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**

```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
- ...

---

The TPC-H schema is a structured data model used in database systems. It includes tables for parts, suppliers, line items, orders, and customers, along with various attributes for each table. The schema is designed to test database performance in a standardized way, with specific criteria for query execution.
### RELATIONAL JOIN

<table>
<thead>
<tr>
<th>Lineitem¹</th>
<th>Order²</th>
<th>Join Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_orderkey</td>
<td>o_orderkey</td>
<td>o_orderkey</td>
</tr>
<tr>
<td>23</td>
<td>11</td>
<td>23</td>
</tr>
<tr>
<td>14</td>
<td>23</td>
<td>11</td>
</tr>
<tr>
<td>56</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>11</td>
<td>27</td>
<td>27</td>
</tr>
<tr>
<td>39</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>27</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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\(^1\)

<table>
<thead>
<tr>
<th>l_orderkey</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
</tr>
<tr>
<td>14</td>
</tr>
<tr>
<td>56</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>39</td>
</tr>
<tr>
<td>27</td>
</tr>
<tr>
<td>23</td>
</tr>
</tbody>
</table>

Foreign Key

= Probe inputs

Order\(^2\)

<table>
<thead>
<tr>
<th>o_orderkey</th>
<th>o_orderpriority</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>23</td>
<td>5</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td>29</td>
<td>4</td>
</tr>
</tbody>
</table>

Primary Key

= Probe inputs

Join Results

<table>
<thead>
<tr>
<th>o_orderkey</th>
<th>o_orderpriority</th>
</tr>
</thead>
<tbody>
<tr>
<td>23</td>
<td>5</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>5</td>
</tr>
</tbody>
</table>

Build hash table

\(^1\) after applying predicate “\(l\text{commitdate} < l\text{receiptdate}\)"

\(^2\) after applying predicates “\(o\text{orderdate} \geq \text{date} \left[ \text{DATE} \right]\) and \(o\text{orderdate} < \text{date} \left[ \text{DATE} \right] + \text{interval} \ '3' \text{ month}\)”
JOINS & E2E PERFORMANCE

**CPU** TPC-H Q4 execution breakdown

- Join: 99%
- Group-by: 1%

**GPU** TPC-H Q4 execution breakdown

- Join: 99%
- Group-by: 1%

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

---

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

<table>
<thead>
<tr>
<th></th>
<th>Peak memory bandwidth&lt;sup&gt;1&lt;/sup&gt;</th>
<th>Random 8B access&lt;sup&gt;1&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-end CPU (6-channel DDR4)</td>
<td>120 GB/s</td>
<td>6GB/s</td>
</tr>
<tr>
<td>NVIDIA Tesla V100</td>
<td>900 GB/s</td>
<td>60GB/s</td>
</tr>
</tbody>
</table>

# PERFORMANCE VS. CAPACITY

<table>
<thead>
<tr>
<th></th>
<th>Peak memory bandwidth¹</th>
<th>Random 8B access¹</th>
<th>Memory capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-end CPU (6-channel DDR4)</td>
<td>120 GB/s</td>
<td>6GB/s</td>
<td>1 TB+</td>
</tr>
<tr>
<td>NVIDIA Tesla V100</td>
<td>900 GB/s</td>
<td>60GB/s</td>
<td>32GB</td>
</tr>
</tbody>
</table>

## PERFORMANCE VS. CAPACITY

<table>
<thead>
<tr>
<th></th>
<th>Peak memory bandwidth¹</th>
<th>Random 8B access¹</th>
<th>Memory capacity</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-end CPU (6-channel DDR4)</td>
<td>120 GB/s</td>
<td>6GB/s</td>
<td>1 TB+</td>
</tr>
<tr>
<td>NVIDIA Tesla V100</td>
<td>900 GB/s</td>
<td>60GB/s</td>
<td>32GB</td>
</tr>
<tr>
<td>NVIDIA DGX-2 (16x V100)</td>
<td>16 x 900 GB/s</td>
<td>16x 60GB/s</td>
<td>512 GB</td>
</tr>
</tbody>
</table>

IS A SINGLE V100 FAST/LARGE ENOUGH?

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

**GPU execution breakdown**

<table>
<thead>
<tr>
<th></th>
<th>join</th>
<th>group-by</th>
<th>1%</th>
<th>7.0 s</th>
<th>99%</th>
</tr>
</thead>
</table>

**GPU execution breakdown, compressed data**

<table>
<thead>
<tr>
<th></th>
<th>join</th>
<th>group-by</th>
<th>1%</th>
<th>3.8 s</th>
<th>99%</th>
</tr>
</thead>
</table>

**Hash table sizes**

<table>
<thead>
<tr>
<th>Query</th>
<th>SF1K</th>
<th>SF3K</th>
<th>SF10K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q4</td>
<td>1.5 GB</td>
<td>4.5 GB</td>
<td>15 GB</td>
</tr>
<tr>
<td>Q18</td>
<td>21 GB</td>
<td>63 GB</td>
<td>210 GB</td>
</tr>
<tr>
<td>Q21</td>
<td>10.5 GB</td>
<td>31.5 GB</td>
<td>105 GB</td>
</tr>
</tbody>
</table>

For further speedup or > SF 1000 need 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

<table>
<thead>
<tr>
<th>Build table</th>
<th>GPU 0</th>
<th>GPU 1</th>
<th>GPU 2</th>
<th>GPU #GPU</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0...B_{i-1}</td>
<td>B_{1...B_{i-1}}</td>
<td>B_{2...B_{i-1}}</td>
<td>...</td>
</tr>
<tr>
<td>Probe table</td>
<td>0...P_{i-1}</td>
<td>P_{1...P_{i-1}}</td>
<td>P_{2...P_{i-1}}</td>
<td>P_{g...P-1}</td>
</tr>
</tbody>
</table>
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)

Hash table

GPU 0

if hash to bucket 0..N-1

0..N-1

if hash to bucket N-1..N-1

N-1..N-1

0..P-1

if hash to bucket P-1..P-1

P-1..P-1

GPU 1

N-1..N-1

B-1..B-1

GPU #GPU

N-1..N-1

B-1..B-1

N-1..N-1

N-1..N-1

P-1..P-1

P-1..P-1
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)

Hash table

if hash to bucket 0..N_1-1

if hash to bucket N_1..N_2-1

if hash to bucket N_{p}..N-1

GPU 0

0..N_1-1

0..B_{1}-1

0..P_{1}-1

GPU 1

N_1..N_2-1

B_{1}..B_{2}-1

P_{1}..P_{2}-1

GPU #GPU

N_{p}..N-1

B_{p}..B-1

P_{p}..P-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

![Graph showing memory subsystem metrics](image-url)
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)

- Parallel efficiency build
- Parallel efficiency probe
- Parallel Efficiency

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:

```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;
```

CPU execution breakdown

- join: 1%
- group-by: 99%

semi-join
Q4: INPUT DATA

1.5M rows per SF

<table>
<thead>
<tr>
<th>o_orderkey</th>
<th>o_orderdate</th>
<th>o_orderpriority</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>1996-01-10</td>
<td>2-HIGH</td>
</tr>
<tr>
<td>32</td>
<td>1995-07-16</td>
<td>2-HIGH</td>
</tr>
<tr>
<td>33</td>
<td>1993-10-27</td>
<td>3-MEDIUM</td>
</tr>
<tr>
<td>34</td>
<td>1998-07-21</td>
<td>3-MEDIUM</td>
</tr>
</tbody>
</table>

6M rows per SF

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_commitdate</th>
<th>l_receiptdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1995-10-07</td>
<td>&gt; 1995-08-27</td>
</tr>
<tr>
<td>32</td>
<td>1995-08-20</td>
<td>&lt; 1995-09-14</td>
</tr>
<tr>
<td>32</td>
<td>1995-10-01</td>
<td>&gt; 1995-09-03</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Q4 JOIN: BUILD

orders

GPU 0

32  1995-07-16  2-HIGH

GPU 1

filter (selectivity 3.8%)

\[
o_{\text{orderdate}} \geq \text{date '}[\text{DATE}]' \text{ and } o_{\text{orderdate}} < \text{date '}[\text{DATE}]' + \text{interval '3' month}
\]

compute destination HT partition

insert (o_{\text{orderkey}}, o_{\text{orderpriority}})

into the local HT partition

push (o_{\text{orderkey}}, o_{\text{orderpriority}})

to the remote GPU
Q4 JOIN: PROBE

Filter (selectivity 63%)

\[ l_{\text{commitdate}} < l_{\text{receiptdate}} \]

Compute destination HT partition

Probe against the local HT partition

Match

Remove element from HT (semi-join)

Increment o_orderpriority counter (groupby)
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

GPU hash table (50% HT occupancy)
PERFORMANCE RESULTS ON DGX-2
Q4 SF1000, input distributed in GPU memory

Q4 execution time (s)

# of GPUs

6M rows chunk
PERFORMANCE RESULTS ON DGX-2
Q4 SF1000, input distributed in GPU memory

Q4 parallel efficiency

# of GPUs

6M rows chunk
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

CUDA API overhead
(kernel launches, recording events)
OPTIMIZED CHUNK SIZE ON DGX-2

Q4 SF1000, input distributed in GPU memory

Q4 execution time (s)

# of GPUs

- 6M rows chunk
- 1 chunk per GPU
OPTIMIZED CHUNK SIZE ON DGX-2

Q4 SF1000, input distributed in GPU memory

# of GPUs

Q4 parallel efficiency

- 6M rows chunk
- 1 chunk per GPU
PERFORMANCE RESULTS ON DGX-2

Q4 SF1000, input in system memory

throughput (GB/s)

0 10 20 30 40 50 60

single V100

replicated HT - redundant build, parallel probe

replicated HT - cooperative build, parallel probe

partitioned HT - cooperative build, parallel probe

PCIe3 x16

4x PCIe3 x16
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?

<table>
<thead>
<tr>
<th>8B</th>
<th>2B</th>
<th>2B</th>
</tr>
</thead>
<tbody>
<tr>
<td>l_orderkey</td>
<td>l_commitdate</td>
<td>l_receiptdate</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>32</td>
<td>1995-10-07</td>
<td>1995-08-27</td>
</tr>
<tr>
<td>32</td>
<td>1995-08-20</td>
<td>1995-09-14</td>
</tr>
<tr>
<td>32</td>
<td>1995-10-01</td>
<td>1995-09-03</td>
</tr>
<tr>
<td>34</td>
<td>34</td>
<td>34</td>
</tr>
</tbody>
</table>
IS THIS THE BEST WE CAN DO?

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_commitdate</th>
<th>l_receiptdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1995-10-07</td>
<td>1995-08-27</td>
</tr>
<tr>
<td>32</td>
<td>1995-08-20</td>
<td>1995-09-14</td>
</tr>
<tr>
<td>32</td>
<td>1995-10-01</td>
<td>1995-09-03</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

filters can be executed on the CPU
## IS THIS THE BEST WE CAN DO?

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_commitdate</th>
<th>l_receiptdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1995-10-07</td>
<td>1995-08-27</td>
</tr>
<tr>
<td>32</td>
<td>1995-08-20</td>
<td>1995-09-14</td>
</tr>
<tr>
<td>32</td>
<td>1995-10-01</td>
<td>1995-09-03</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

can be compressed to <8B per key
IS THIS THE BEST WE CAN DO?

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_commitdate</th>
<th>l_receiptdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1995-10-07</td>
<td>1995-08-27</td>
</tr>
<tr>
<td>32</td>
<td>1995-08-20</td>
<td>1995-09-14</td>
</tr>
<tr>
<td>32</td>
<td>1995-10-01</td>
<td>1995-09-03</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

can be compressed to <2B per date
**IS THIS THE BEST WE CAN DO?**

<table>
<thead>
<tr>
<th>l_orderkey</th>
<th>l_commitdate</th>
<th>l_receiptdate</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1995-10-07</td>
<td>1995-08-27</td>
</tr>
<tr>
<td>32</td>
<td>1995-08-20</td>
<td>1995-09-14</td>
</tr>
<tr>
<td>32</td>
<td>1995-10-01</td>
<td>1995-09-03</td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

can be compressed to <2B per date
RLE-DELTA-RLE COMPRESSION

Uncompressed → bit-packing

2,2,2 runs
1,2,3 vals

1,1,1 Delta

2:0,0,0

3:0

3 runs
1 vals

1:0

Compressed → bit-packing

112233

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

Best published CPU-only results*
2x Intel Xeon Platinum 8180

DGX-2
GPU HT, CPU input w/o compression

DGX-2
GPU HT, CPU input with compression

DGX-2
GPU HT, GPU input

Query time (s)

lower is better

*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