Arbeitsgruppe Datenbanken und Software Engineering
Otto-von-Guericke Universität Magdeburg

Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems

Iya Arefyeva, David Broneske, Marcus Pinnecke,
Mudit Bhatnagar, Gunter Saake

GvDB2017, Blankenburg/Harz, 02.06.2017

Motivation

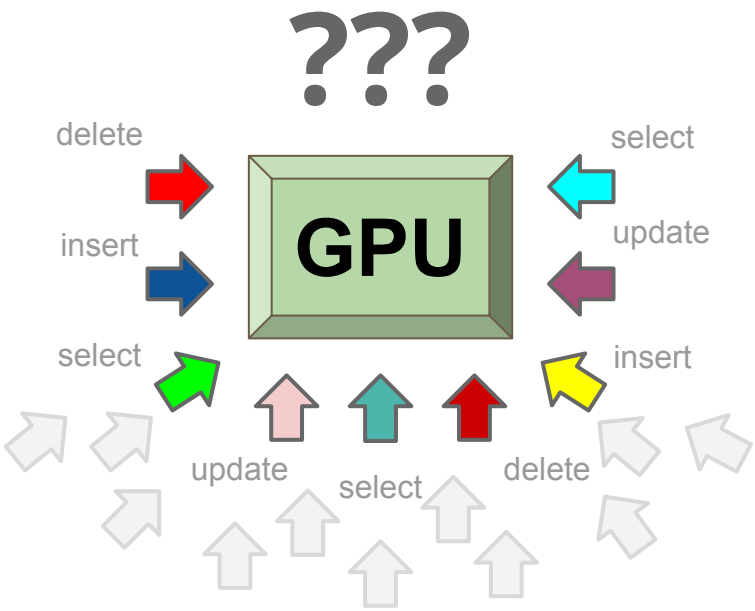
Online analytical processing (OLAP):

- few transactions performed on big chunks of data
- easy to exploit data parallelism

fits perfectly to the GPU style of processing

Online transaction processing (OLTP):

- thousands of transactions within a short period of time
- many small transactions with various operations
- data should be processed as soon as possible due to user interaction



Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

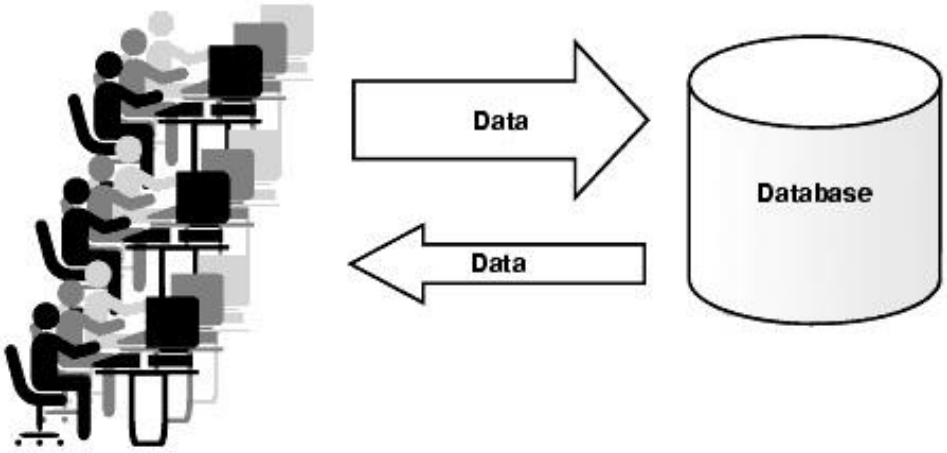
Motivation

GPU accelerated systems for OLAP: GDB [1], HyPE [2], CoGaDB [3], Ocelot [4], H^2TAP [5]

GPU accelerated systems for OLTP: GPUTx [6]

Is the GPU style of processing suitable for OLTP?

What is the best storage model when GPU is used?

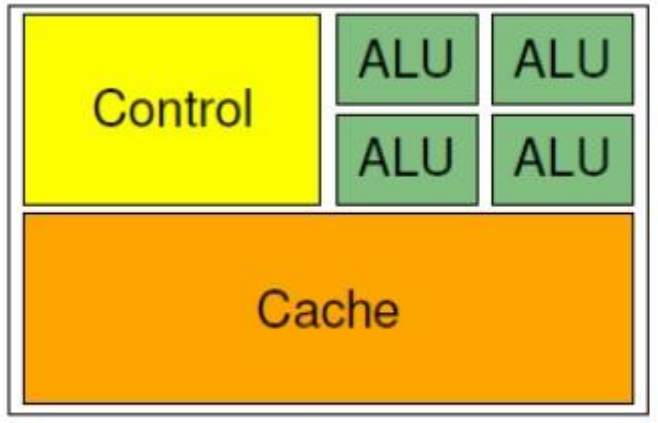


CPU vs. GPU

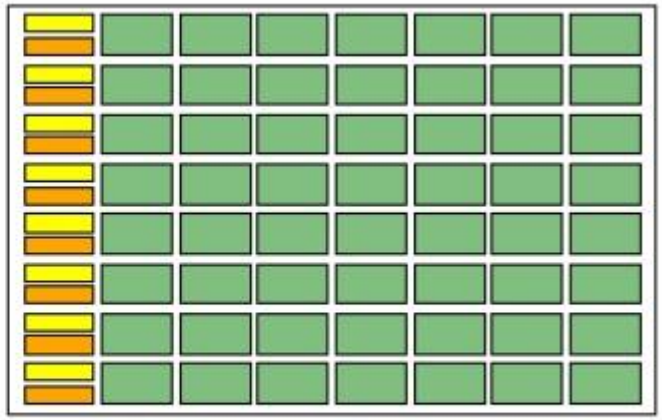
- **ALU** (Arithmetic Logical Unit) is responsible for computing tasks.
- **Control unit** handles synchronization.
- **Cache** keeps frequently accessed data.

- CPU is composed of few cores
  → few threads at a time
- GPU is composed of thousands of cores
  → multiple threads at a time

Well-suited for execution on GPU algorithms: *data parallel* and *data intensive*.



CPU



GPU

Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

GPU computing

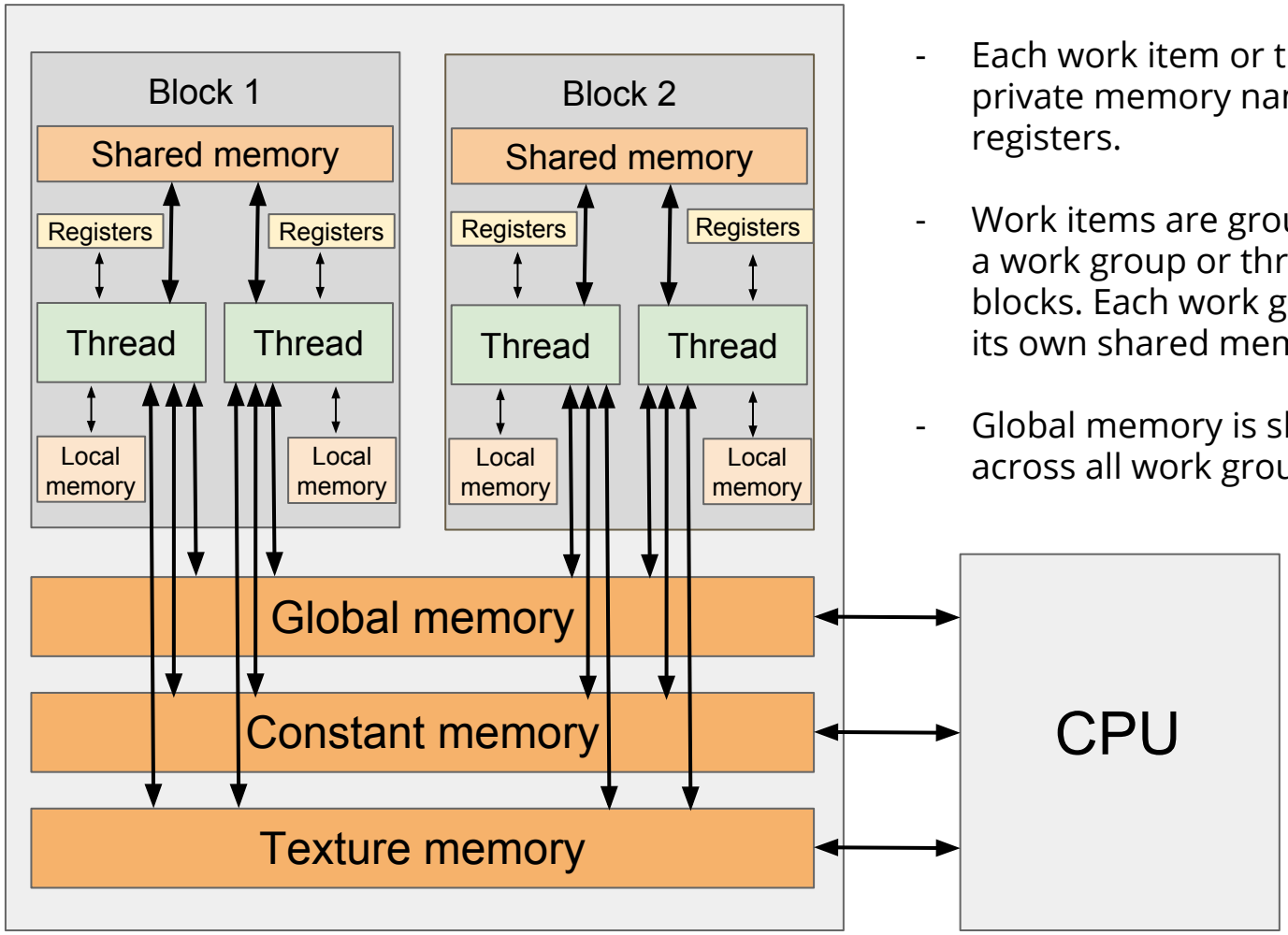
- **Different memory types**: global, shared, local, constant and texture.
- **Coalesced memory access**: to optimize execution behavior, each thread within a work group should access sequential blocks of memory.

global	visible to all threads within the application, and lasts for the duration of the host allocation
shared	visible to all threads within a block and lasts for the duration of the block
local	visible only to the thread that wrote it and lasts only for the lifetime of that thread
constant	read only, used for data that does not change over the course of a kernel execution
texture	read only, improves performance when reads are physically adjacent

- **Communication bottleneck**: data needs to be transferred to GPU and back over a PCIe bus.
- **Bandwidth bottleneck**: the bandwidth of a PCIe bus is lower than the bandwidth of a GPU.

Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

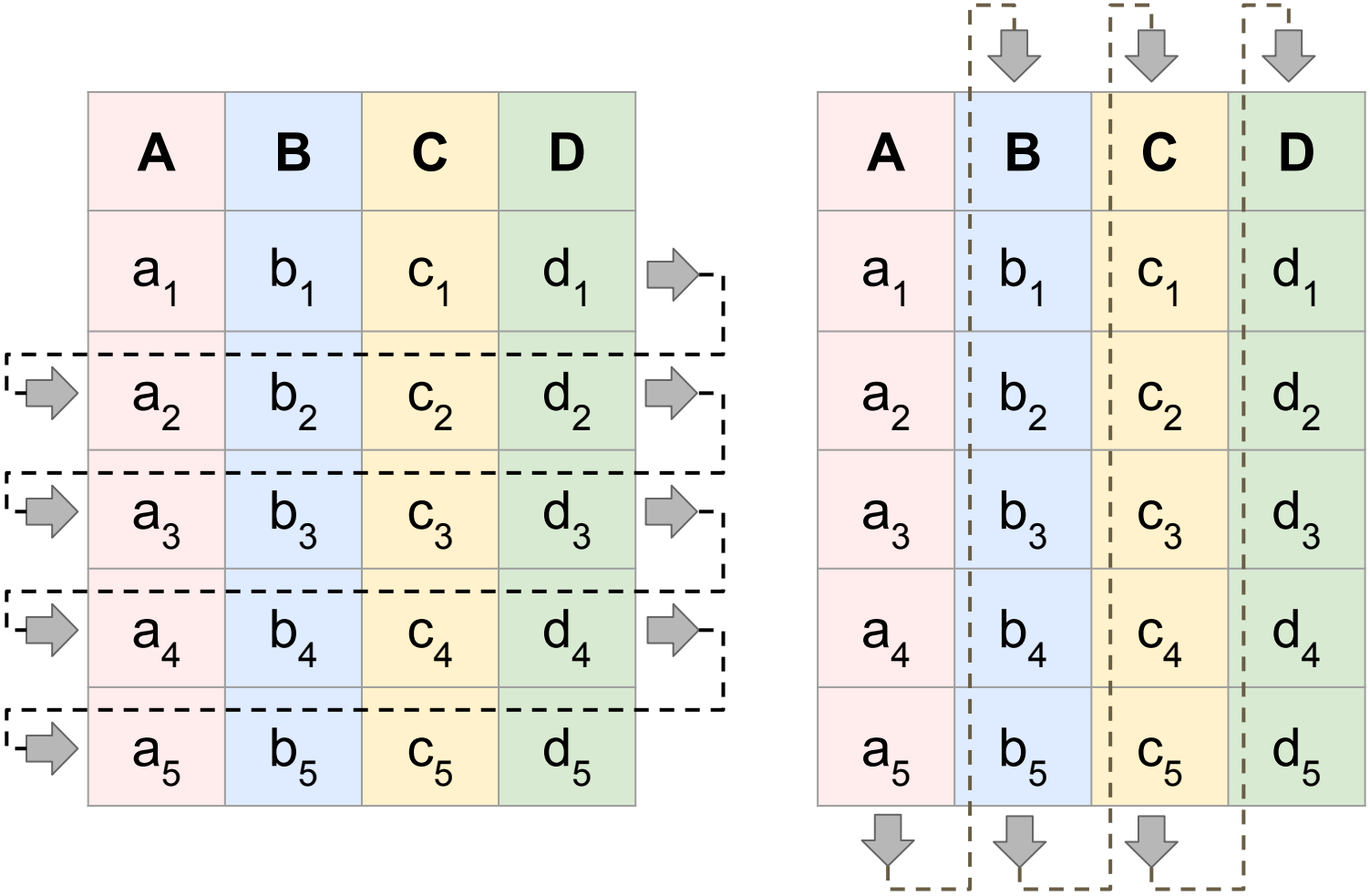
GPU memory types



- Each work item or thread has private memory named registers.
- Work items are grouped into a work group or thread blocks. Each work group has its own shared memory.
- Global memory is shared across all work groups.

Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

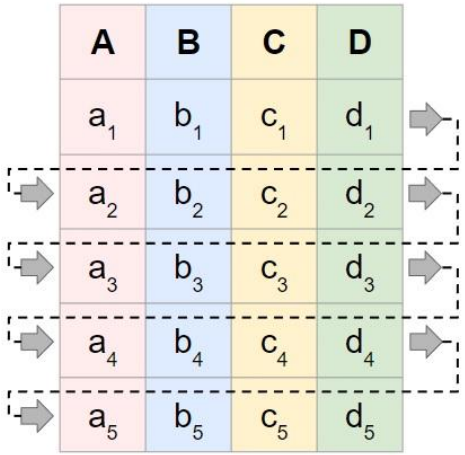
Row store vs. Column store



Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

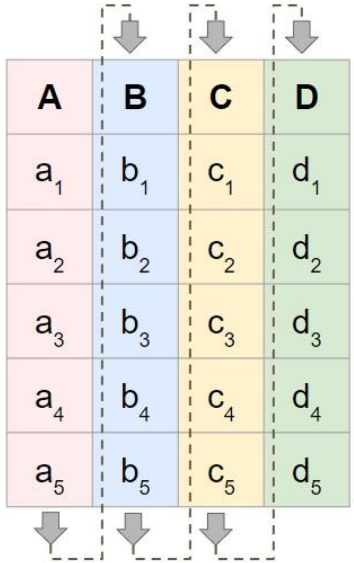
Row store vs. Column store

- Row-wise storage is well suited for operators, that work on all attributes of a tuple.
- Column-wise storage could be beneficial, when only a small subset of the attributes is needed.



Column store for GPU in OLAP [1, 3, 6, 7]:

- allows for coalesced memory access
- has a better compression rate → more data can be stored in the device memory
- less data is transferred when only a subset of the columns is needed

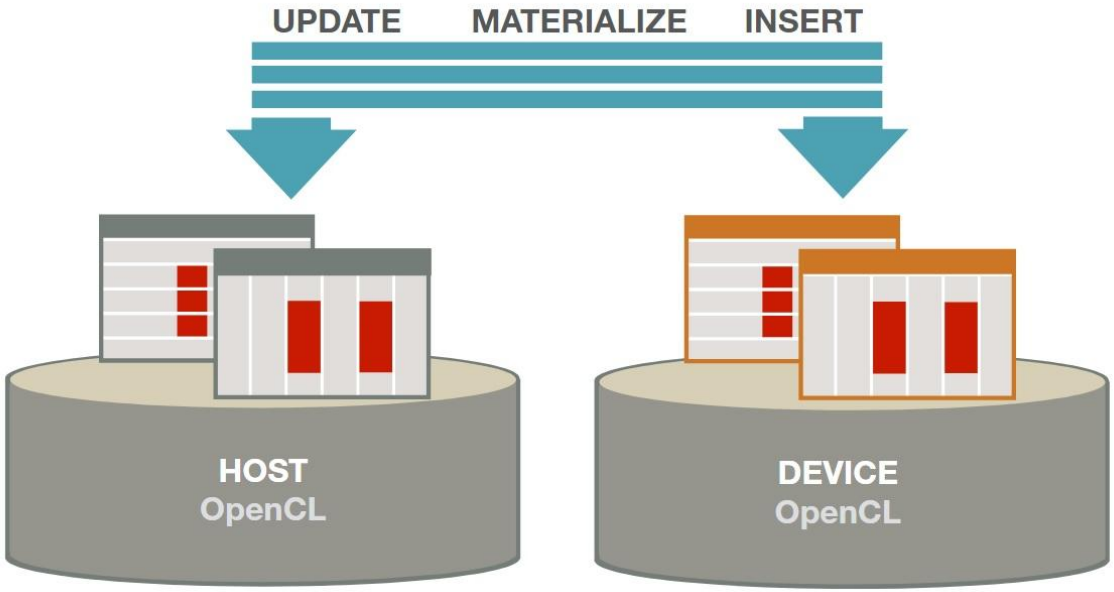


What storage model is the best for a typical access pattern in OLTP?

Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Our contribution

- Implementation of an in-memory database for the TPC-C benchmark [8].
- Implementation of three operators (insert, update and materialize) for row and column store using OpenCL.
- Comparative study of performance of the storage models for CPU and GPU.

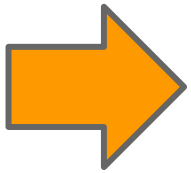


Operators: insert

- Copies fields from the input table to the corresponding fields of the output table.

Input		
1	0.1	“aaa”
2	0.2	“bbb”
3	0.3	“ccc”
4	0.4	“ddd”
5	0.5	“eee”

Output		
undef	undef	undef
undef	undef	undef
undef	undef	undef
undef	undef	undef
undef	undef	undef

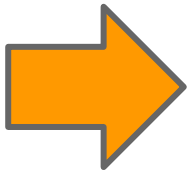


Output		
1	0.1	“aaa”
2	0.2	“bbb”
3	0.3	“ccc”
4	0.4	“ddd”
5	0.5	“eee”

Operators: update

- Attributes of numerical types increased by 10, text fields get replaced by the same text.

Input		
1	0.1	“aaa”
2	0.2	“bbb”
3	0.3	“ccc”
4	0.4	“ddd”
5	0.5	“eee”



Output		
11	10.1	“aaa”
12	10.2	“bbb”
13	10.3	“ccc”
14	10.4	“ddd”
15	10.5	“eee”

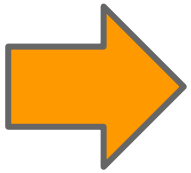
Operators: materialize

- Retrieves the attributes of the selected tuples according to their position and writes them to the output table.

Input		
1	0.1	“aaa”
2	0.2	“bbb”
3	0.3	“ccc”
4	0.4	“ddd”
5	0.5	“eee”

Input		
0	2	4

Output		
undef	undef	undef
undef	undef	undef
undef	undef	undef



Output		
1	0.1	“aaa”
3	0.3	“ccc”
5	0.5	“eee”

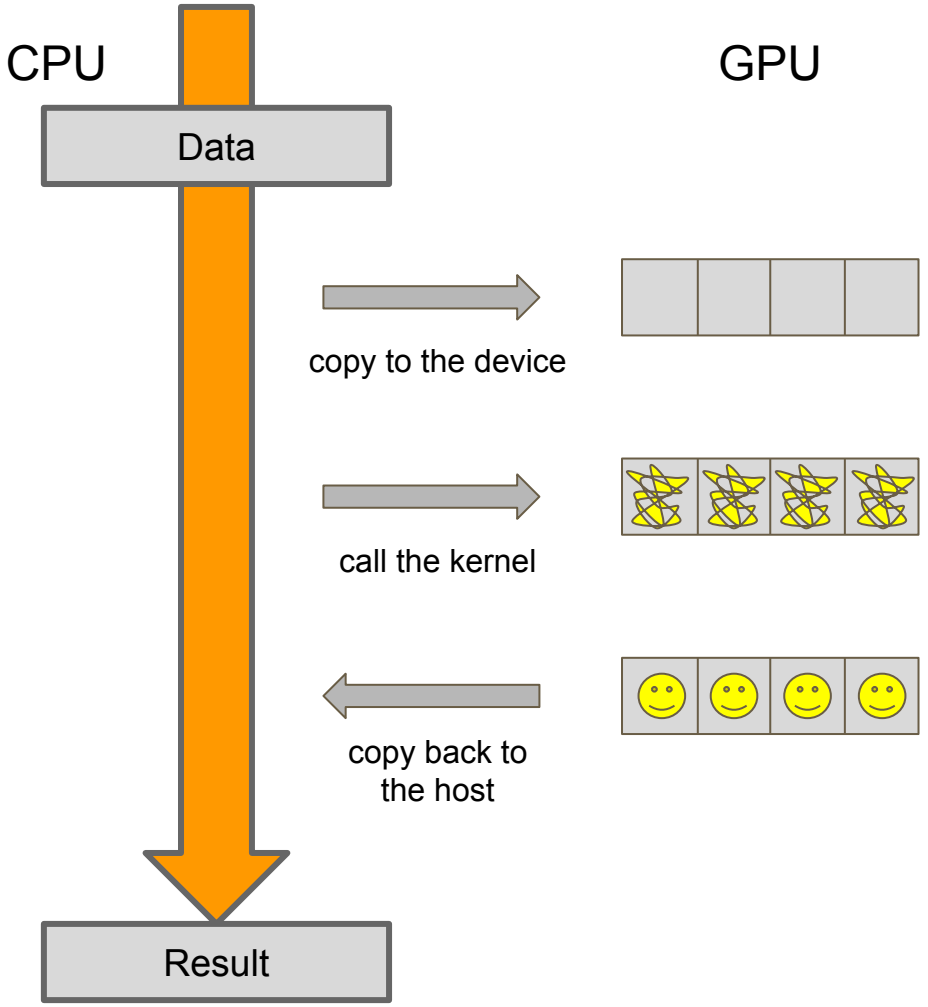
Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Implementation using OpenCL

A **kernel** is a program executed on an OpenCL device.

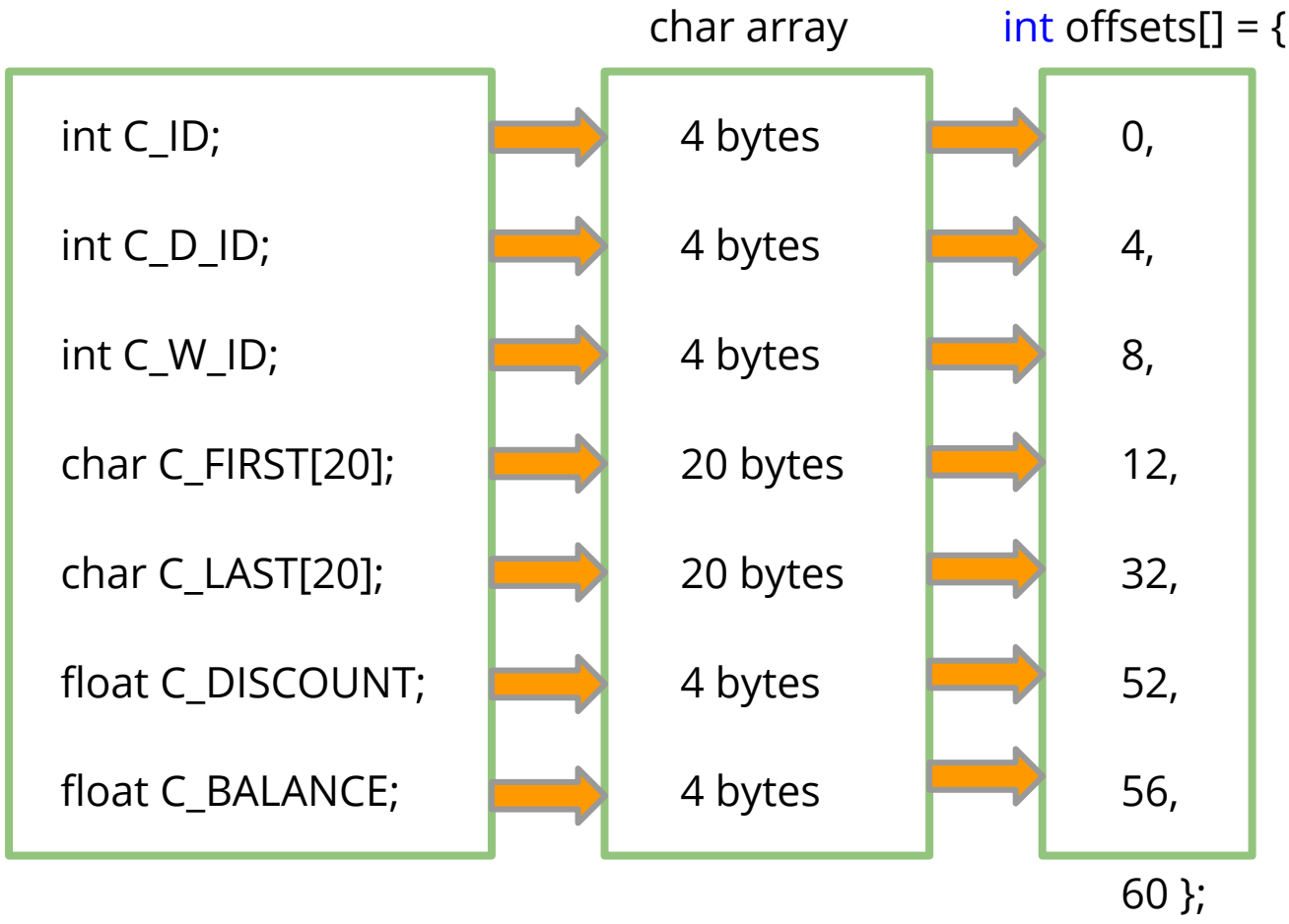
CPU (host) communicates with GPU for executing kernels:

- data to be processed is sent to GPU over a PCIe bus
- CPU invokes the kernel to be executed over the data
- processed data is transferred back to CPU over a PCIe bus



Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Row store



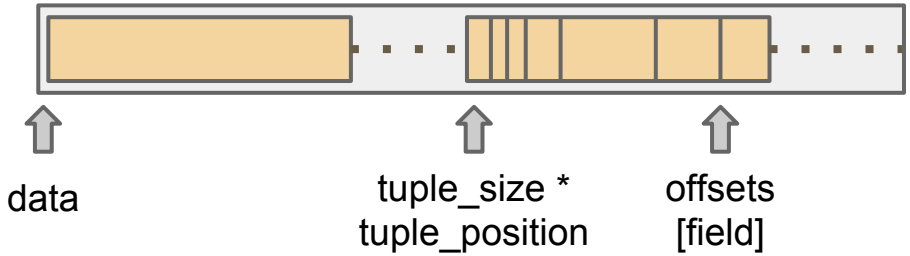
Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Row store: reading values

```
global char *read_value (
  ⇨ global char *data,
  ⇨ int tuple_position,
  ⇨ int field,
  ⇨ global int offsets[],
  ⇨ int num_of_attributes) {

    int tuple_size = offsets[num_of_attributes];
    global char *offset = data + tuple_position * tuple_size;
    offset += offsets[field];
    return offset;

}
```



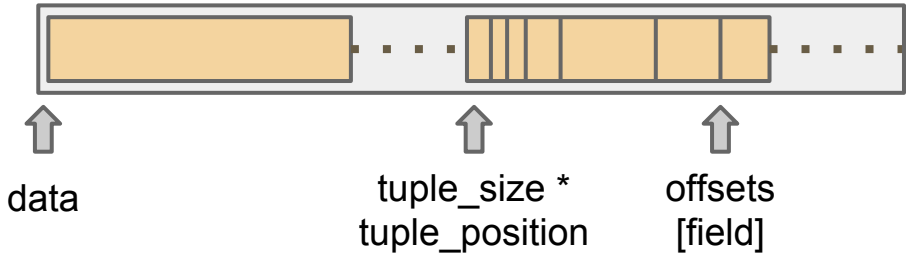
Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Row store: writing to fields

```
global void write_value (
    global char *data,
    int tuple_position,
    char *value,
    int field,
    global int offsets[],
    int num_of_attributes) {

    int tuple_size = offsets[num_of_attributes];
    global char *offset = data + tuple_position * tuple_size;
    offset += offsets[field];
    memcpy(offset, value, (offsets[field + 1] - offsets[field]));

}
```



Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Column store

```
struct CUSTOMER {

    std::vector<int> C_ID;
    std::vector<int> C_D_ID;
    std::vector<int> C_W_ID;
    std::vector<charArray20> C_FIRST;
    std::vector<charArray20> C_LAST;
    std::vector<float> C_DISCOUNT;
    std::vector<float> C_BALANCE;

};
```


Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Row store kernels vs. column store kernels

A column store kernel performs operations on one element:

```
kernel void insert_int(global int* input, global int* output) {
        const int g_id = get_global_id(0);
        output[g_id] = input[g_id];
}
```

A row store kernel performs operations on the whole tuple:

```
kernel void insert_tuple(global char* input, global char* output, global int offsets[],
                                global int num_of_attributes) {
        const int g_id = get_global_id(0);
        for (int i = 0; i < num_of_attributes; i++) {
                write_value(output, g_id,
                                read_value(input, g_id, i, offsets, num_of_attributes),
                                i, offsets, num_of_attributes);
        }
}
```


Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Evaluation

Four combinations:

1. CPU and row store
2. CPU and column store
3. GPU and row store
4. GPU and column store

For varying numbers of tuples:

- Execution time including transfer time
  (transferring data from CPU memory to GPU memory in case of GPU; copying data inside RAM for CPU)
- Execution time excluding transfer time
- Execution time on different fractions of a table's columns

❏ CPU: Intel(R) Core(TM) i5-2500 @3.30 GHz ❏ GPU: NVIDIA GeForce GT 640 ❏ OpenCL 1.2

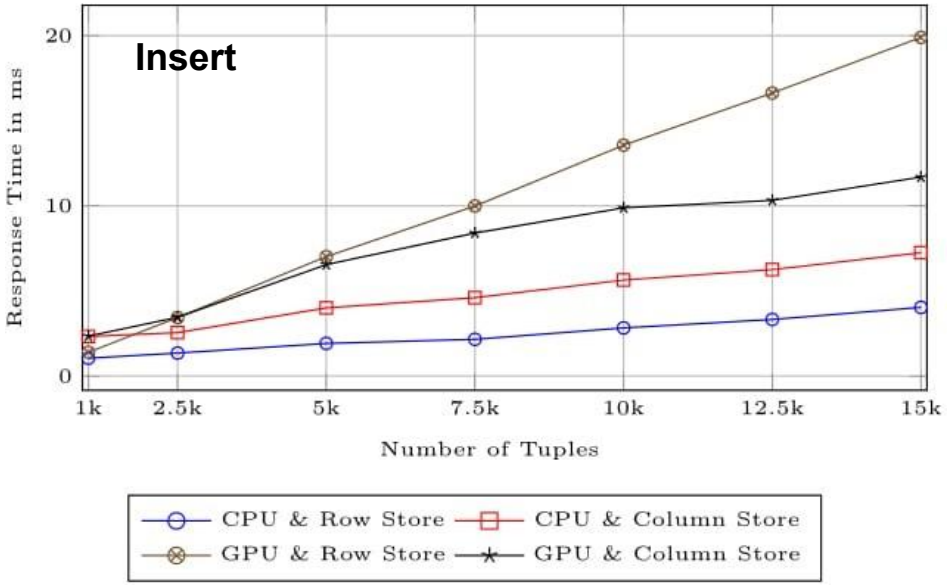
Evaluation

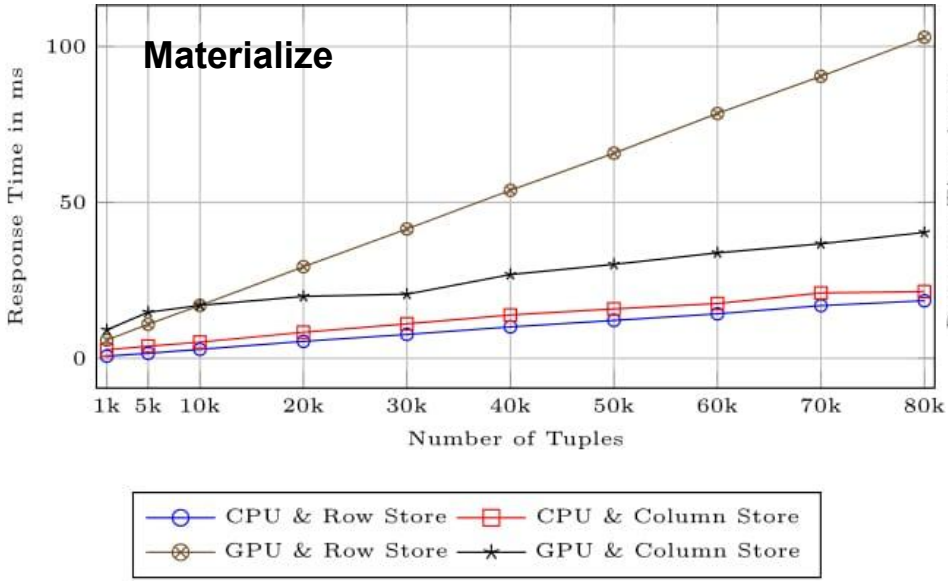
The table CUSTOMER from TPC-C benchmark:

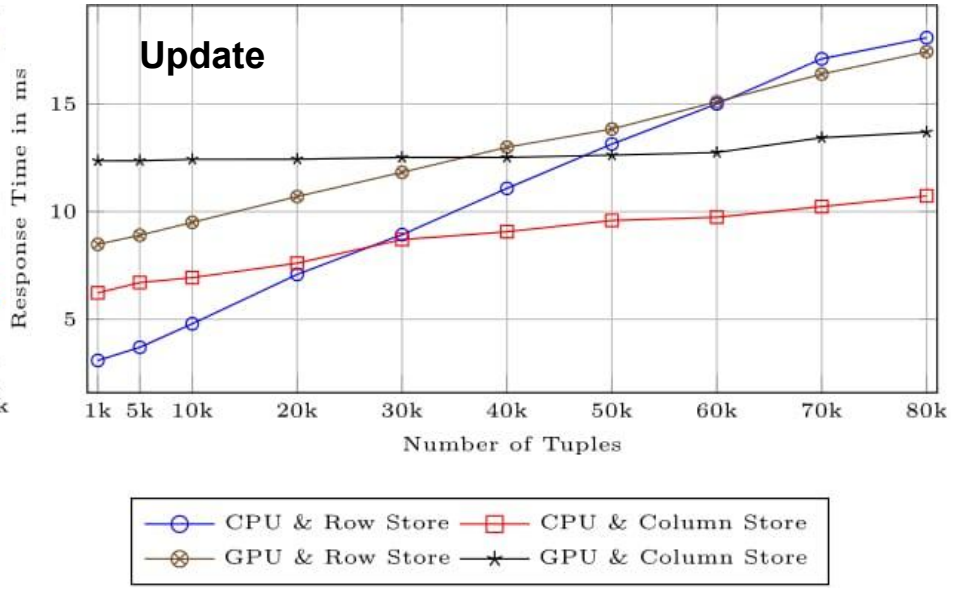
field	type	field	type
C_ID	integer	C_PHONE	char, size 20
C_D_ID	integer	C_SINCE	char, size 10
C_W_ID	integer	C_CREDIT	char, size 2
C_FIRST	char, size 20	C_CREDIT_LIM	float
C_MIDDLE	char, size 2	C_DISCOUNT	float
C_LAST	char, size 20	C_BALANCE	float
C_STREET_1	char, size 20	C_YTD_PAYMENT	float
C_STREET_2	char, size 20	C_PAYMENT_CNT	integer
C_CITY	char, size 20	C_DELIVERY_CNT	integer
C_STATE	char, size 2	C_DATA	char, size 30
C_ZIP	char, size 9		

Execution time including the transfer time

- For insert and materialization CPU and row store outperforms other combinations.
- Row store is beneficial for GPU only on small number of tuples.
- For update the poor performance of row store is caused by the data structure.
- CPU and row store in average performs better than CPU and column store.
- GPU and column store performs better than GPU and row store.





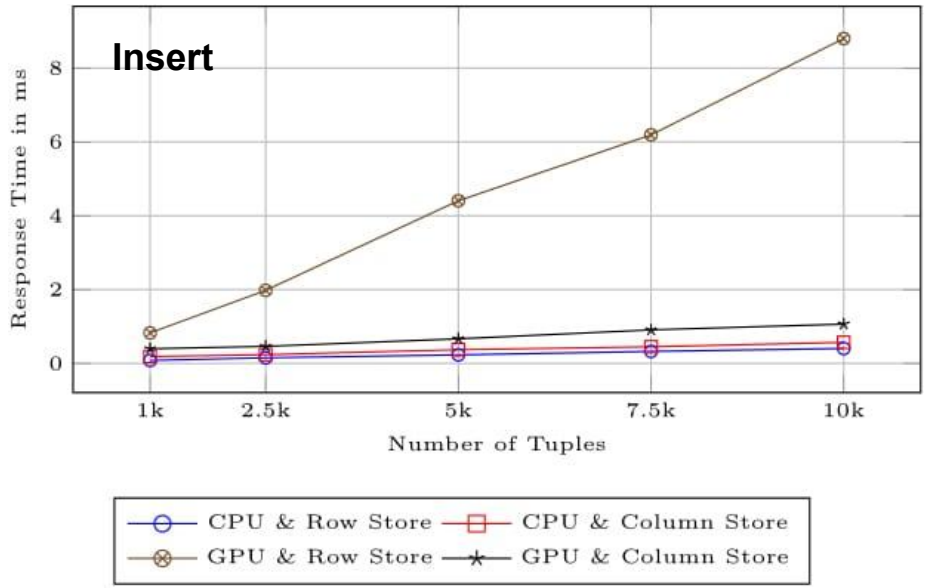


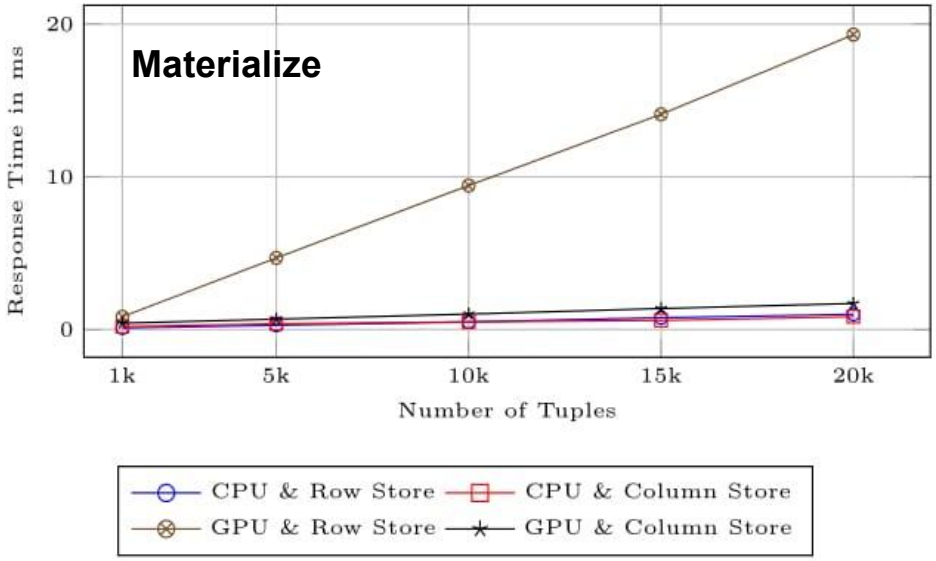
Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

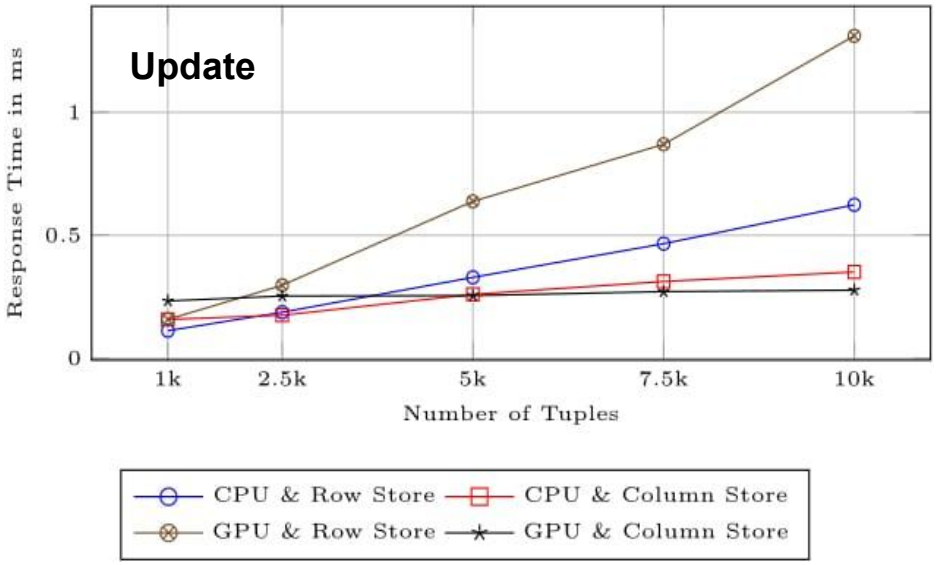
Execution time excluding the transfer time

- For the materialize operator, CPU and row store is outperformed by CPU and column store.
- For the update operator, GPU and column store outperforms CPU and column store on big number of tuples.
- The overall picture stays the same.

- Transfer time does not play a vital role when all the attributes are affected.





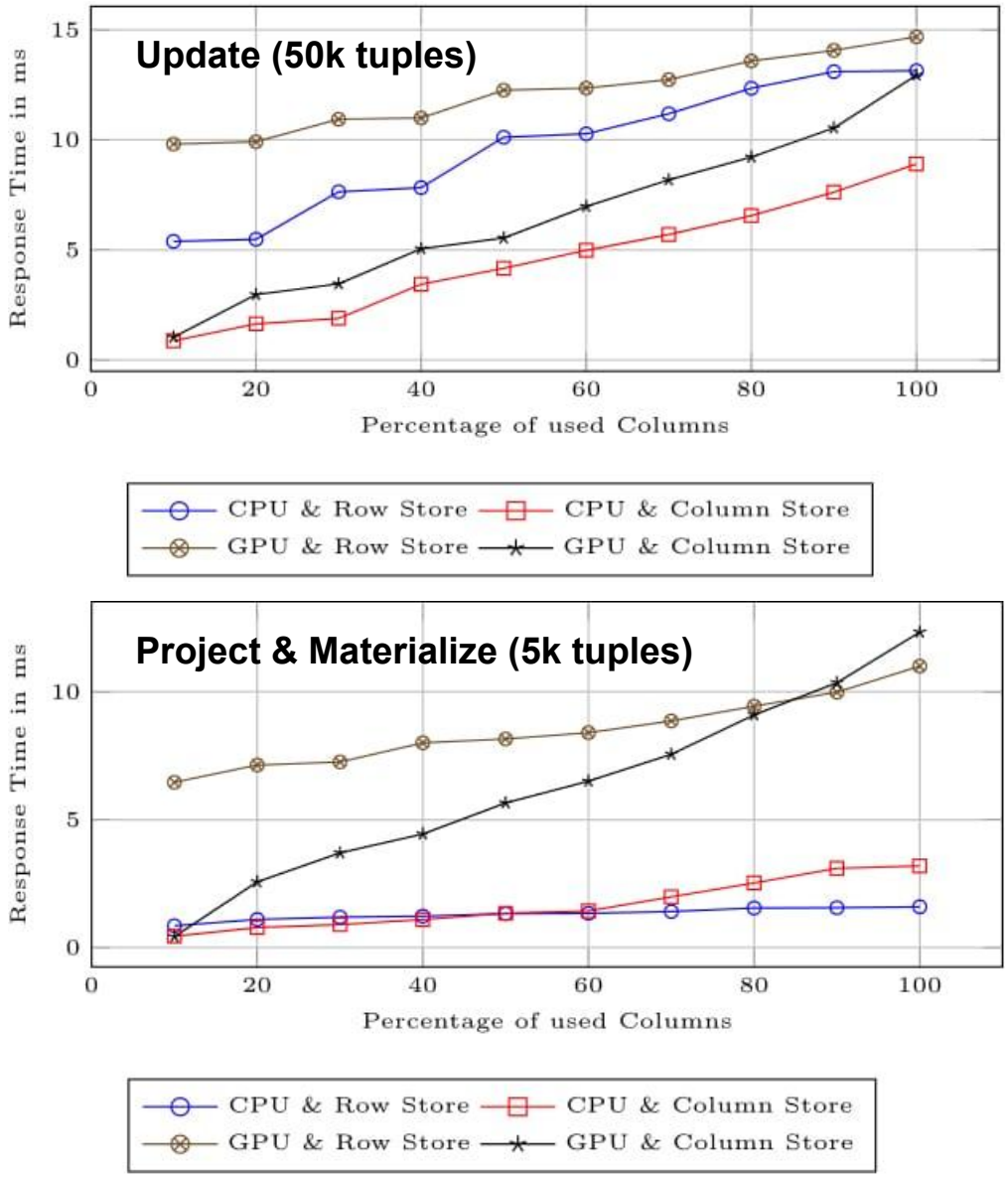


Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Execution time for different fractions of the table's columns

- For column store only the required attributes are transferred.
- For row store the whole table still needs to be transferred.

- Column store outperforms row store on small number of columns.
- Transfer time matters when operators work on only some of the attributes.



Iya Arefyeva, "Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems"

Takeaways

1. Small batch sizes are better than big batches for the row store operator on the GPU.

2. For bigger batch sizes, column store outperforms row store on GPU due to better coalescing and performing less instructions.

3. Transfer times, when batches are small, only plays a vital role for operators that work on a subset of attributes.

4. Column store outperforms row store when only some attributes are updated/retrieved, because only the required data is transferred.

Takeaways

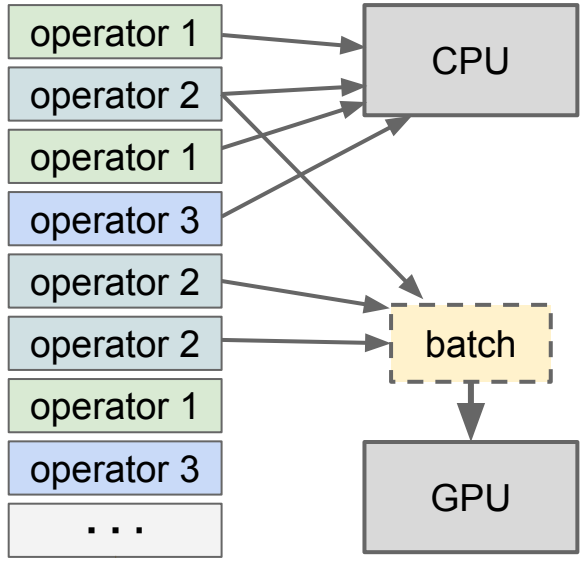
5. CPU performs best with row store and GPU with column store for inserts and materializations.

6. For the update operator, column store is the best storage model for both devices.

7. Kernels for the column store perform less instructions, than kernels for row store. → GPU with row store is not utilized efficiently.

Future work

- Improvement of the implementation of row store.
- Improvement of the implementation of column store (e.g. compression).
- Implementation of further operators.
- Batch processing vs. instance processing on CPU and GPU for intermixed workload.
- Usage of GPU as the primary storage for OLAP, usage of CPU only for the recent data.



Iya Arefyeva, “Column vs. Row Stores for Data Manipulation in Hardware Oblivious CPU/GPU Database Systems”

Thank you!

Questions?

References

1. He, B., Lu, M., Yang, K., Fang, R., Govindaraju, N.K., Luo, Q. and Sander, P.V., 2009. Relational query coprocessing on graphics processors. *ACM Transactions on Database Systems (TODS)*, *34*(4), p.21.
2. Breß, S. and Saake, G., 2013. Why it is time for a HyPE: A hybrid query processing engine for efficient GPU coprocessing in DBMS. *Proceedings of the VLDB Endowment*, *6*(12), pp.1398-1403.
3. Breß, S., 2014. The design and implementation of CoGaDB: A column-oriented GPU-accelerated DBMS. *Datenbank-Spektrum*, *14*(3), pp.199-209.
4. Heimel, M., Saecker, M., Pirk, H., Manegold, S. and Markl, V., 2013. Hardware-oblivious parallelism for in-memory column-stores. *Proceedings of the VLDB Endowment*, *6*(9), pp.709-720.
5. Appuswamy, R., Karpathiotakis, M., Porobic, D. and Ailamaki, A., 2017. The Case For Heterogeneous HTAP. In *8th Biennial Conference on Innovative Data Systems Research* (No. EPFL-CONF-224447).
6. He, B. and Yu, J.X., 2011. High-throughput transaction executions on graphics processors. *Proceedings of the VLDB Endowment*, *4*(5), pp.314-325.
7. Bakkum, P. and Skadron, K., 2010, March. Accelerating SQL database operations on a GPU with CUDA. In *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units* (pp. 94-103). ACM.
8. Transaction Processing Performance Council. TPC-C benchmark revision 5.11. online at http://www.tpc.org/tpcc/