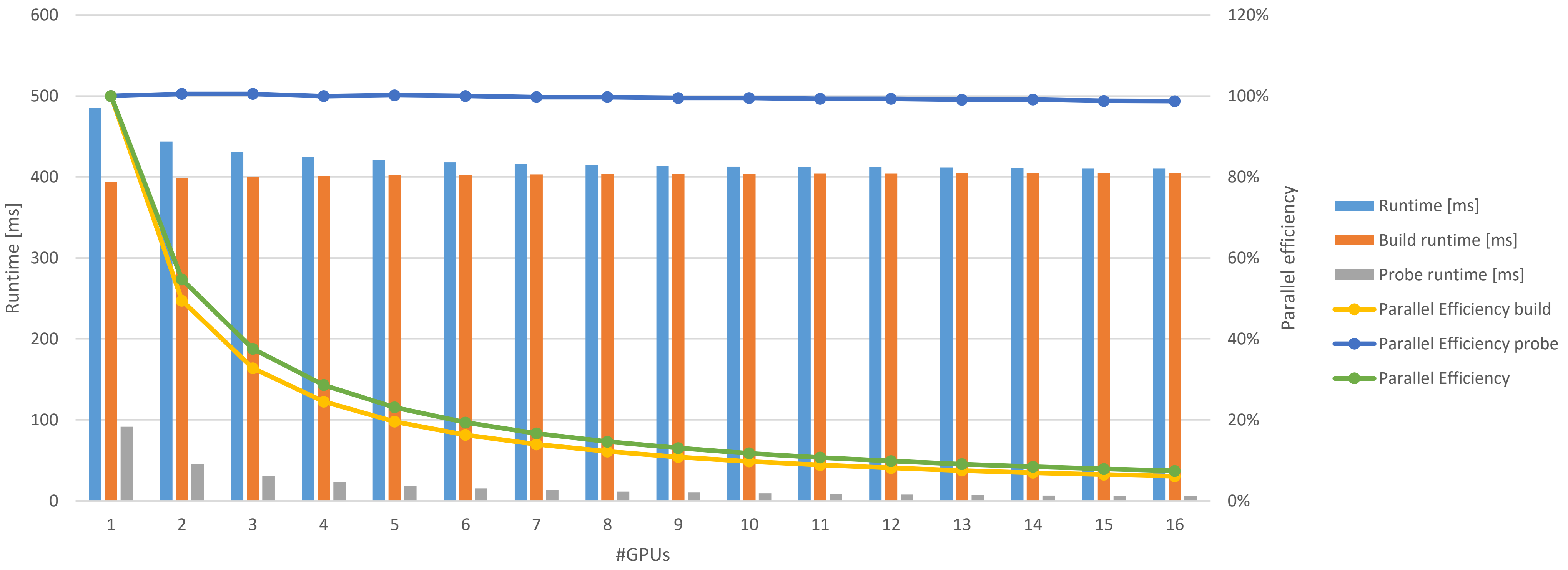
selectivity = 0 for analytical purposes we will look at a real problem later

build and probe tables are evenly partitioned across GPUs

	GPU 0	GPU 1	GPU 2	…	GPU #GPU
Build table	$0 \ldots B_1\text{-}1$	$B_1 \ldots B_2\text{-}1$	$B_2 \ldots B_3\text{-}1$		$B_\# \ldots B\text{-}1$
Probe table	$0 \ldots P_1\text{-}1$	$P_1 \ldots P_2\text{-}1$	$P_2 \ldots P_3\text{-}1$		$P_\# \ldots P\text{-}1$

SCALING OF INNER JOIN ON DGX-2

with redundant build of replicated HT



Runtimes are the minimum of 5 repetitions for probe + build (excluding setup overhead, e.g. allocation of hash tables or temp buffers)

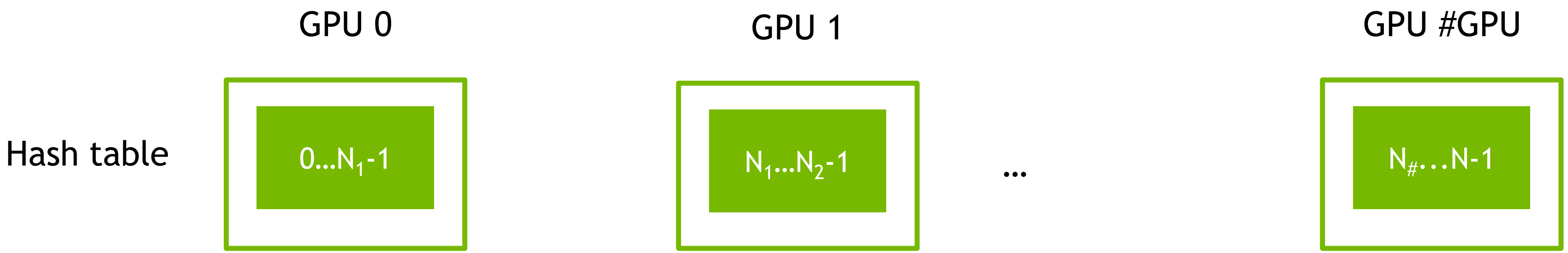
SCALING OF INNER JOIN

Basic Idea

Open addressing hash table with N buckets

key -> hash_value = hf(key) -> bucket_idx = hash_value%N

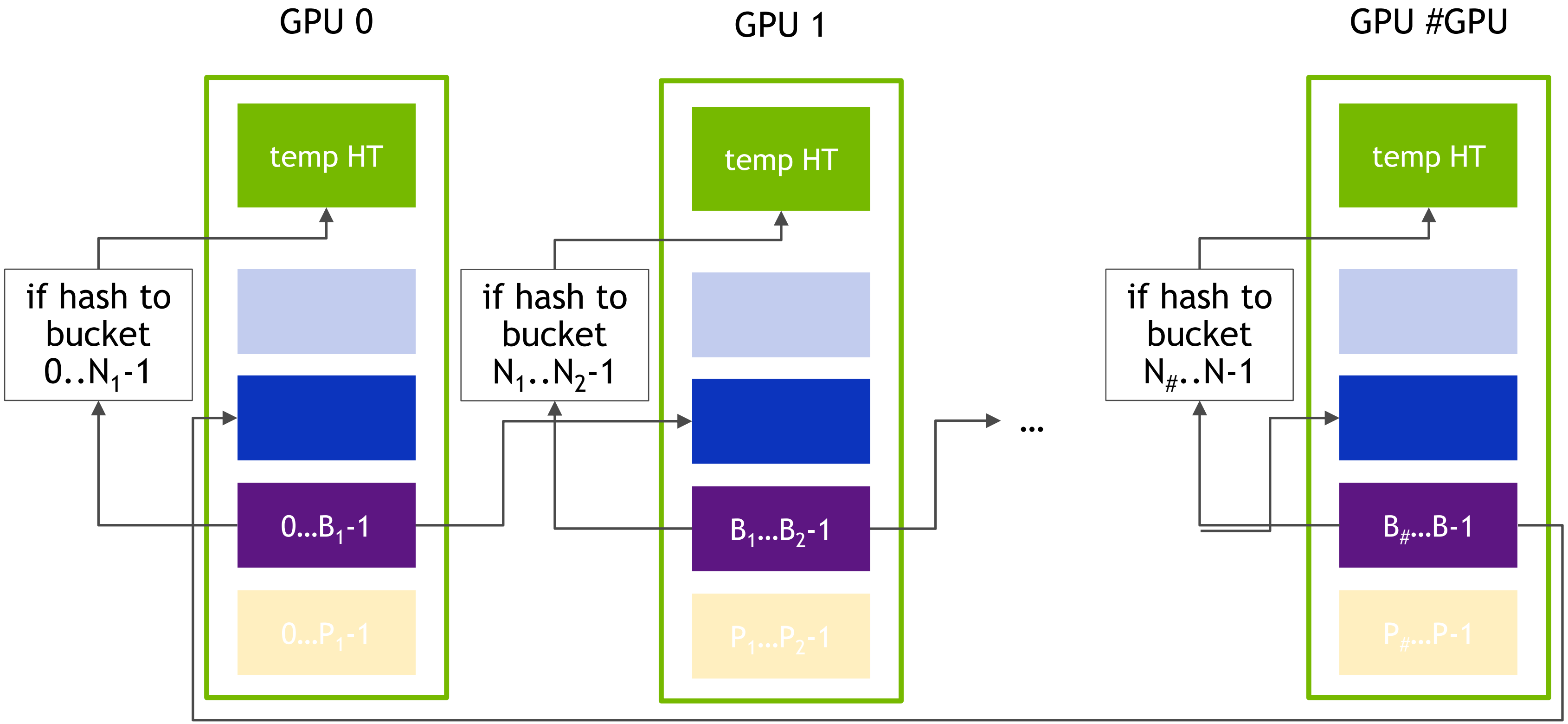
Partition N hash table buckets equally onto GPUs:



The bucket_idx and target HT partition can be computed locally from the key

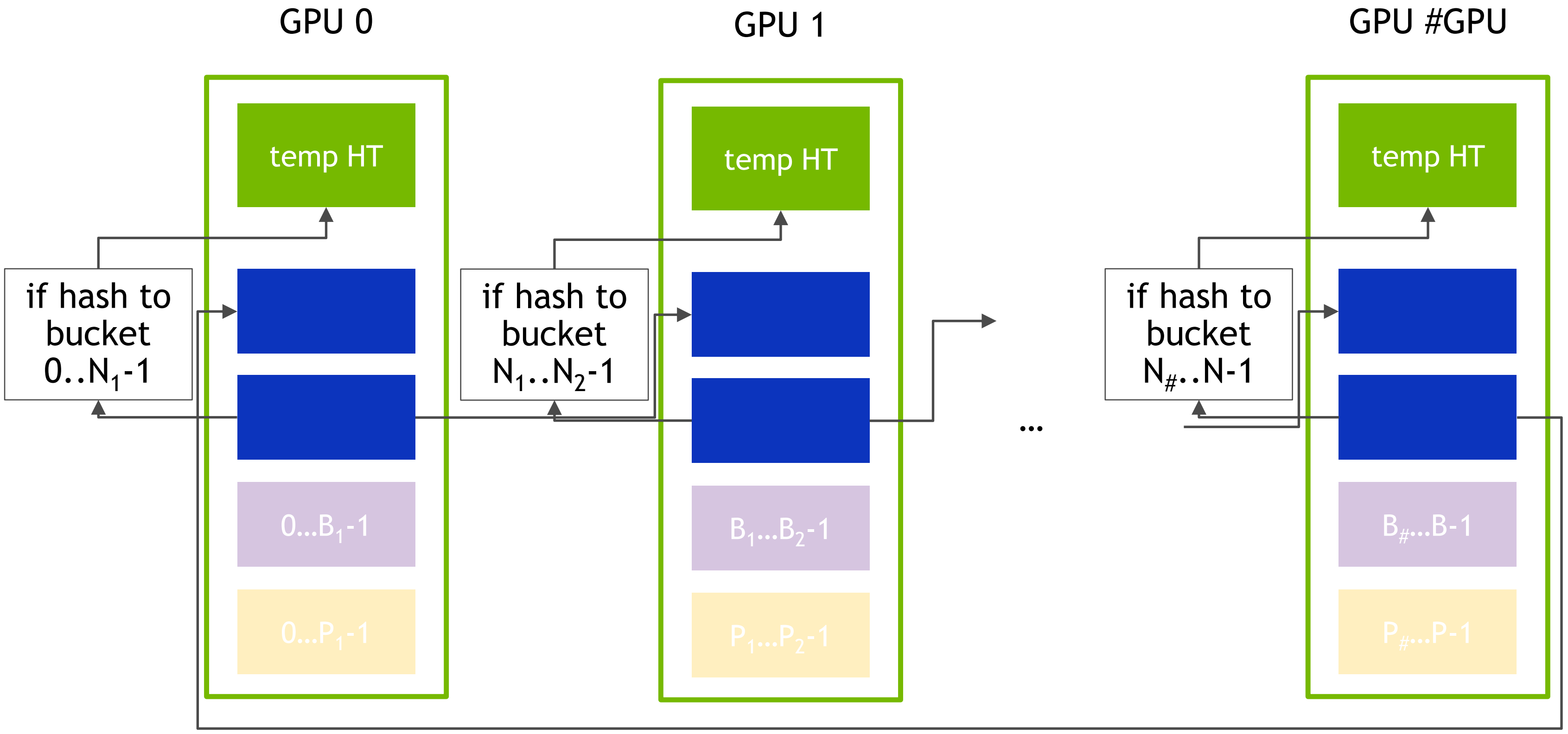
SCALING OF INNER JOIN

parallel build of a replicated HT (step 0 of phase 1)



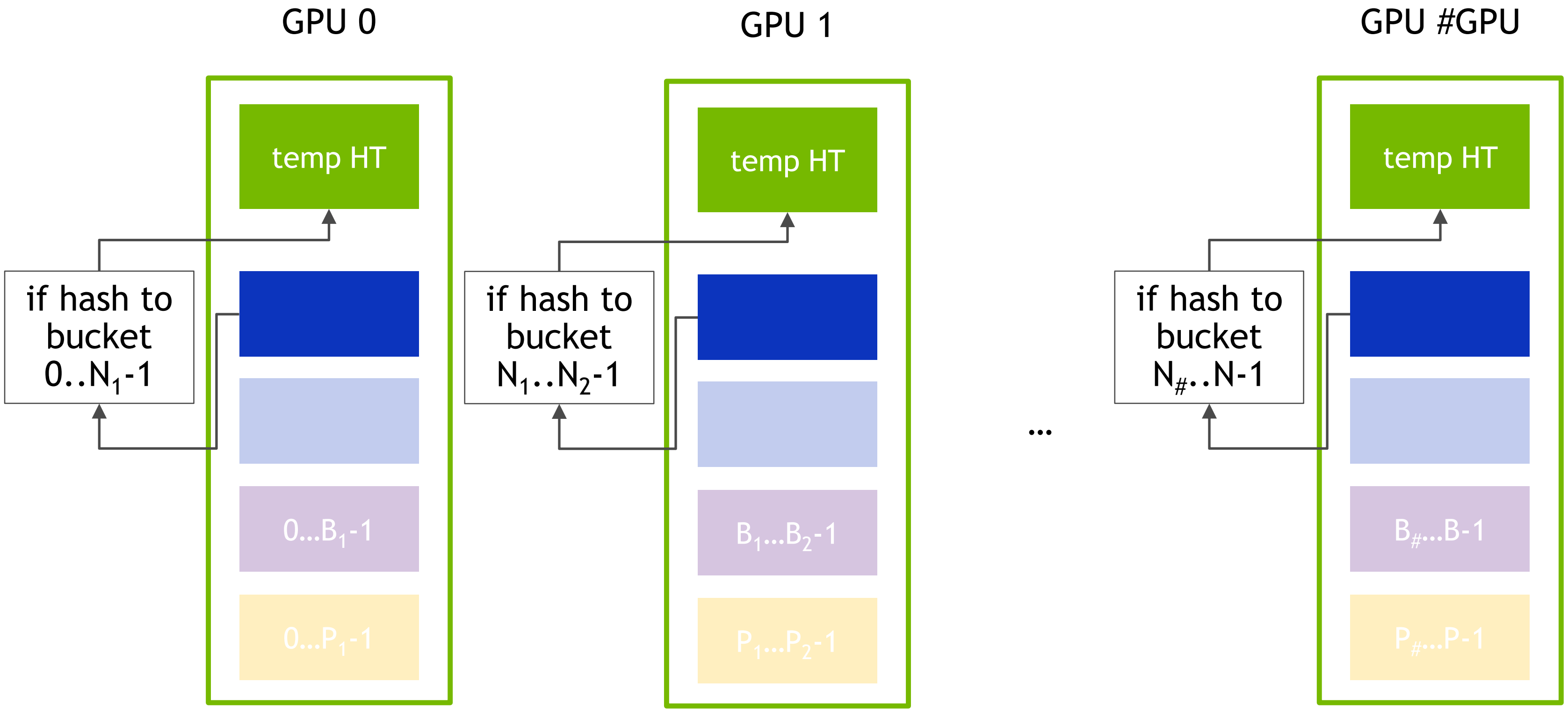
SCALING OF INNER JOIN

parallel build of a replicated HT (step 1..#GPU-1 of phase 1)



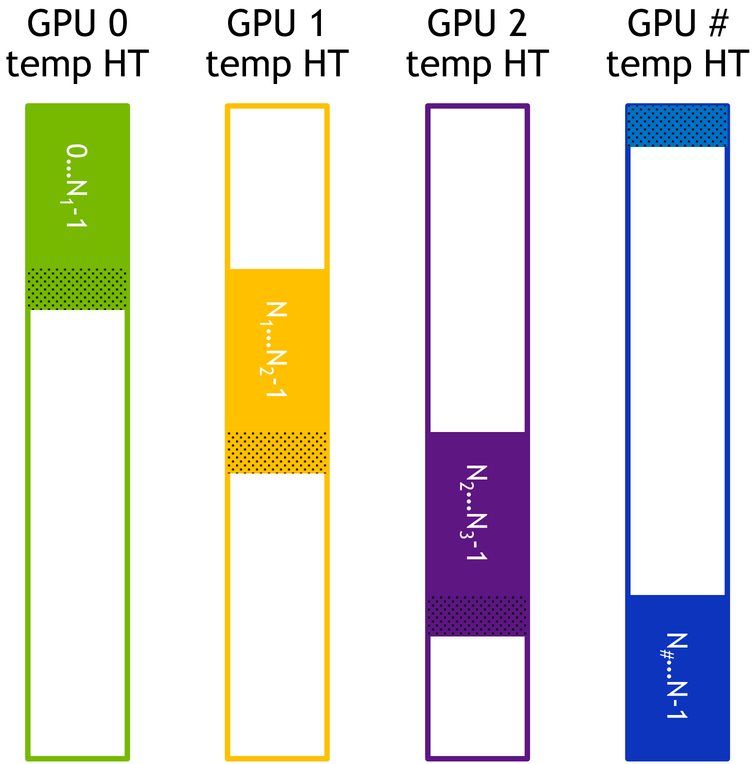
SCALING OF INNER JOIN

parallel build of a replicated HT (step #GPU of phase 1)



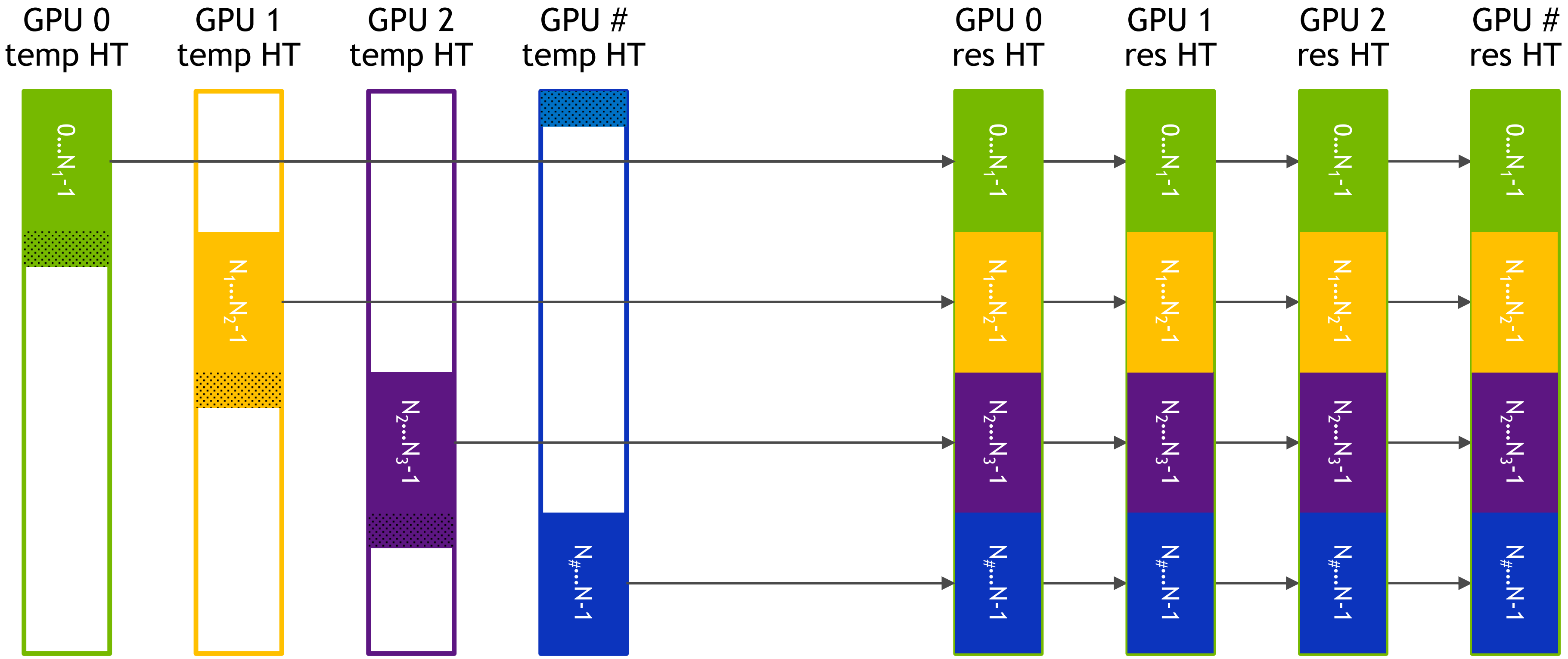
SCALING OF INNER JOIN

parallel build of a replicated HT (phase 2 – merge step)



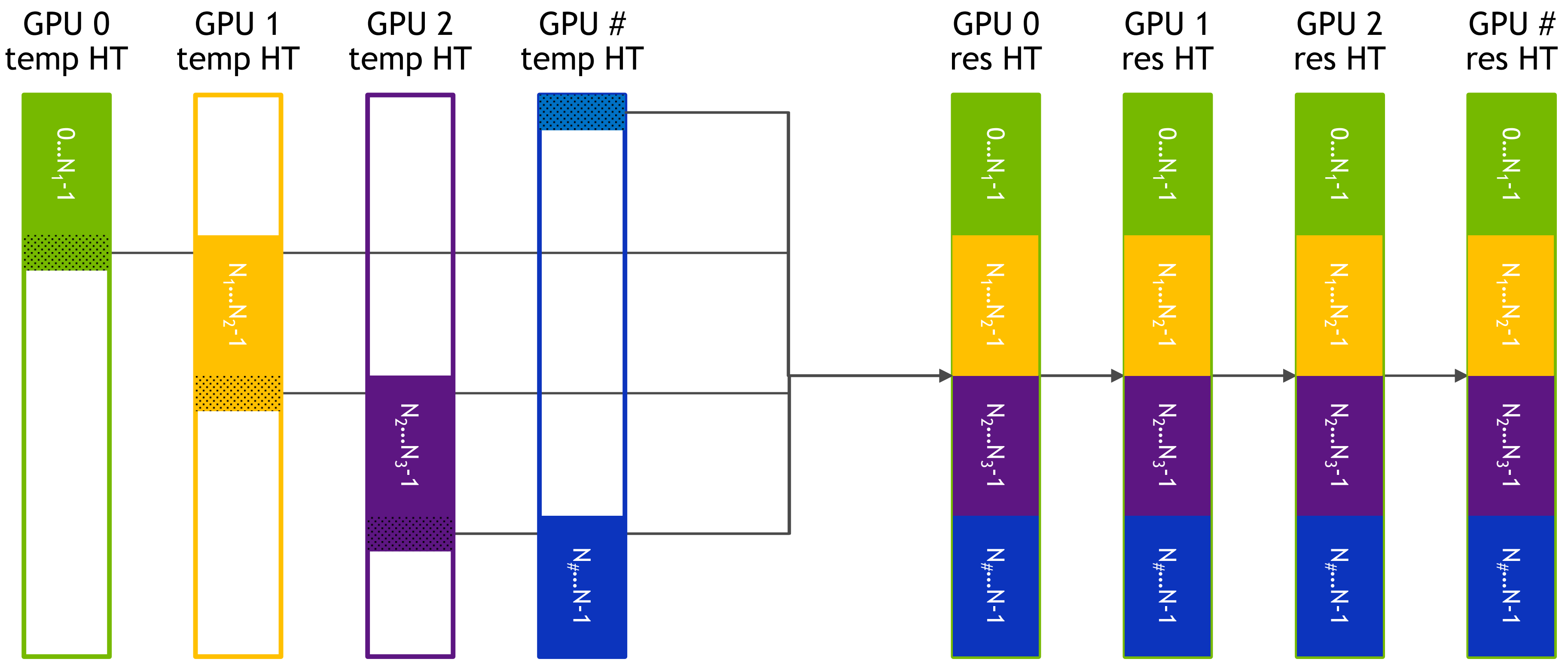
SCALING OF INNER JOIN

parallel build of a replicated HT (phase 2 – merge step)



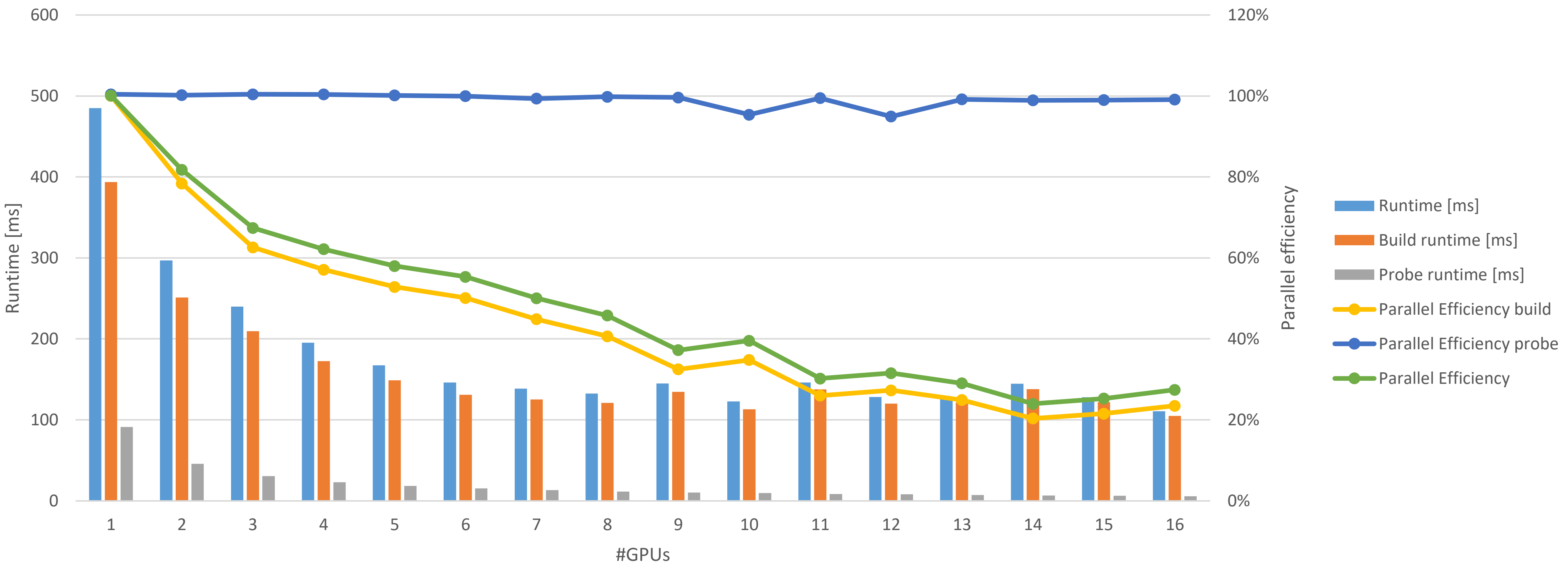
SCALING OF INNER JOIN

parallel build of a replicated HT (phase 2 – merge step)



SCALING OF INNER JOIN ON DGX-2

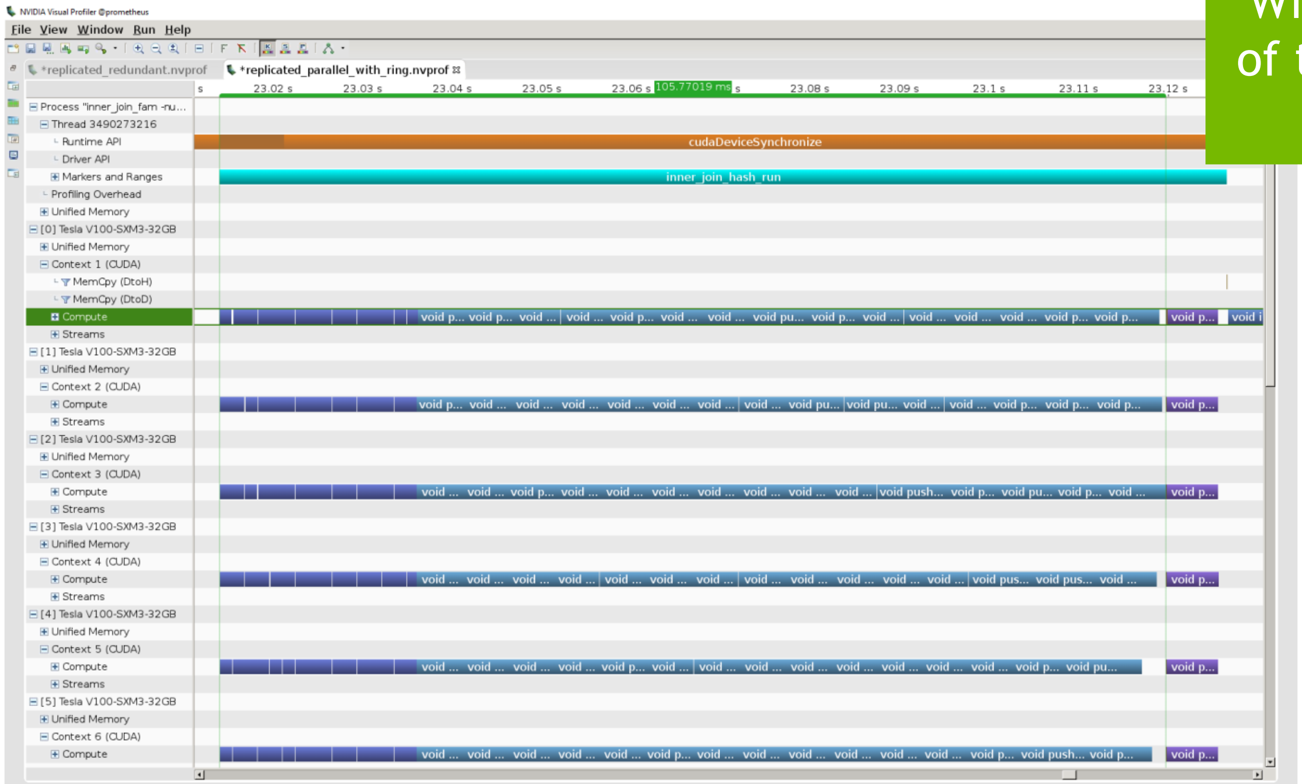
with parallel build of replicated HT



Runtimes are the minimum of 5 repetitions for probe + build (excluding setup overhead, e.g. allocation of hash tables or temp buffers)

SCALING OF INNER JOIN ON DGX-2

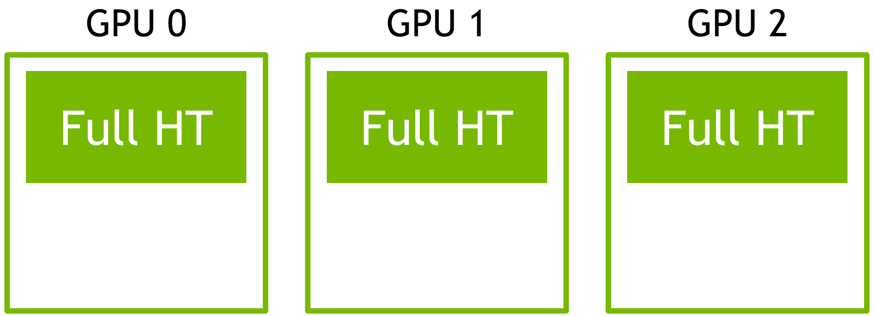
with parallel build of replicated HT

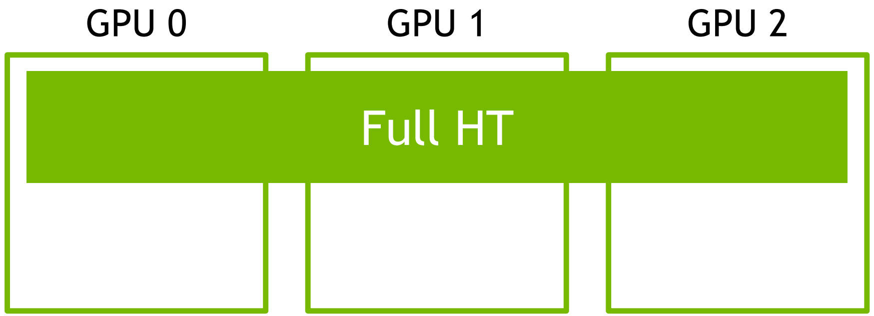


With 16 GPUs most of the time is spend in HT merging

SCALING OF INNER JOIN

parallel build of partitioned HT and parallel probe



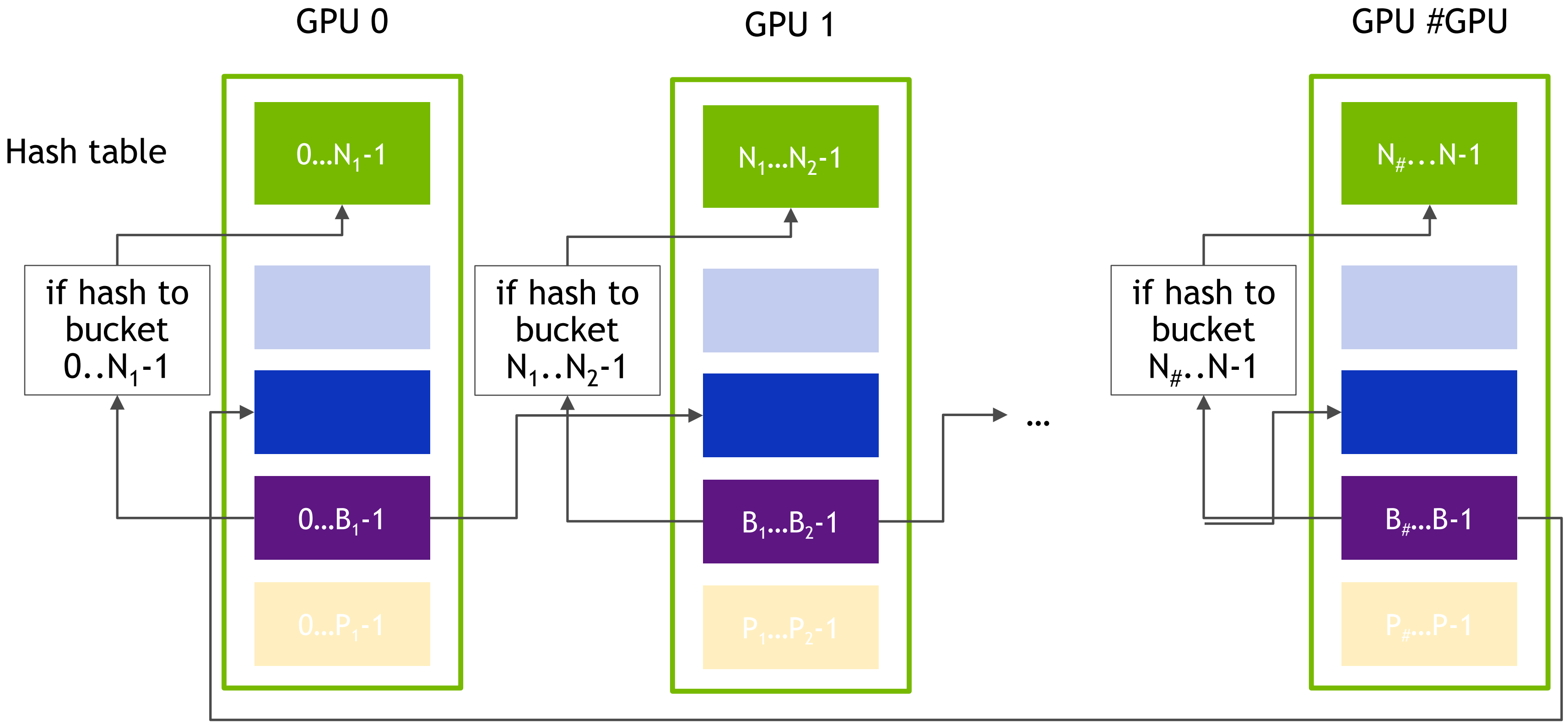


Replicated:
- Limited capacity
- Slower building
  - Need to merge HT partitions
- Faster probing
  - No inter-GPU traffic

Partitioned:
- High capacity
- Faster building
  - No need to merge partitions
- Slower probing
  - Need to access remote partitions

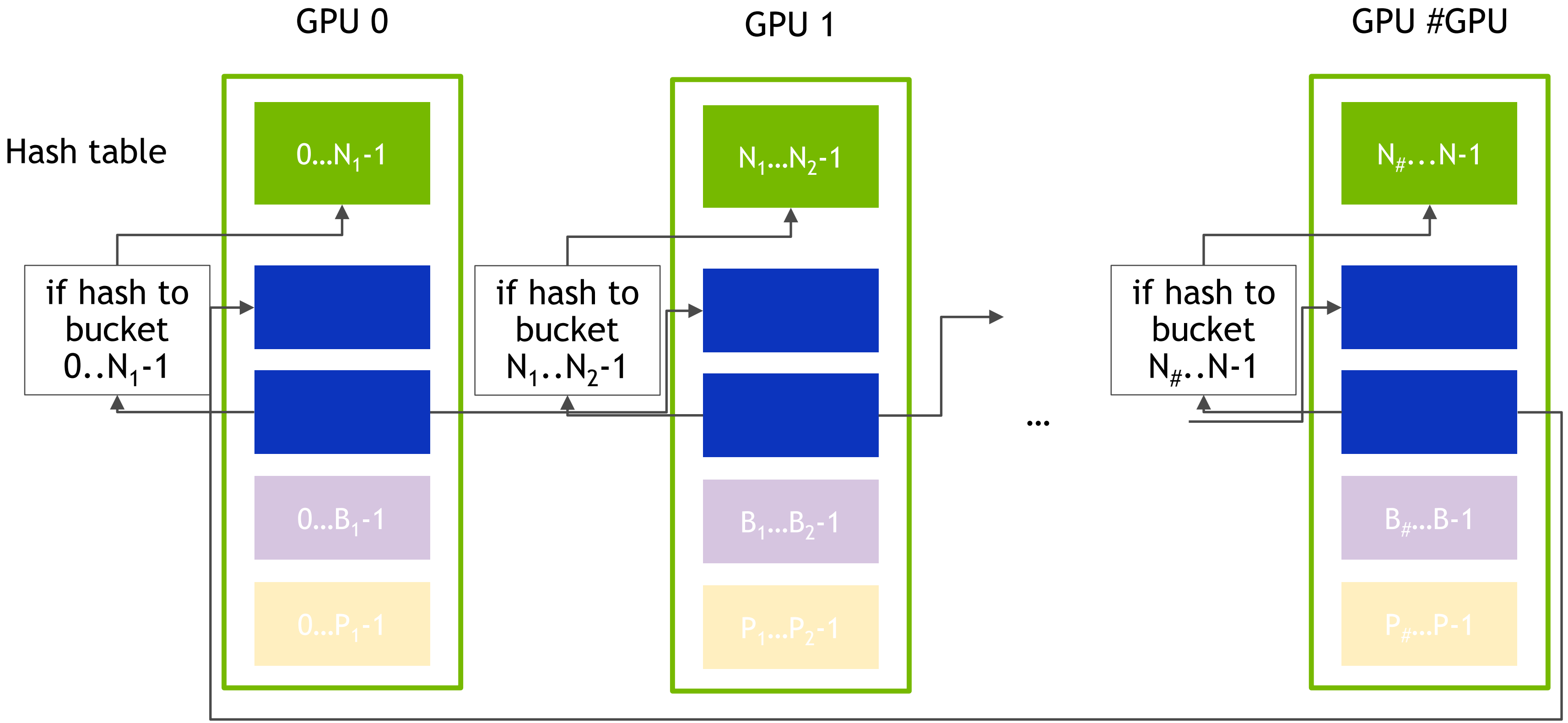
SCALING OF INNER JOIN

parallel build of a partitioned HT (step 0)



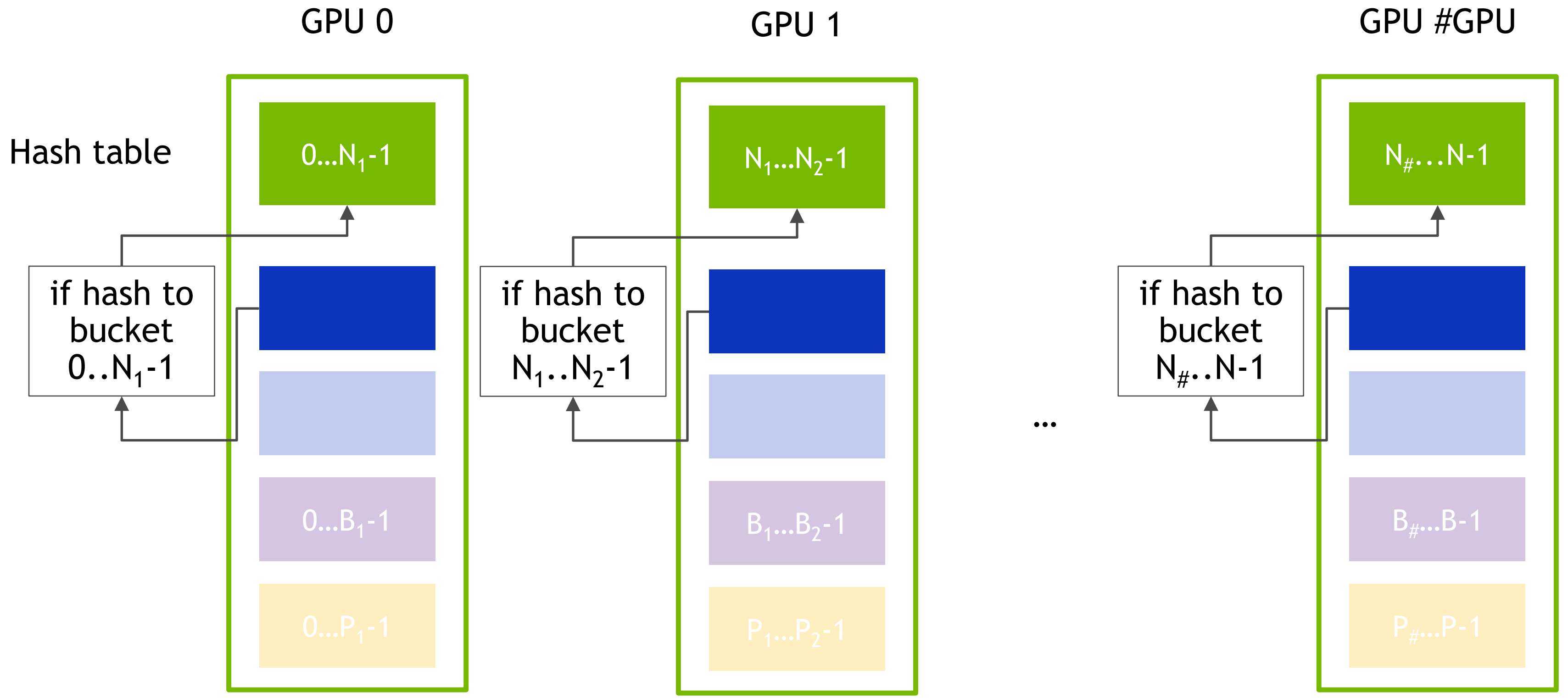
SCALING OF INNER JOIN

parallel build of a partitioned HT (step 1..#GPU-1)



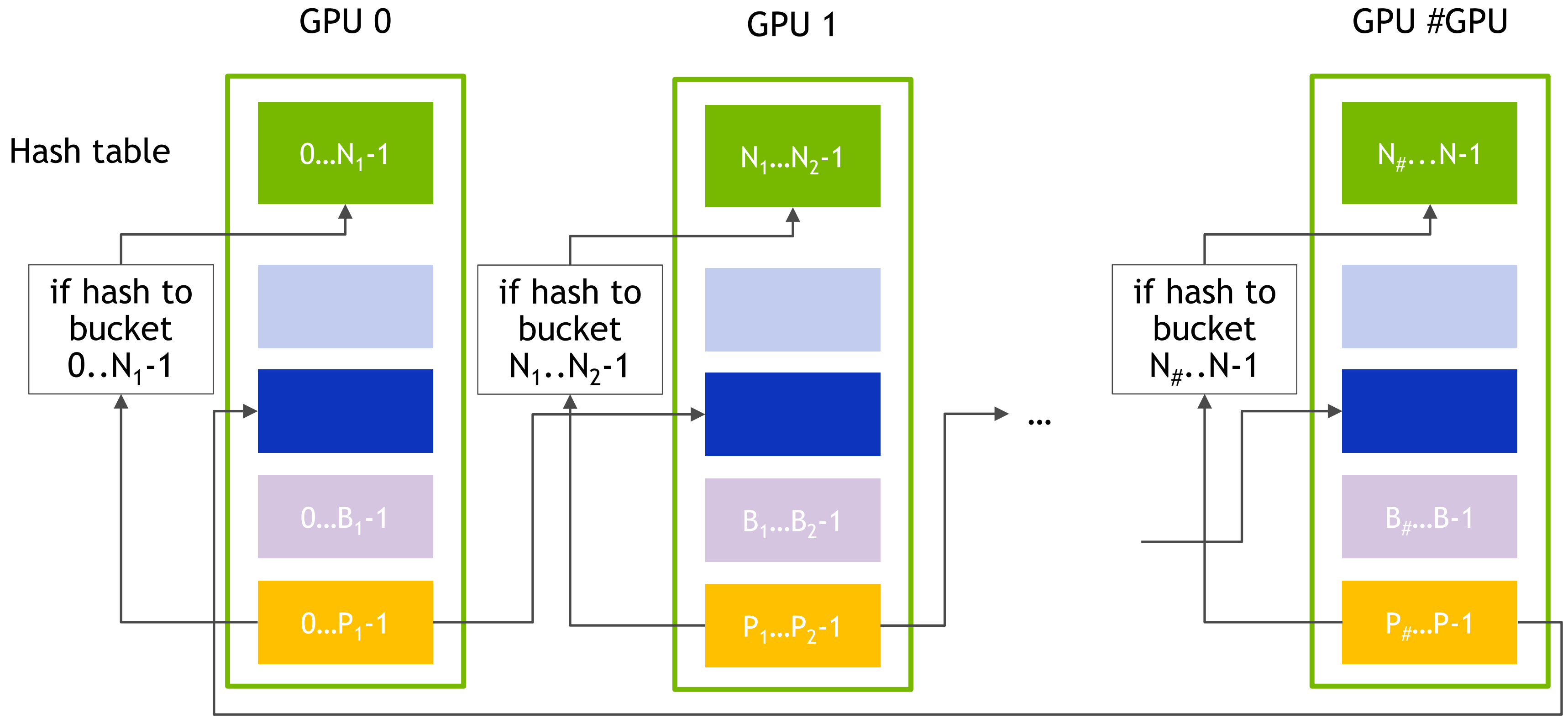
SCALING OF INNER JOIN

parallel build of a partitioned HT (ring exchange) (step #GPU)



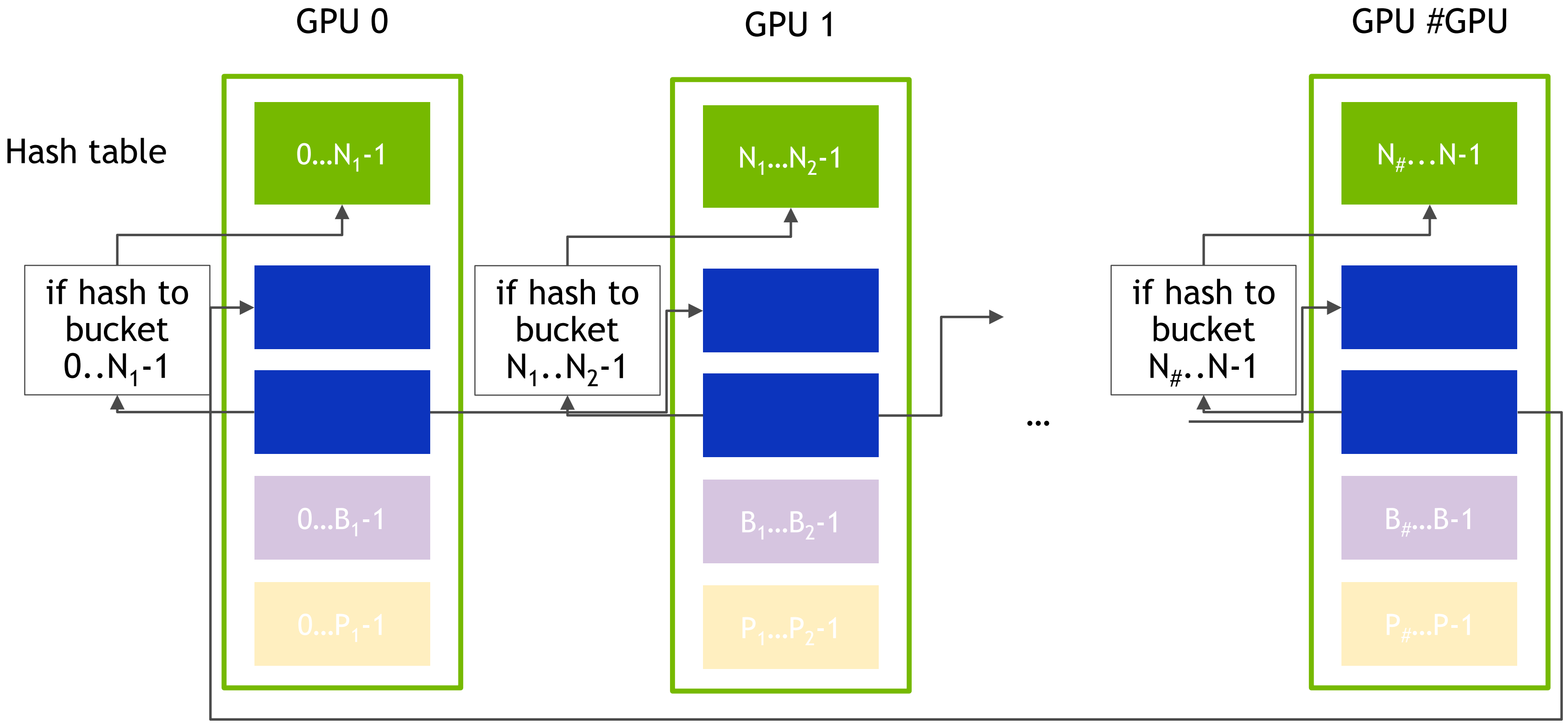
SCALING OF INNER JOIN

parallel probe of a partitioned HT (ring exchange) (step 0)



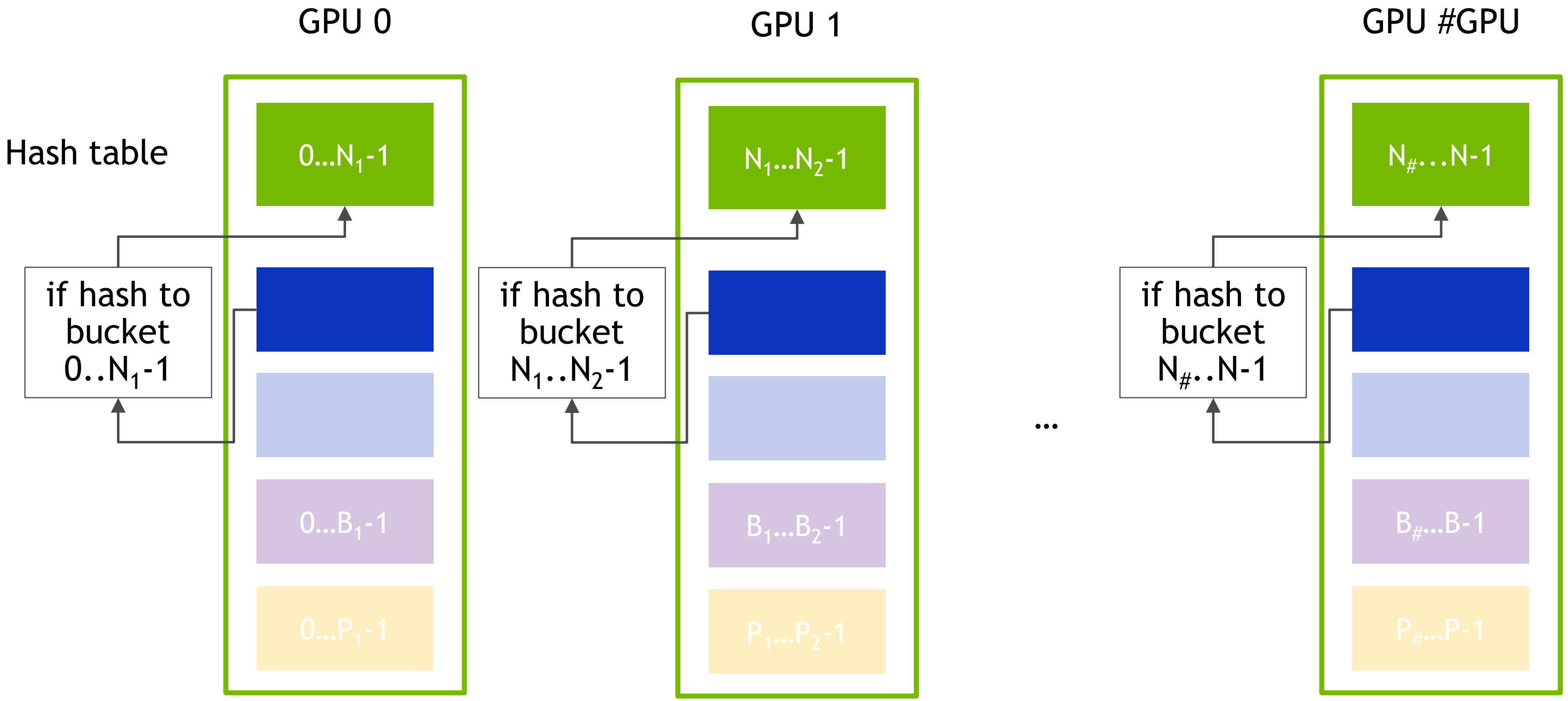
SCALING OF INNER JOIN

parallel probe of a partitioned HT (ring exchange) (step 1..#GPU-1)



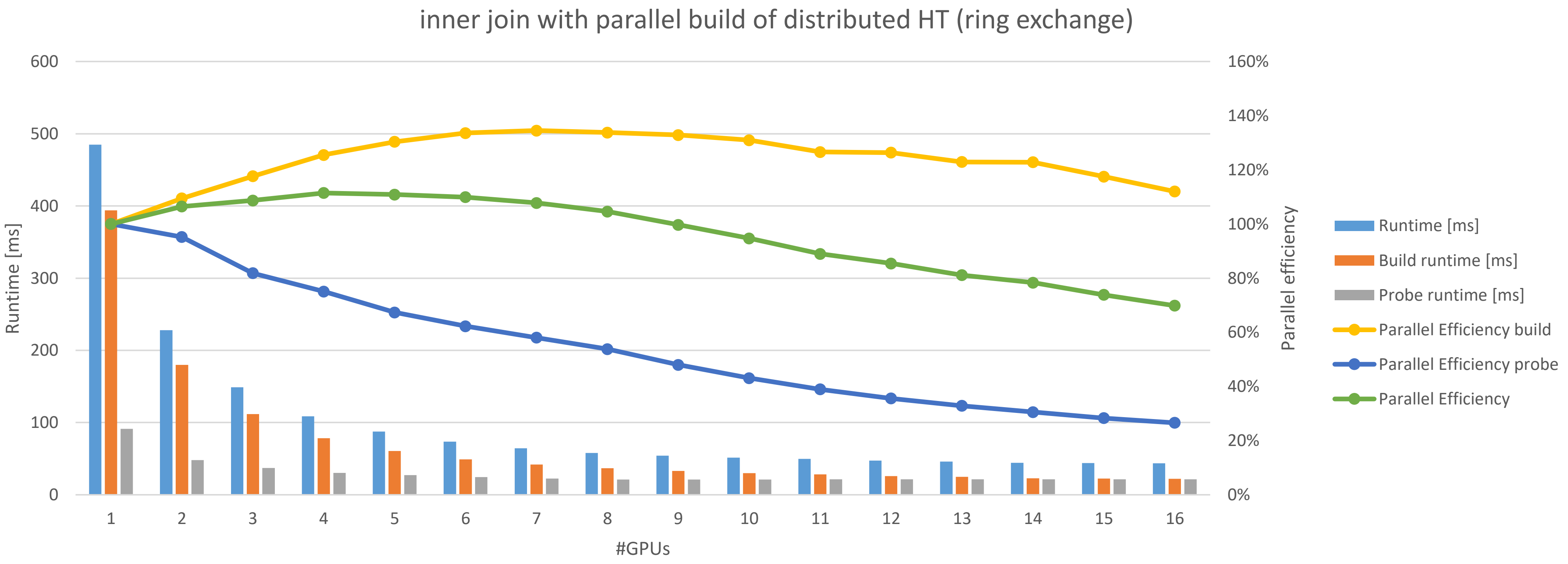
SCALING OF INNER JOIN

parallel probe of a partitioned HT (ring exchange) (step #GPU)



SCALING OF INNER JOIN ON DGX-2

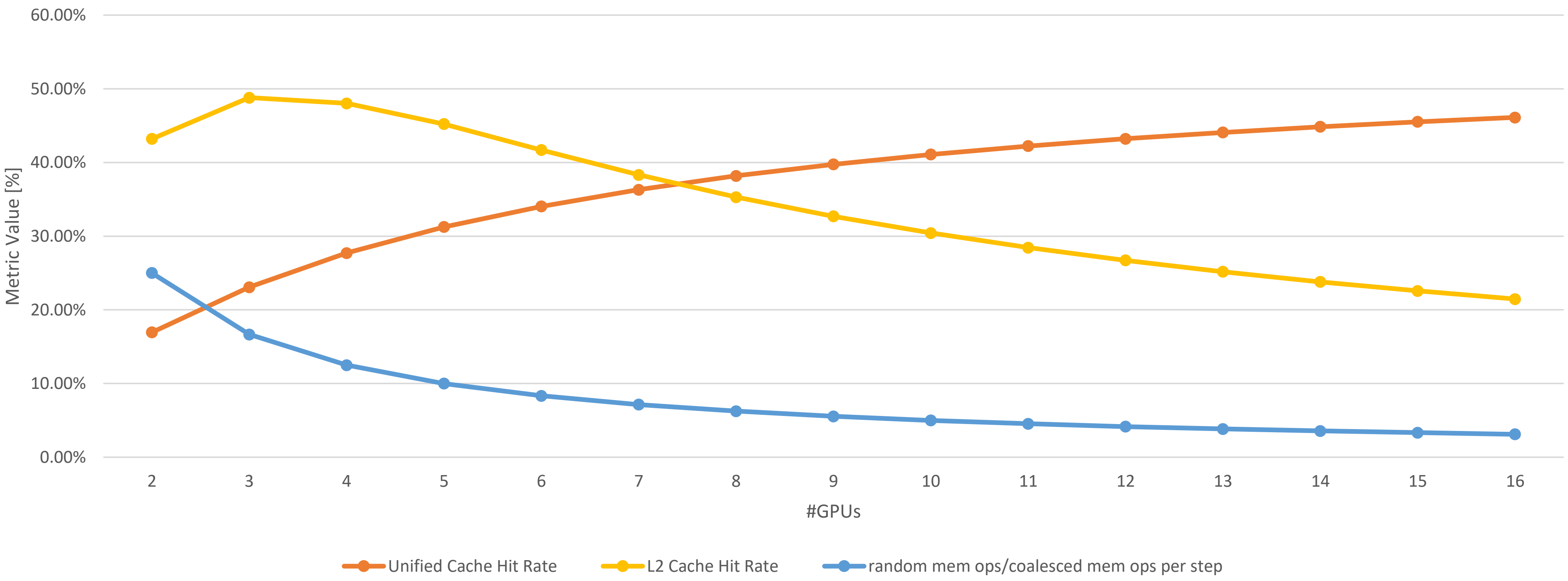
parallel build of partitioned HT and parallel probe (ring exchange)



Runtimes are the minimum of 5 repetitions for probe + build (excluding setup overhead, e.g. allocation of hash tables or temp buffers)

SCALING OF INNER JOIN ON DGX-2

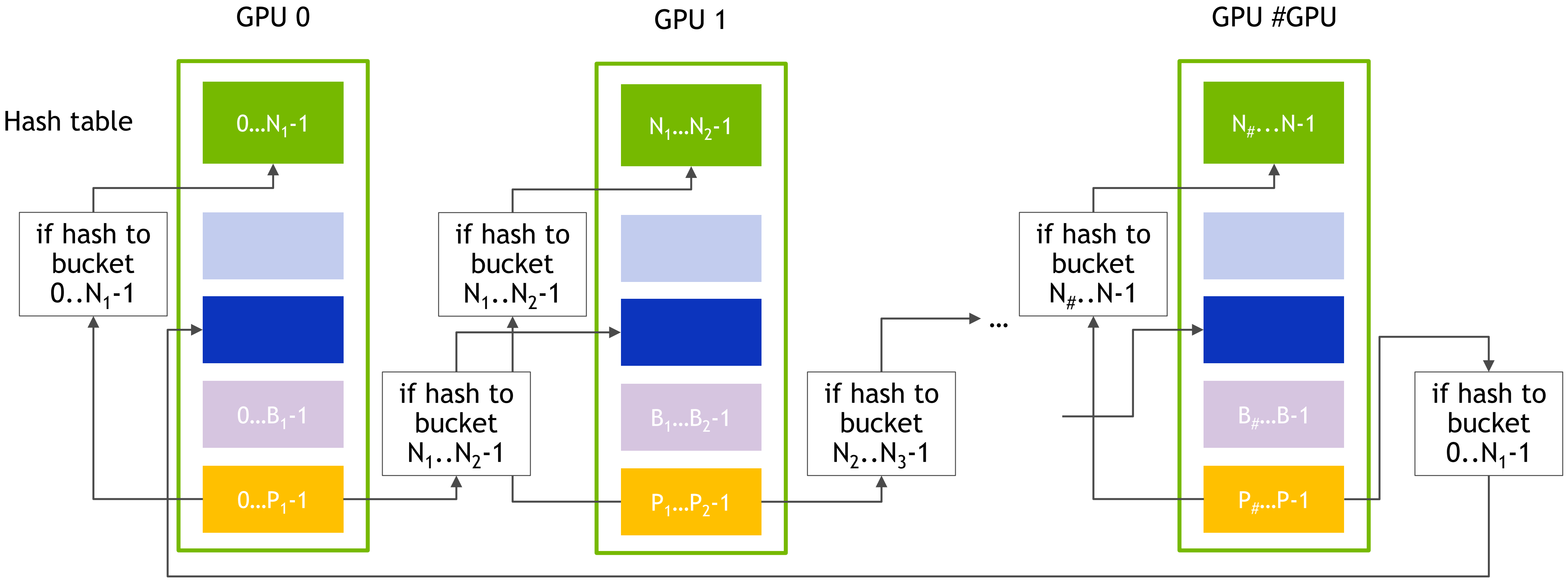
parallel build of partitioned HT – Memory Subsystem Metrics



NVIDIA

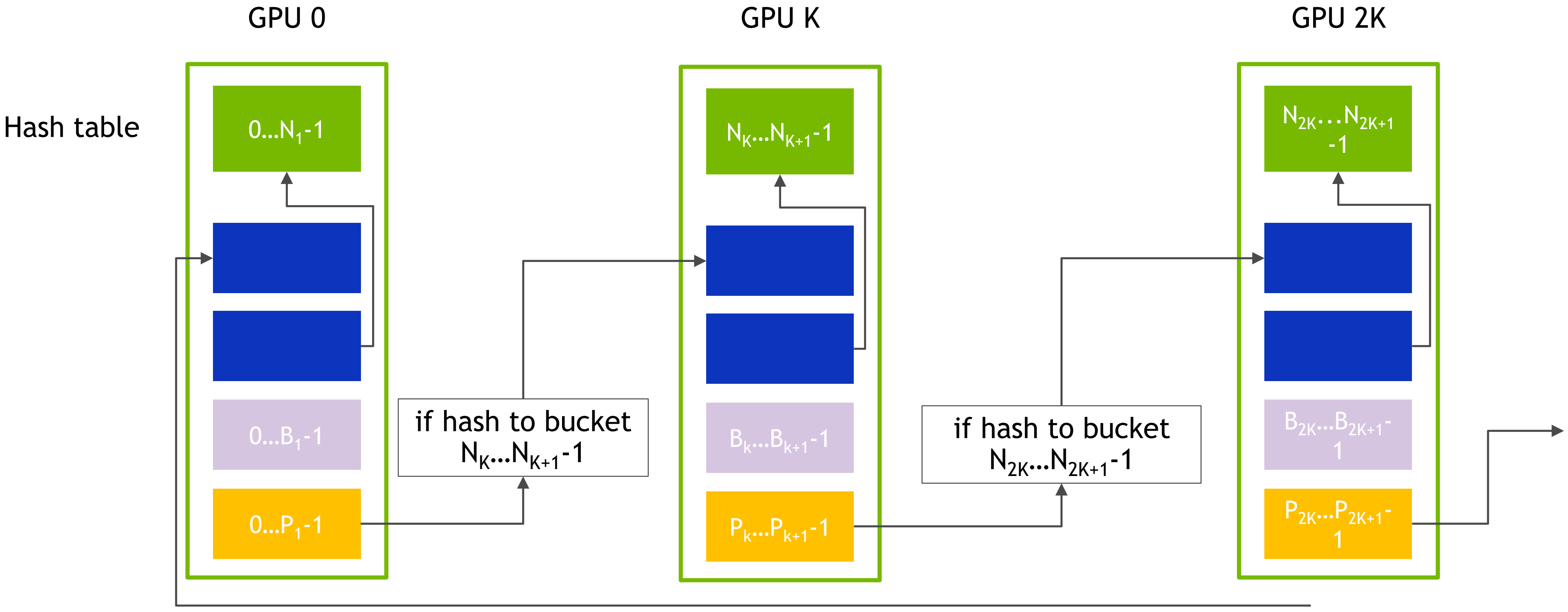
SCALING OF INNER JOIN

parallel probe of a partitioned HT (staged direct send) (round 0)



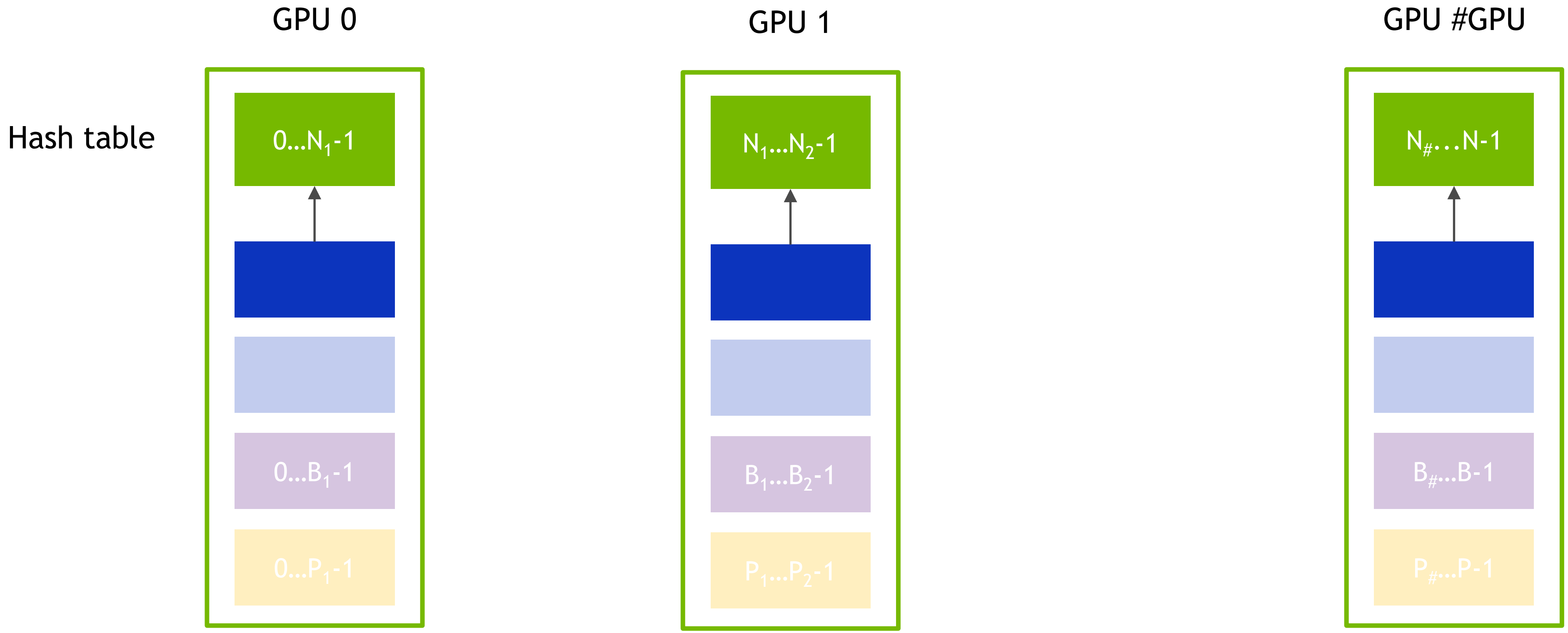
SCALING OF INNER JOIN

parallel probe of a partitioned HT (staged direct send) (round (k-1))



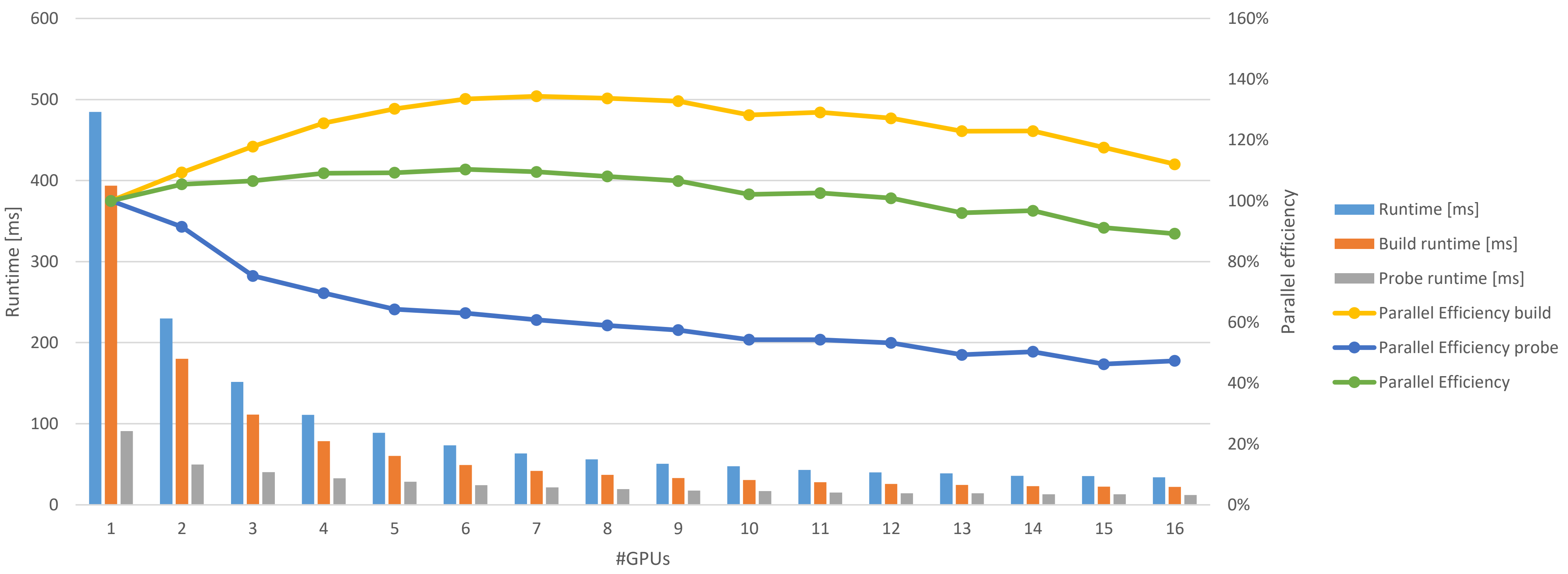
SCALING OF INNER JOIN

parallel probe of a partitioned HT (staged direct send) (round #GPU)



SCALING OF INNER JOIN ON DGX-2

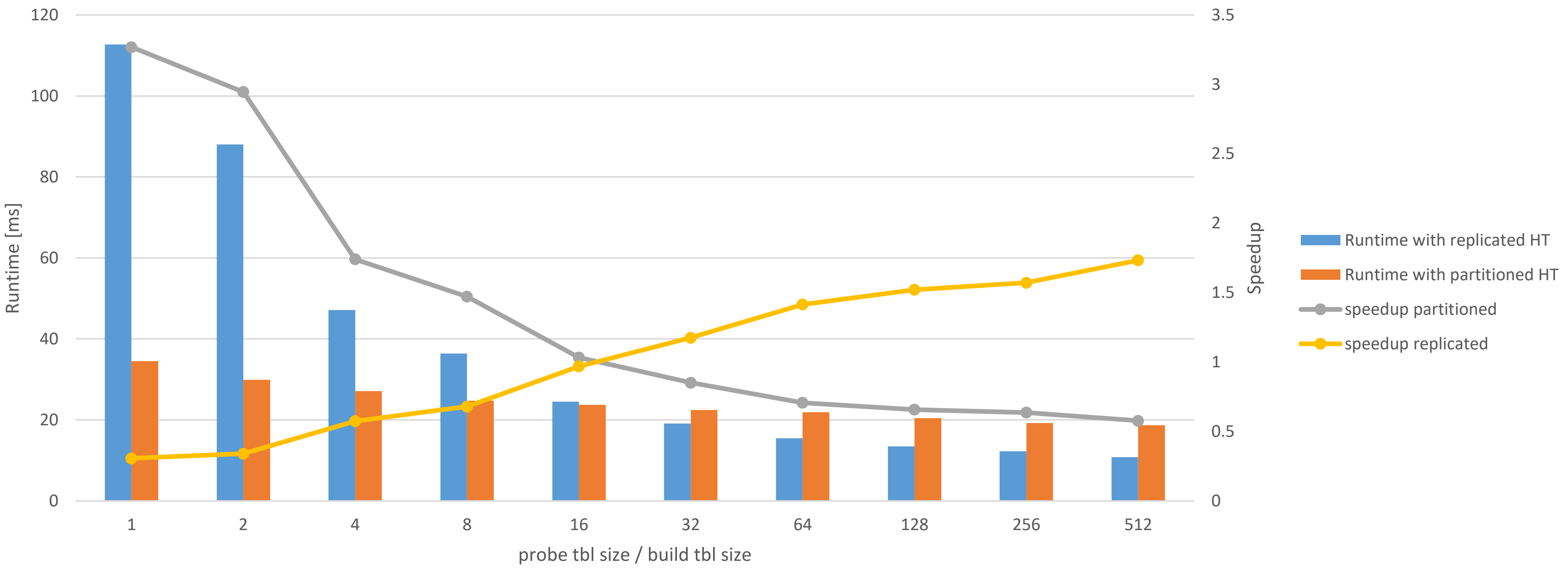
parallel build of partitioned HT and parallel probe (staged direct send)



Runtimes are the minimum of 5 repetitions for probe + build (excluding setup overhead, e.g. allocation of hash tables or temp buffers)

SCALING OF INNER JOIN ON DGX-2

replicated HT vs. partitioned HT (16 GPUs, total # rows = 671088640)



Runtimes are the minimum of 5 repetitions for probe + build (excluding setup overhead, e.g. allocation of hash tables or temp buffers)

REAL OLAP QUERIES

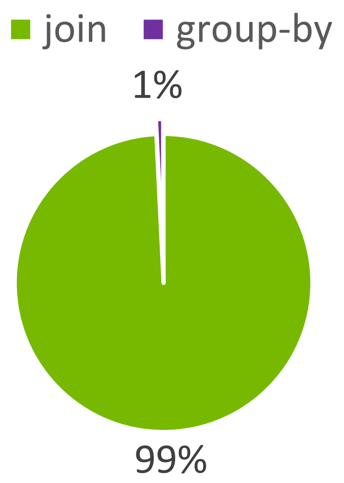
TPC-H BENCHMARK

SQL code for TPC-H Query 4:

```
select
  o_orderpriority,
  count(o_orderkey) as order_count,
from
  orders
where
  o_orderdate >= date '[DATE]' and
  o_orderdate < date '[DATE]' + interval '3' month and
  exists (select * from lineitem
          where l_orderkey = o_orderkey and
                l_commitdate < l_receiptdate)
group by
  o_orderpriority,
order by
  o_orderpriority;
```

semi-join

CPU execution breakdown



Q4: INPUT DATA

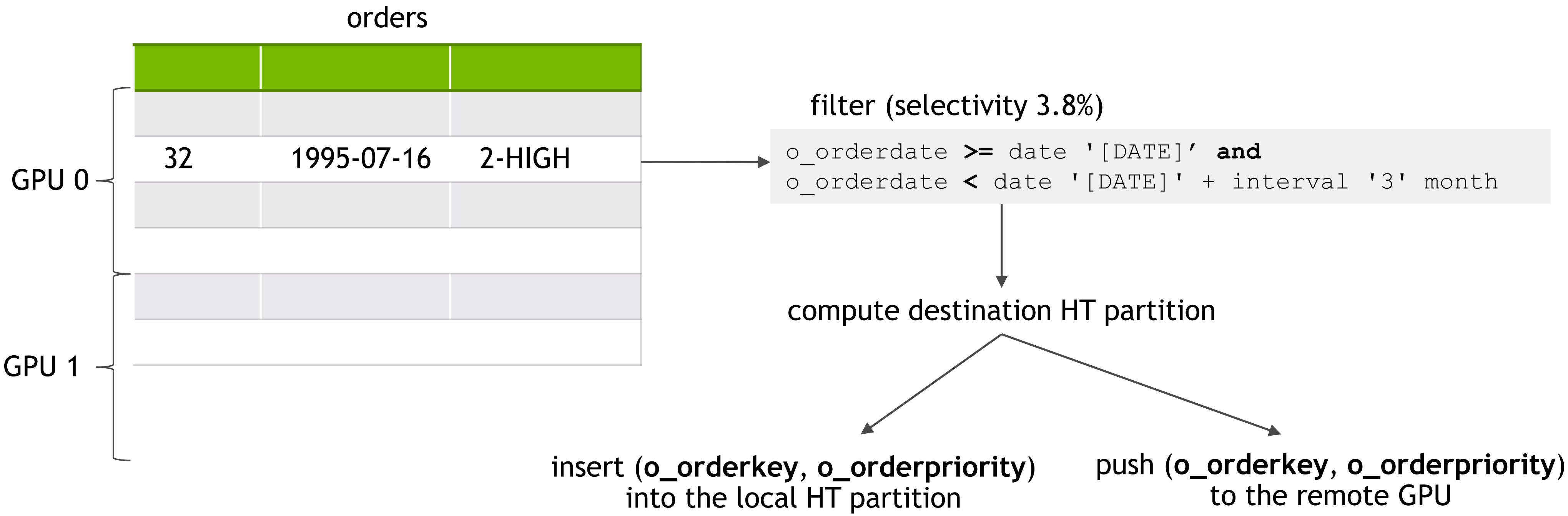
1.5M rows per SF

o_orderkey	o_orderdate	o_orderpriority
7	1996-01-10	2-HIGH
32	1995-07-16	2-HIGH
33	1993-10-27	3-MEDIUM
34	1998-07-21	3-MEDIUM

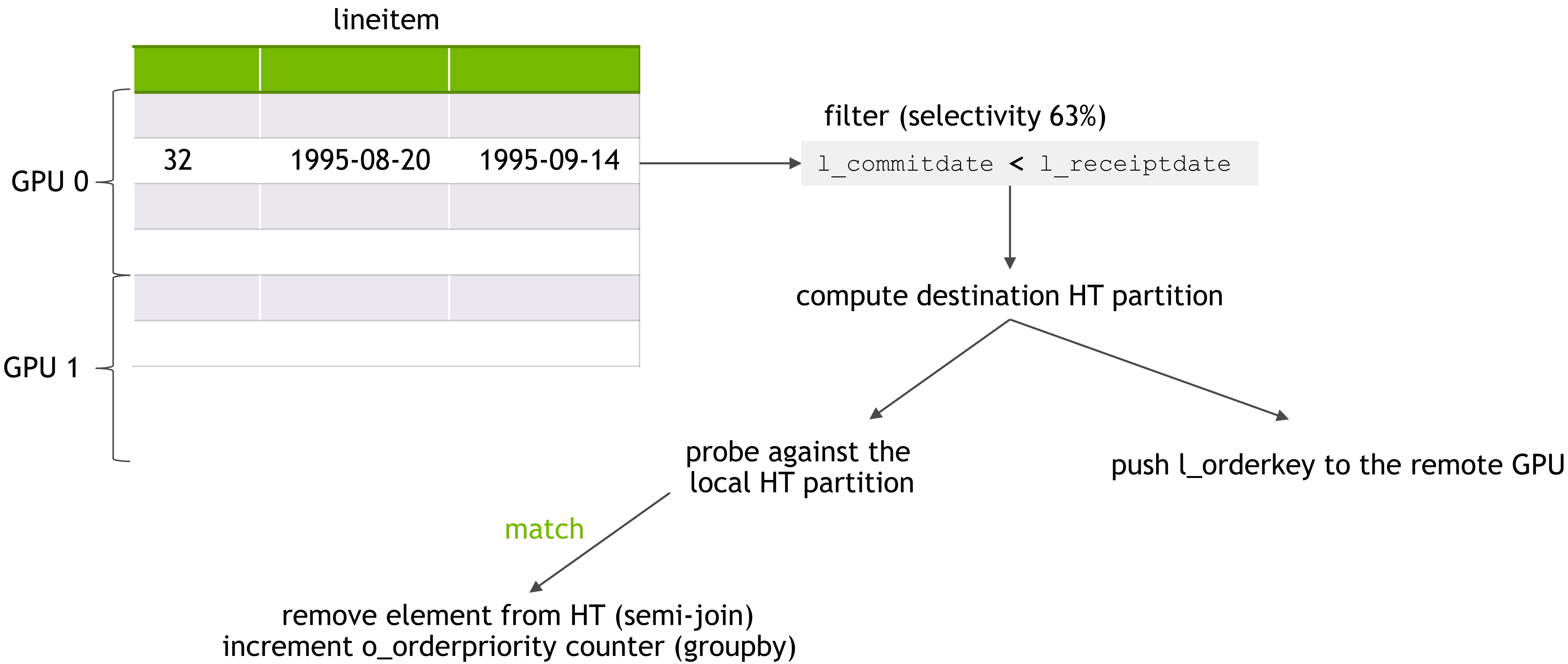
6M rows per SF

l_orderkey	l_commitdate	l_receiptdate
7		
7		
7		
32	1995-10-07 >	1995-08-27
32	1995-08-20 <	1995-09-14
32	1995-10-01 >	1995-09-03
34		
34		

Q4 JOIN: BUILD



Q4 JOIN: PROBE

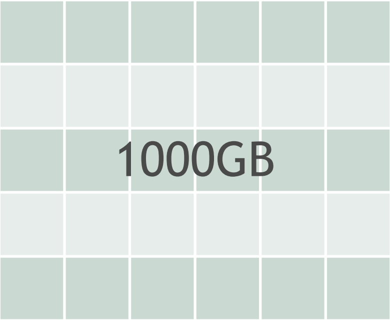


TEST SETUP

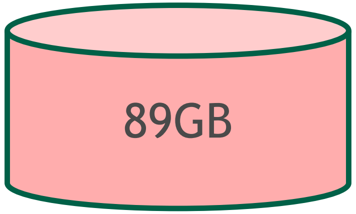
TPC-H Q4 SF1000

Performance metrics: **time**, **parallel efficiency**, **throughput** (input data size / time)

Use 8B keys, 2B encoded dates, 1B encoded priority string



All tables in CSV format

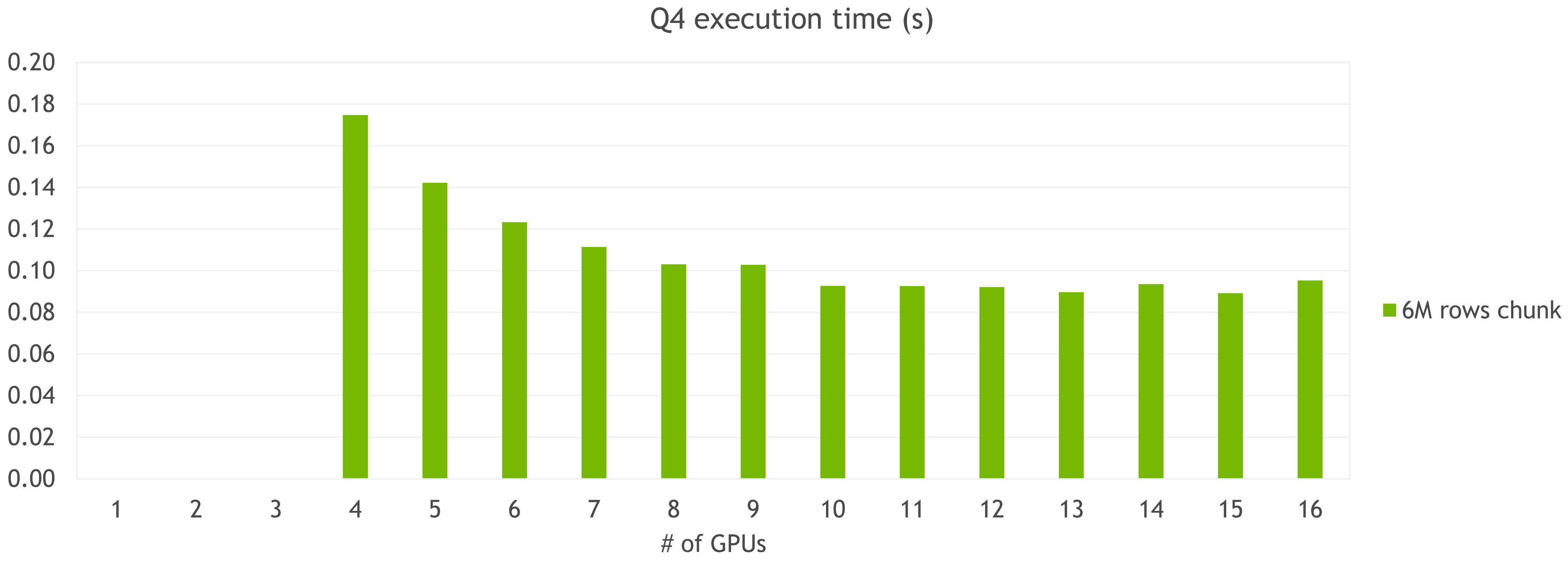


Input columns
used in Q4

GPU hash table
(50% HT occupancy)

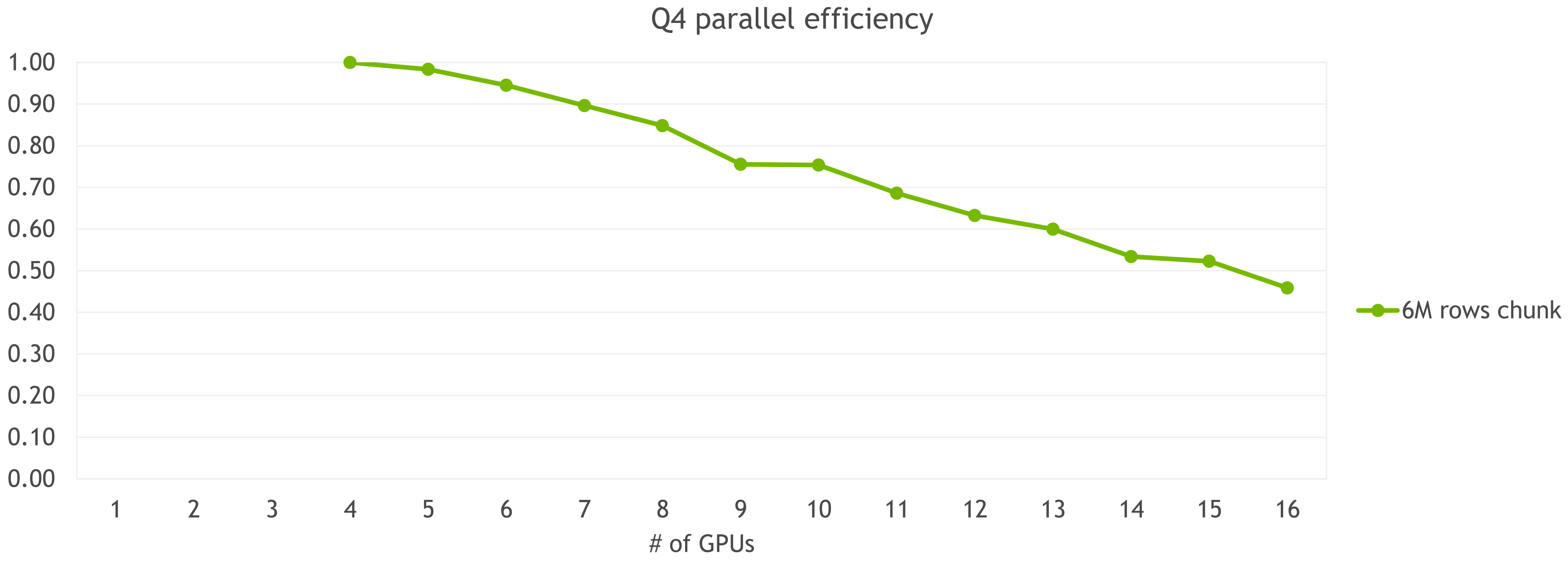
PERFORMANCE RESULTS ON DGX-2

Q4 SF1000, input distributed in GPU memory

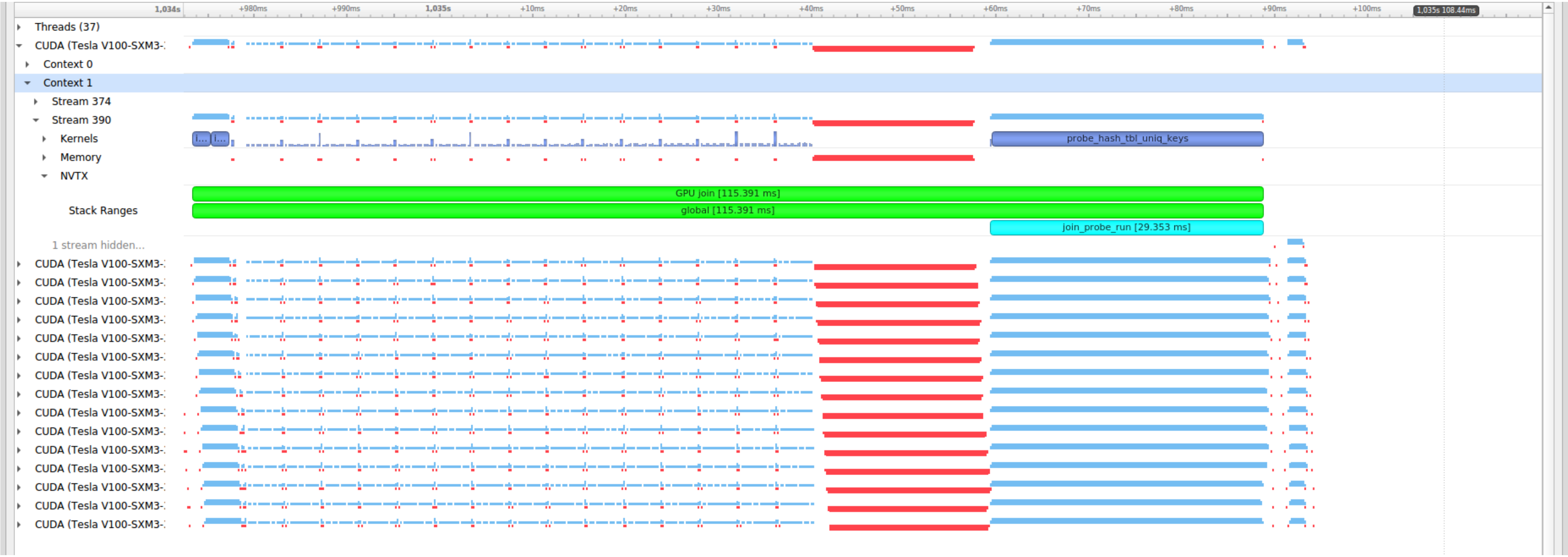


PERFORMANCE RESULTS ON DGX-2

Q4 SF1000, input distributed in GPU memory

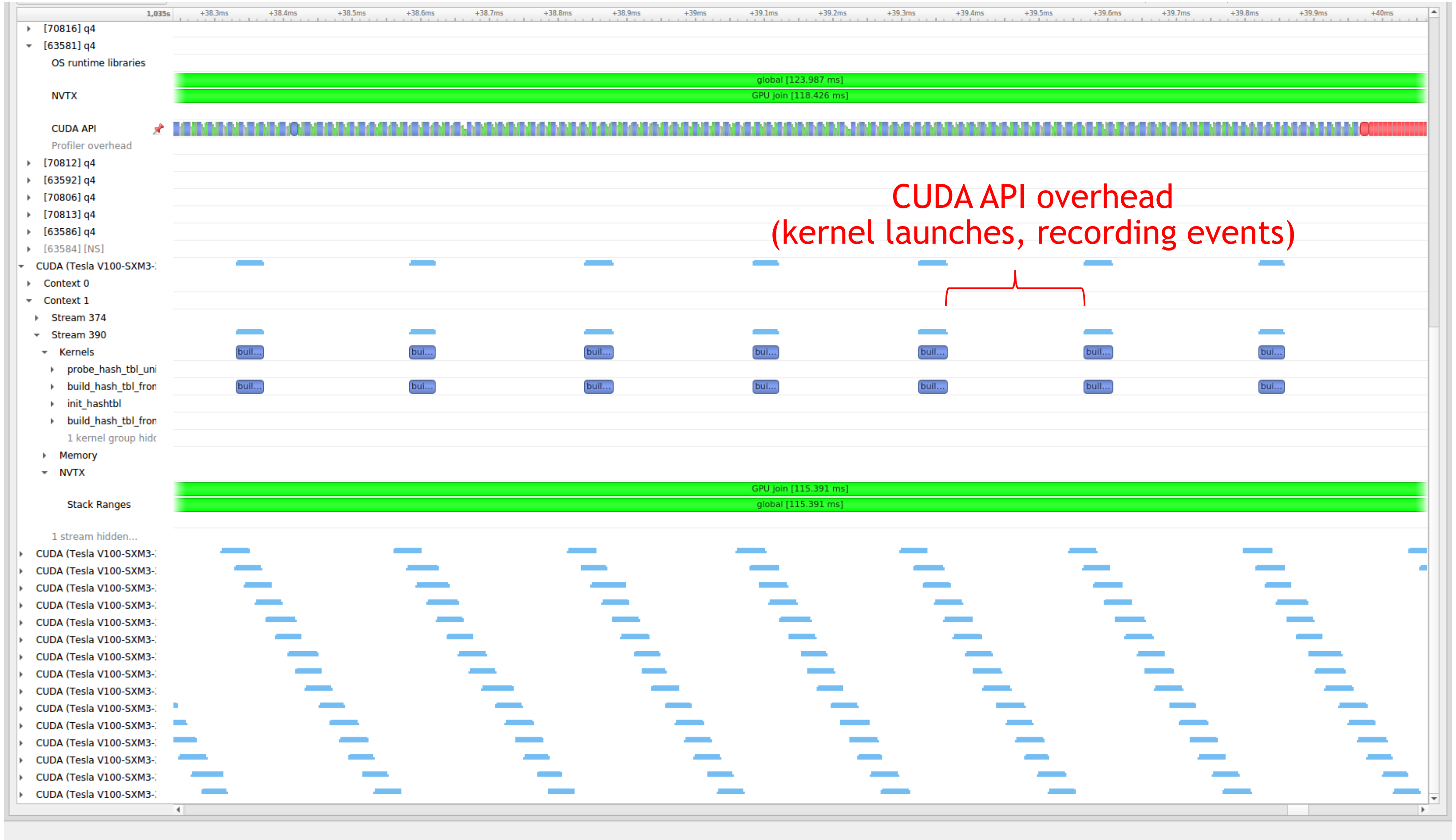


DGX-2 PROFILE: INPUT IN GPU MEMORY



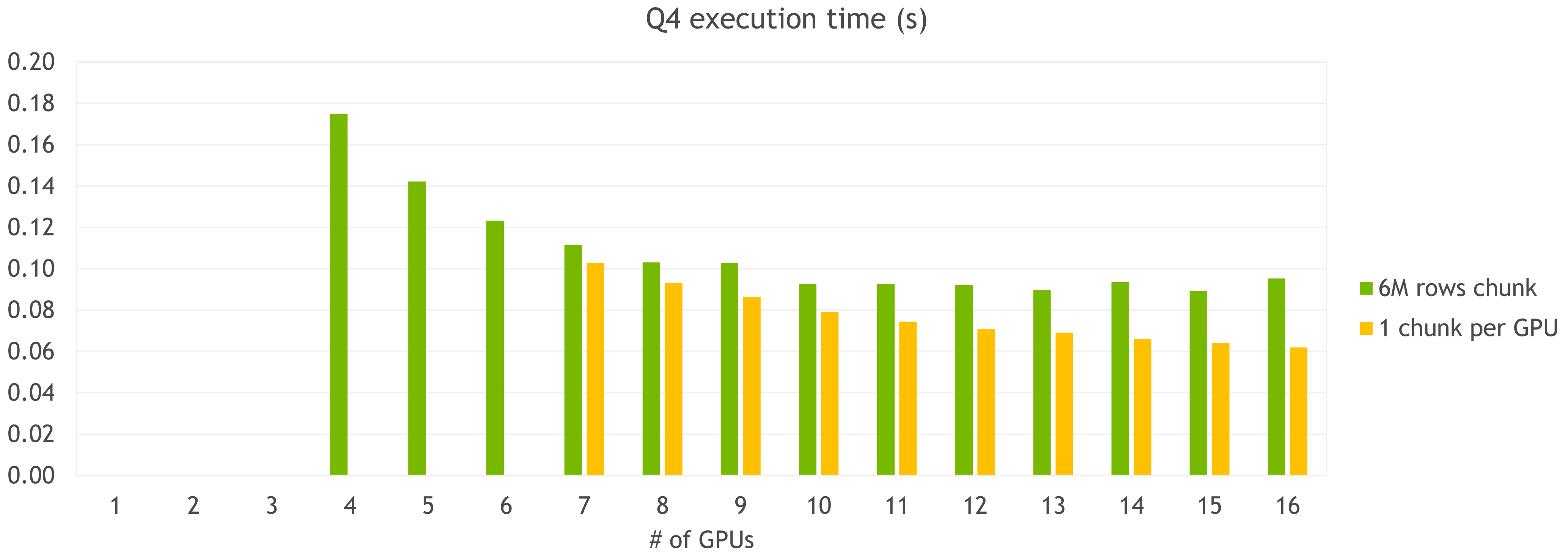
the main bottleneck is HT build (74% of the overall query time)

DGX-2 PROFILE: INPUT IN GPU MEMORY



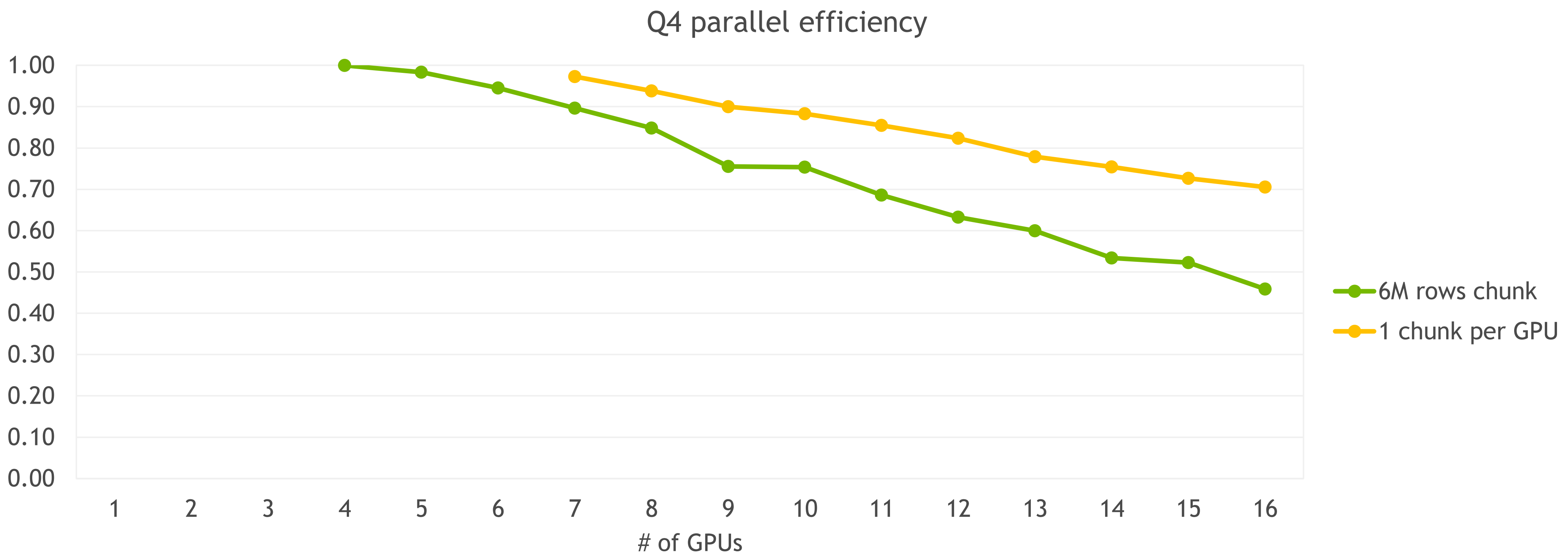
OPTIMIZED CHUNK SIZE ON DGX-2

Q4 SF1000, input distributed in GPU memory



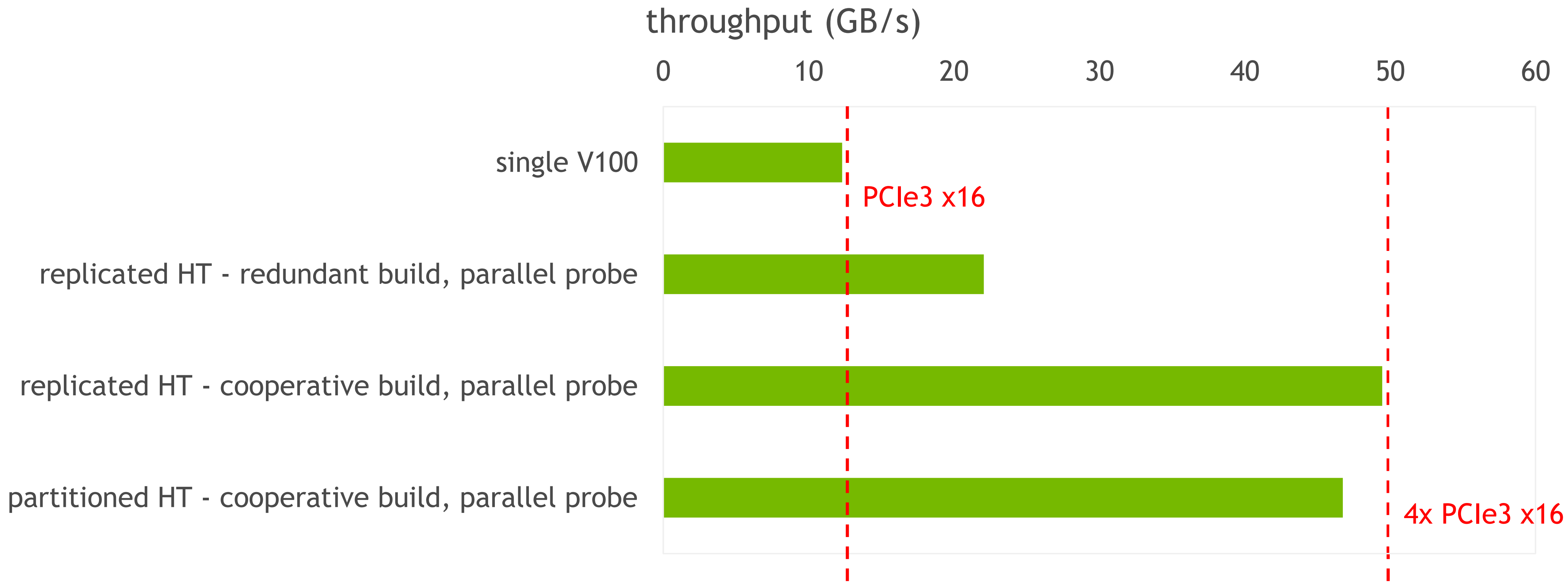
OPTIMIZED CHUNK SIZE ON DGX-2

Q4 SF1000, input distributed in GPU memory

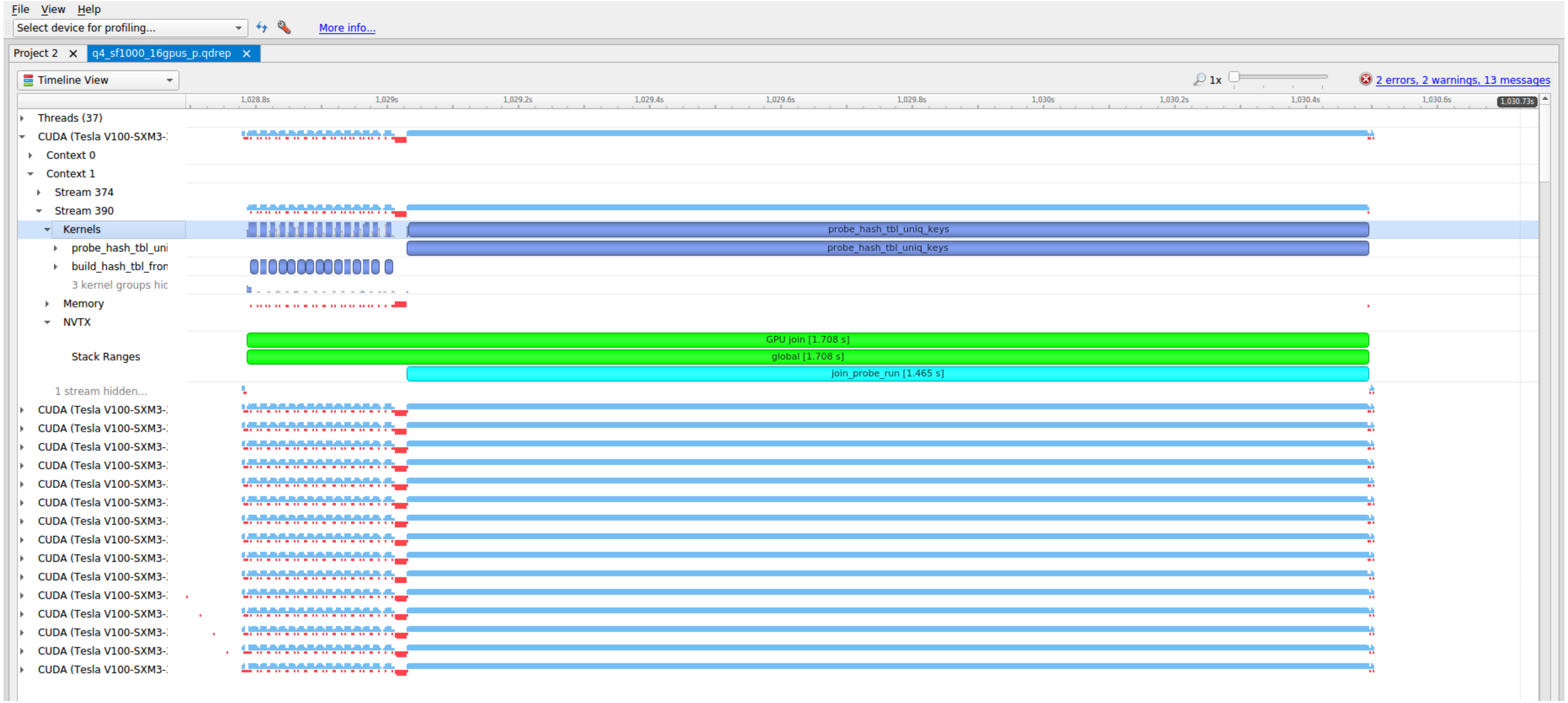


PERFORMANCE RESULTS ON DGX-2

Q4 SF1000, input in system memory



DGX-2 PROFILE: INPUT IN CPU MEMORY



the main bottleneck is HT probe (82% of the overall query time)

IS THIS THE BEST WE CAN DO?

8B	2B	2B
l_orderkey	l_commitdate	l_receiptdate
7		
7		
7		
32	1995-10-07	1995-08-27
32	1995-08-20	1995-09-14
32	1995-10-01	1995-09-03
34		
34		

IS THIS THE BEST WE CAN DO?

l_orderkey	l_commitdate	l_receiptdate
7		
7		
7		
32	1995-10-07	1995-08-27
32	1995-08-20	1995-09-14
32	1995-10-01	1995-09-03
34		
34		

filters can be executed on the CPU

IS THIS THE BEST WE CAN DO?

8B	2B	2B
l_orderkey	l_commitdate	l_receiptdate
7		
7		
7		
32	1995-10-07	1995-08-27
32	1995-08-20	1995-09-14
32	1995-10-01	1995-09-03
34		
34		

can be compressed to <8B per key

IS THIS THE BEST WE CAN DO?

8B	2B	2B
l_orderkey	l_commitdate	l_receiptdate
7		
7		
7		
32	1995-10-07	1995-08-27
32	1995-08-20	1995-09-14
32	1995-10-01	1995-09-03
34		
34		

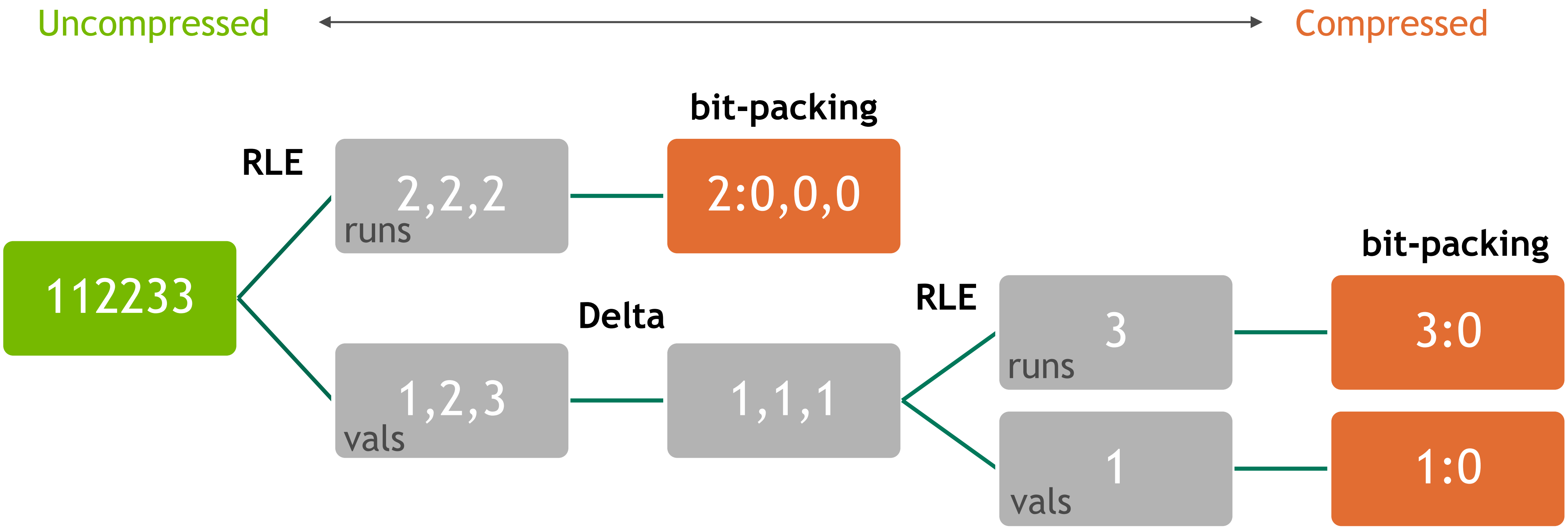
can be compressed to <2B per date

IS THIS THE BEST WE CAN DO?

8B	2B	2B
l_orderkey	l_commitdate	l_receiptdate
7		
7		
7		
32	1995-10-07	1995-08-27
32	1995-08-20	1995-09-14
32	1995-10-01	1995-09-03
34		
34		

can be compressed to <2B per date

RLE-DELTA-RLE COMPRESSION



[1] *c.f.“Breaking the Speed of Interconnect with Compression for Database Applications”, GTC Silicon Valley 2018, Session ID S8417*
http://on-demand-gtc.gputechconf.com/gtc-quicklink/7LVQs

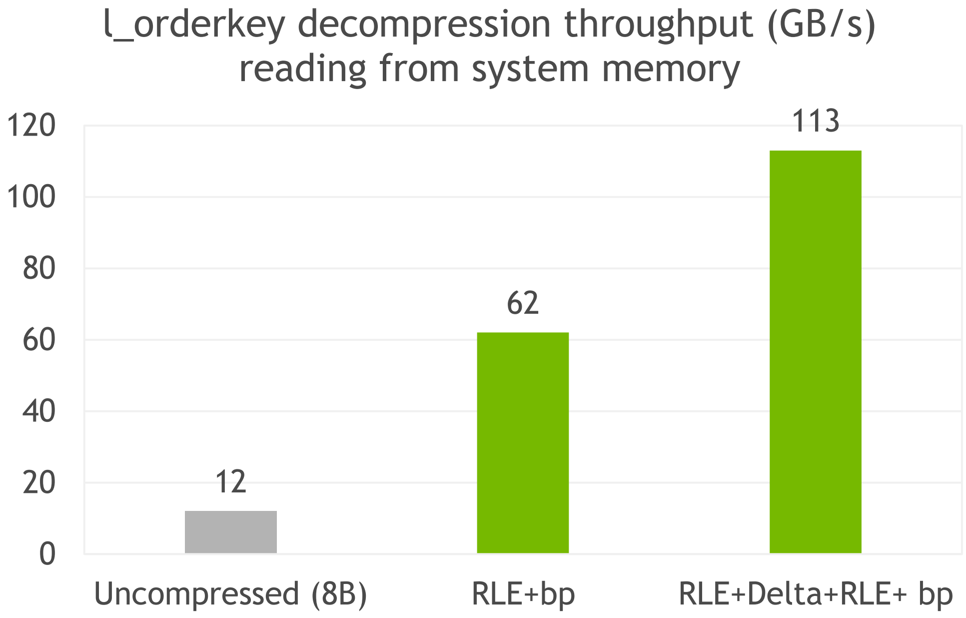
APPLYING COMPRESSION TO TPC-H Q4

Use RLE + Delta + RLE + bit-packing

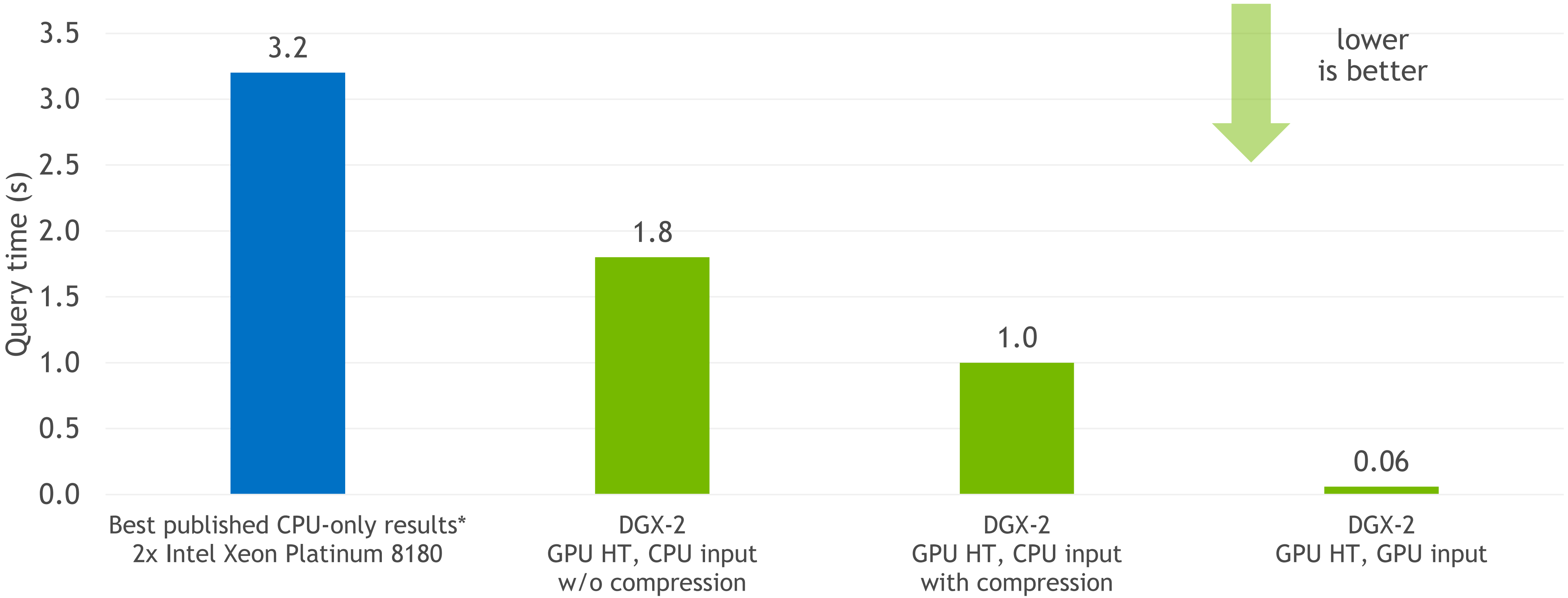
Compression rate for SF1K l_orderkey: **14x**

Multiple streams per GPU

Pipeline decompress & probe kernels



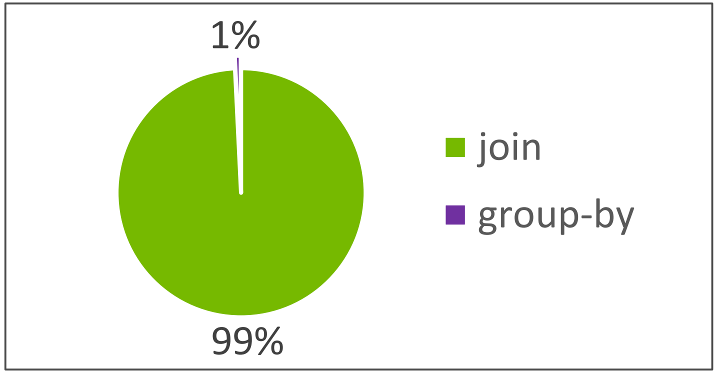
TPC-H SF1000 Q4 RESULTS



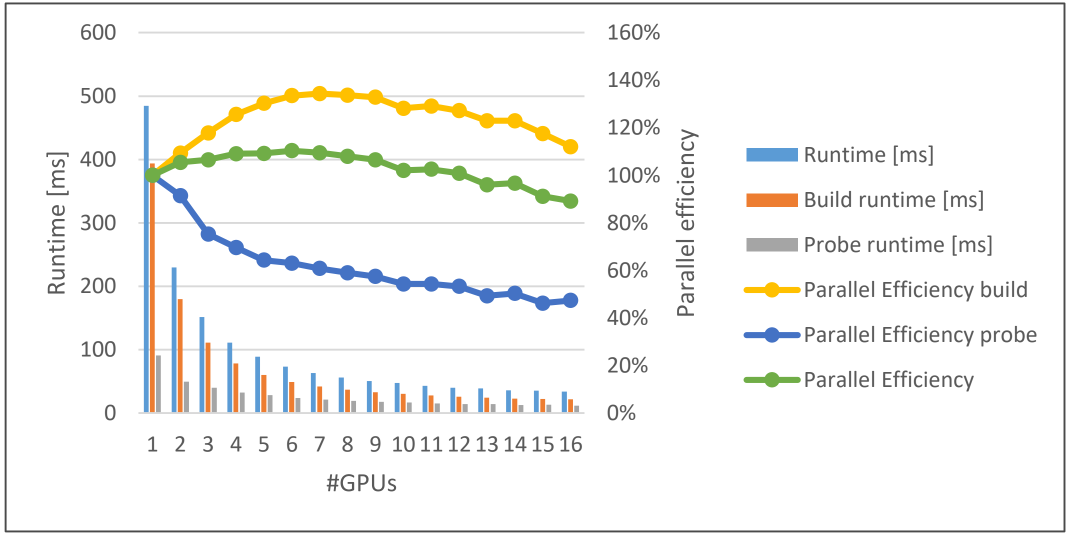
*CPU-only results from: http://www.tpc.org/tpch/results/tpch_result_detail.asp?id=117111701

TAKEAWAY

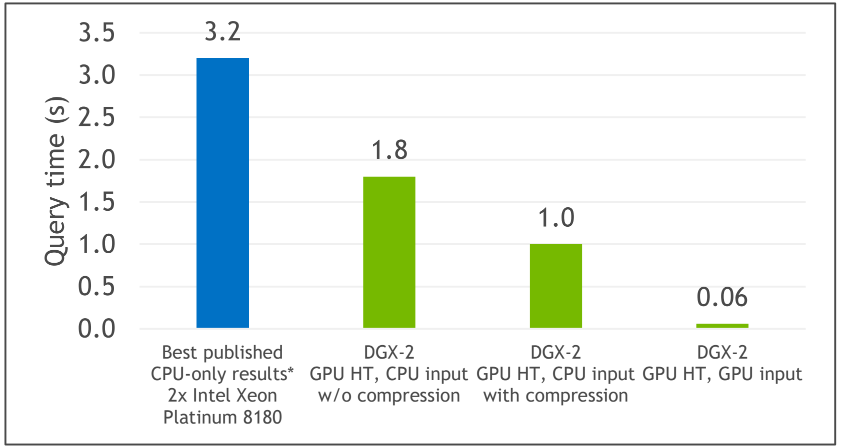
1. Joins is the key bottleneck in OLAP



2. Multi-GPU joins improve perf and enable larger workloads



3. Speed-ups on real analytical queries



DGX-2 can run TPC-H Q4 SF1K in 1 second!
(input data in system memory)

If columns preloaded to GPU memory
Q4 time goes down to just 60ms

NVIDIA®