OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING

Diego Tomé
CWI
Amsterdam, The Netherlands
diego.tome@cwi.nl

Tim Gubner
CWI
Amsterdam, The Netherlands
tim.gubner@cwi.nl

Mark Raasveldt*
CWI
Amsterdam, The Netherlands
m.raasveldt@cwi.nl

Eyal Rozenberg
CWI
Amsterdam, The Netherlands
e.rozenberg@cwi.nl

Peter Boncz
CWI
Amsterdam, The Netherlands
peter.boncz@cwi.nl

CWI
Database Architectures

OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING

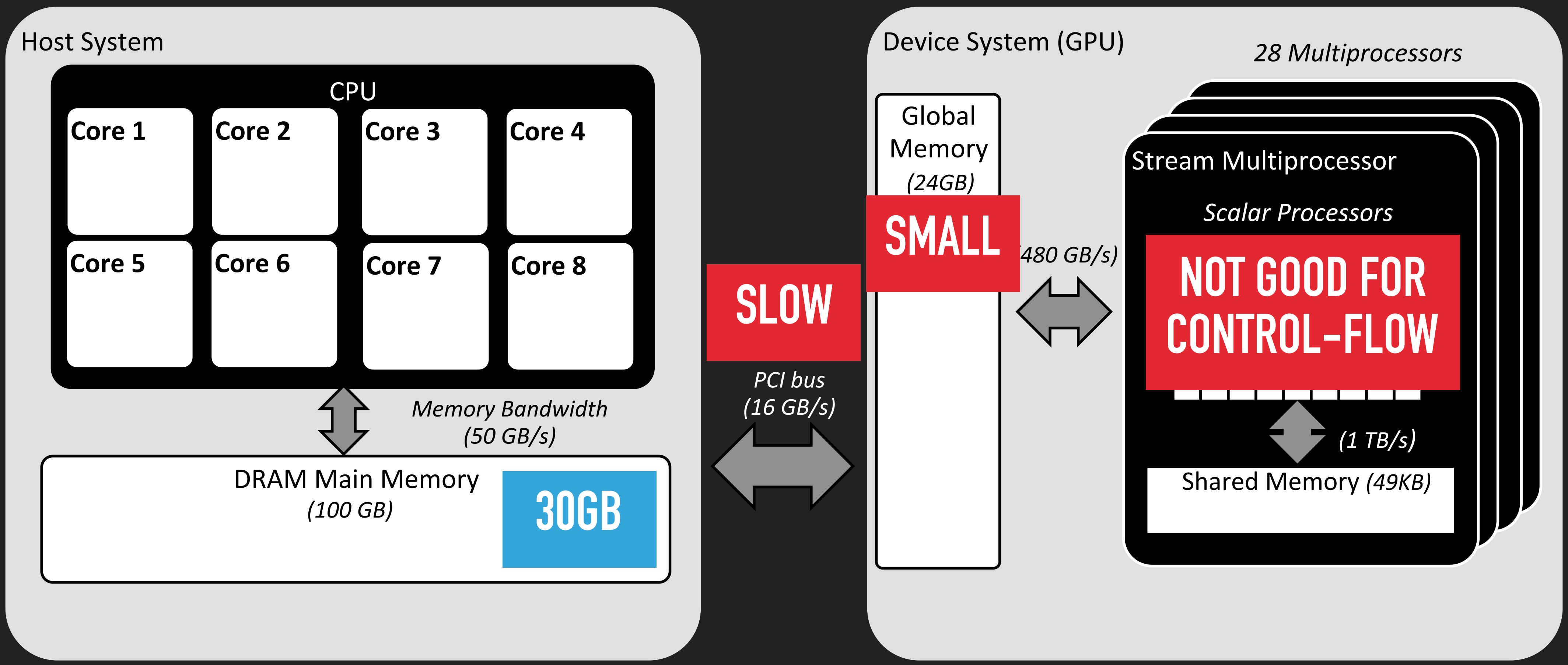
MOTIVATION

GPU AS ACCELERATOR FOR DATABASE SYSTEMS

- Promising: Order-of-magnitude performance gains
  - But only if the input data is on GPU Global Memory
- Unclear how to integrate into full system
- Entire dataset does not fit into GPU global memory

GPU-Accelerated Database Systems: Survey and Open Challenges
Breß, Sebastian and Heimel, Max and Siegmund, Norbert and Bellatreche, Ladjel and Saake, Gunter
Springer, 2014

CHALLENGES IN THE GPU ARCHITECTURE




```
if (expression)
    do something
else
    do something else
```

GPU-CPU CO-PROCESSING SYSTEM

- GPU only system could be worse in many cases
- Solution: GPU-CPU co-processing system
  - Ideally never slower than CPU-only systems
  - Automatically decide between CPU and GPU

WHAT SHOULD GPU-CPU CO-PROCESSING SYSTEM LOOK LIKE?

GPU-CPU CO-PROCESSING SYSTEM

- We take a small step into this design space
- Focus on GROUP BY and AGGREGATION in this system
- In the context of the familiar OLAP scenario: TPC-H Q1
- Investigate the design space and look into trade-offs
  - Without assuming data is in GPU memory!

TPC-H BENCHMARK QUERY 01

```
SELECT
    l_returnflag,
    l_linestatus,
    sum(l_quantity) as sum_qty,
    sum(l_extendedprice) as sum_base_price,
    sum(l_extendedprice * (1 - l_discount)) as sum_disc_price,
    sum(l_extendedprice * (1 - l_discount) * (1 + l_tax)) as sum_charge,
    avg(l_quantity) as avg_qty,
    avg(l_extendedprice) as avg_price,
    avg(l_discount) as avg_disc,
    count(*) as count_order
FROM
    lineitem
WHERE
    l_shipdate <= date '1998-12-01' - interval '90' day
GROUP BY
    l_returnflag,
    l_linestatus
ORDER BY
    l_returnflag,
    l_linestatus;
```

Aggregations

Filter

Grouping

OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING

DESIGN SPACE

DESIGN SPACE - COMPRESSION

Column	SQL Type	Uncompressed Size (bits)	Method	Compressed Size (bits)	Optimum Compressed Size (bits)
l_quantity	INTEGER	32	Null-Suppression	8	6
l_extendedprice	DECIMAL(15,2)	64	Null-Suppression	32	21
l_discount	DECIMAL(15,2)	64	Null-Suppression	8	4
l_tax	DECIMAL(15,2)	64	Null-Suppression	8	4
l_shipdate	DATE	32	Frame-of-Reference + NS	16	12
l_returnflag	VARCHAR(1)	8	Dictionary	2	2
l_linestatus	VARCHAR(1)	8	Dictionary	1	1
Total per Tuple (as bytes)		272		75	50

Table 1: Compression schemes for the columns in lineitem used in TPC-H Query 1

SUB-OPTIMALLY IS 27% OF THE ORIGINAL TUPLE

OPTIMUM IS 18% OF THE ORIGINAL TUPLE

FROM SF = 100 WE TRANSFER 5.625 GB OF DATA

Database Compression on Graphics Processors
Fang, Wenbin and He, Bingsheng and Luo, Qiong
Proceedings of the VLDB Endowment, 2010

Faster Across the PCIe Bus: A GPU Library
Rozenberg, Eyal and Boncz, Peter
Damon@SIGMOD, 2017

DESIGN SPACE – HASHING VS. SORTING

- Hash Function on group keys
- Sorting data, by order of the group keys
- Previous work shows that hash-based performs better
- Sorting is better only with a very high amount of groups

HASH-BASED

Optimizing GPU-accelerated Group-By and Aggregation
Tomas Karnagel and Ren’e Muller and Guy M. Lohman
ADMS@VLDB, 2015

Sort vs. Hash Revisited: Fast Join Implementation on Modern Multi-core CPUs
Kim, Changkyu et. al.
Proceedings of the VLDB Endowment, 2009

DESIGN SPACE – HASH TABLE DESIGN

- Collision resolution
  - Following a chain to find a position
  - Continued probing of subsequent table cells
  - Re-applying the hash function
- Collision Avoidance
  - Use of an injective function as hash **PERFECT HASH FUNCTION**

Efficient Hash Tables on the Gpu
Alcantara, Dan Anthony Feliciano
PhD thesis University of California at Davis, 2011

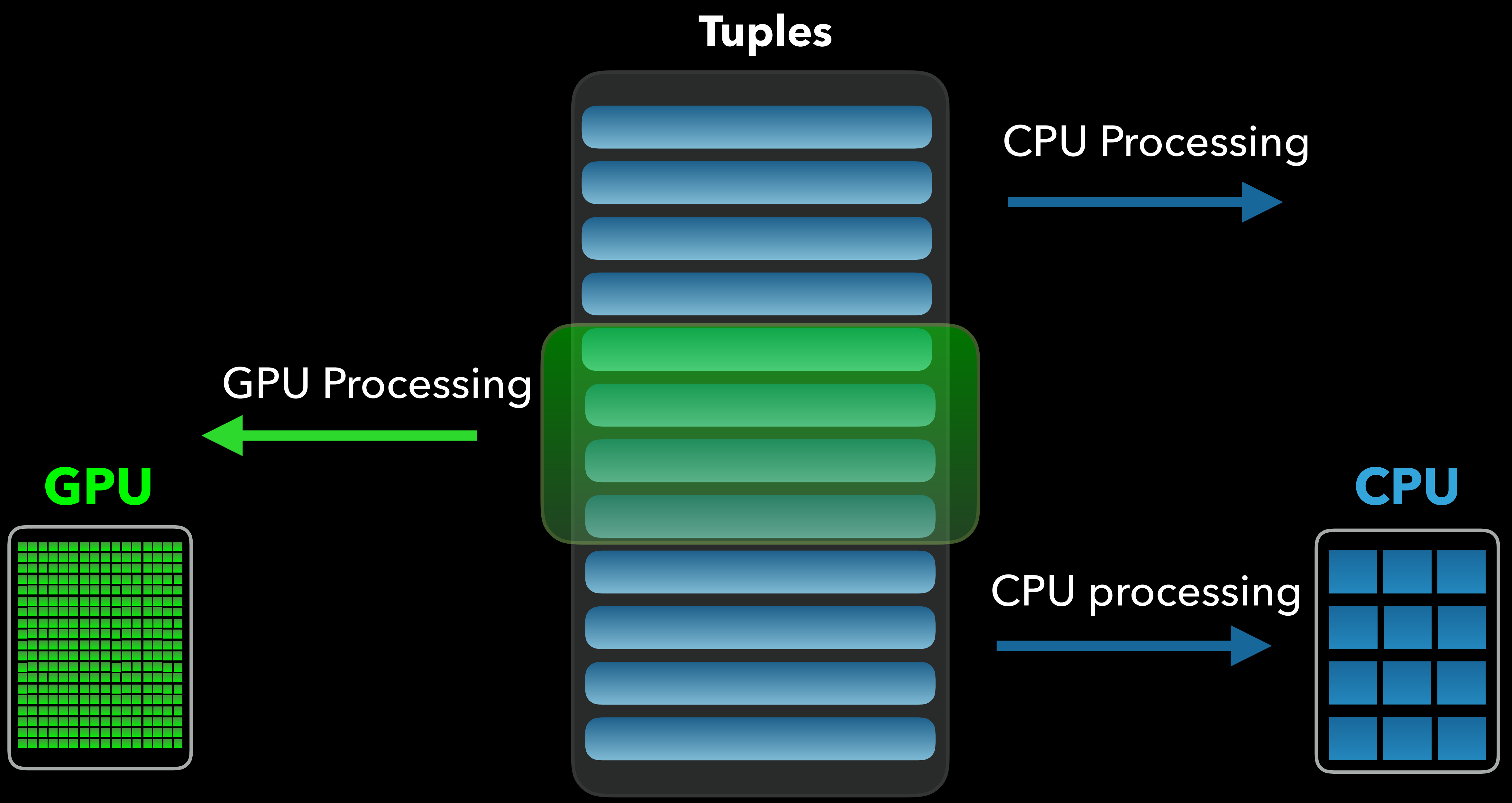
Hash, displace, and compress
Belazzougui, D. et. al.
Springer, 2009

DESIGN SPACE - HASH TABLE PLACEMENT

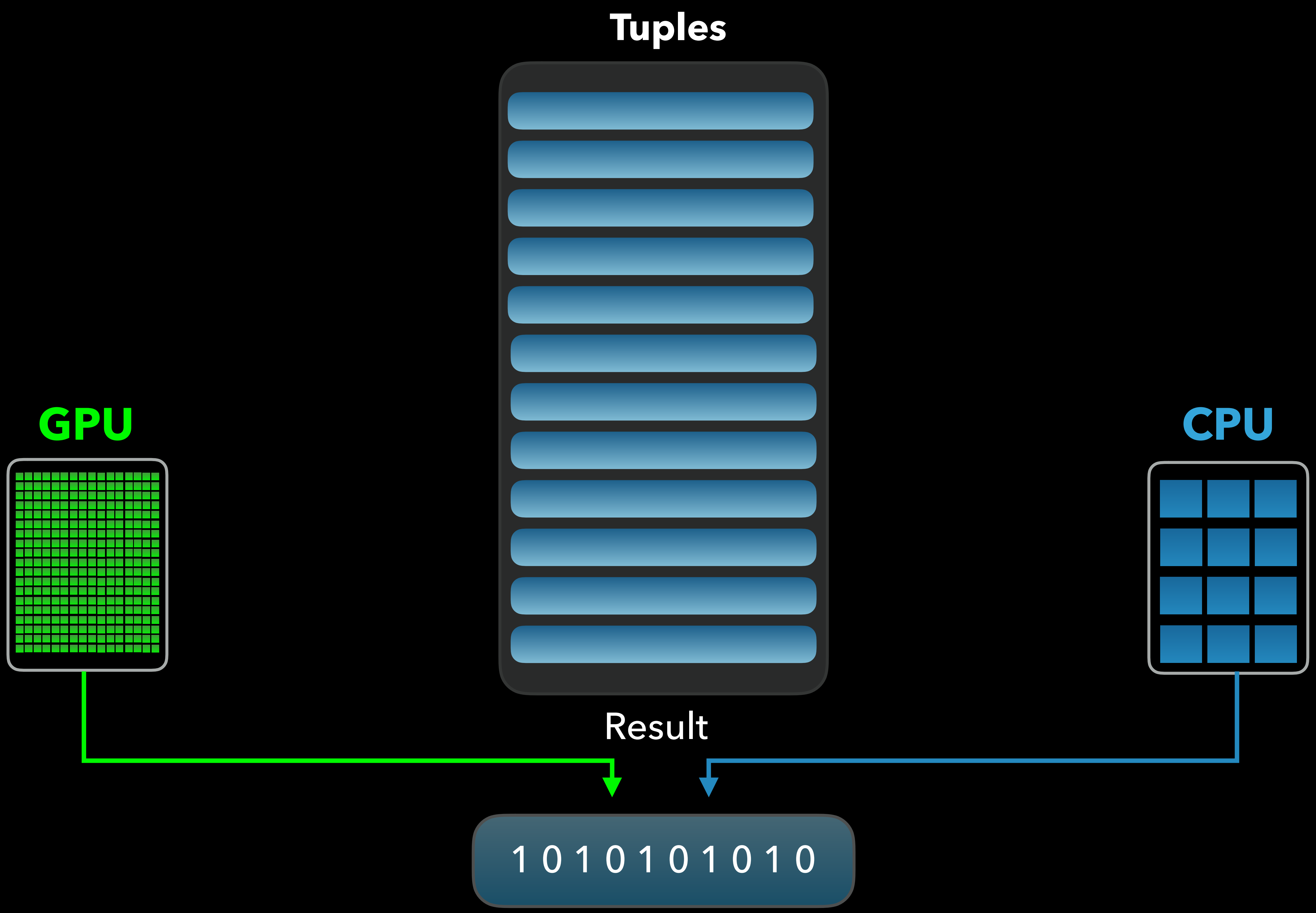
- **Caches** are not coherent
- **Registers** are bigger than **Shared memory**
- **Shared memory** is organized into 32 interleaved banks
- Excessive use of **Registers** is translated to **Global memory** access

EXPLORE THE WHOLE HIERARCHY

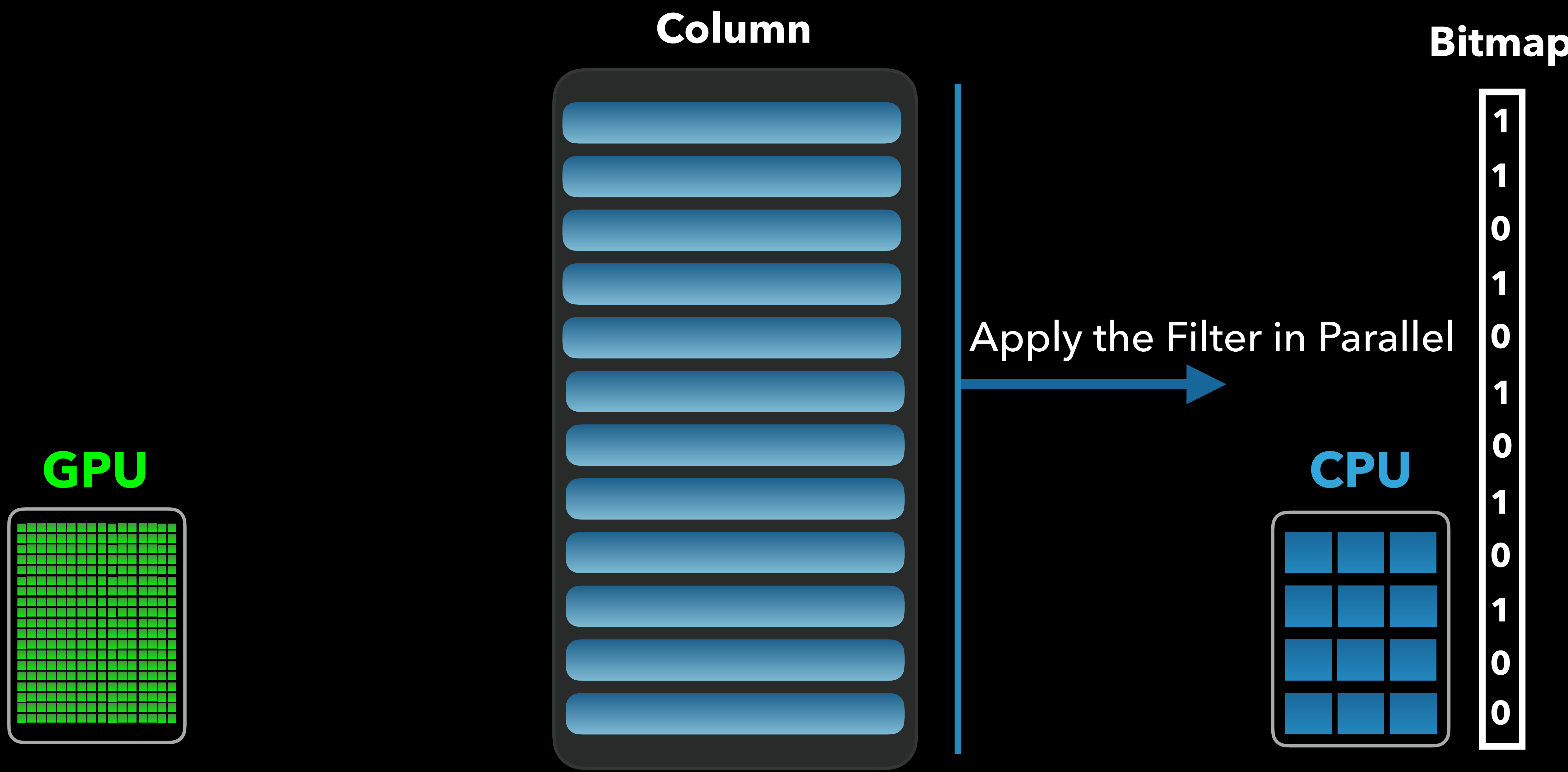
How Data parallel Co-processing Works?



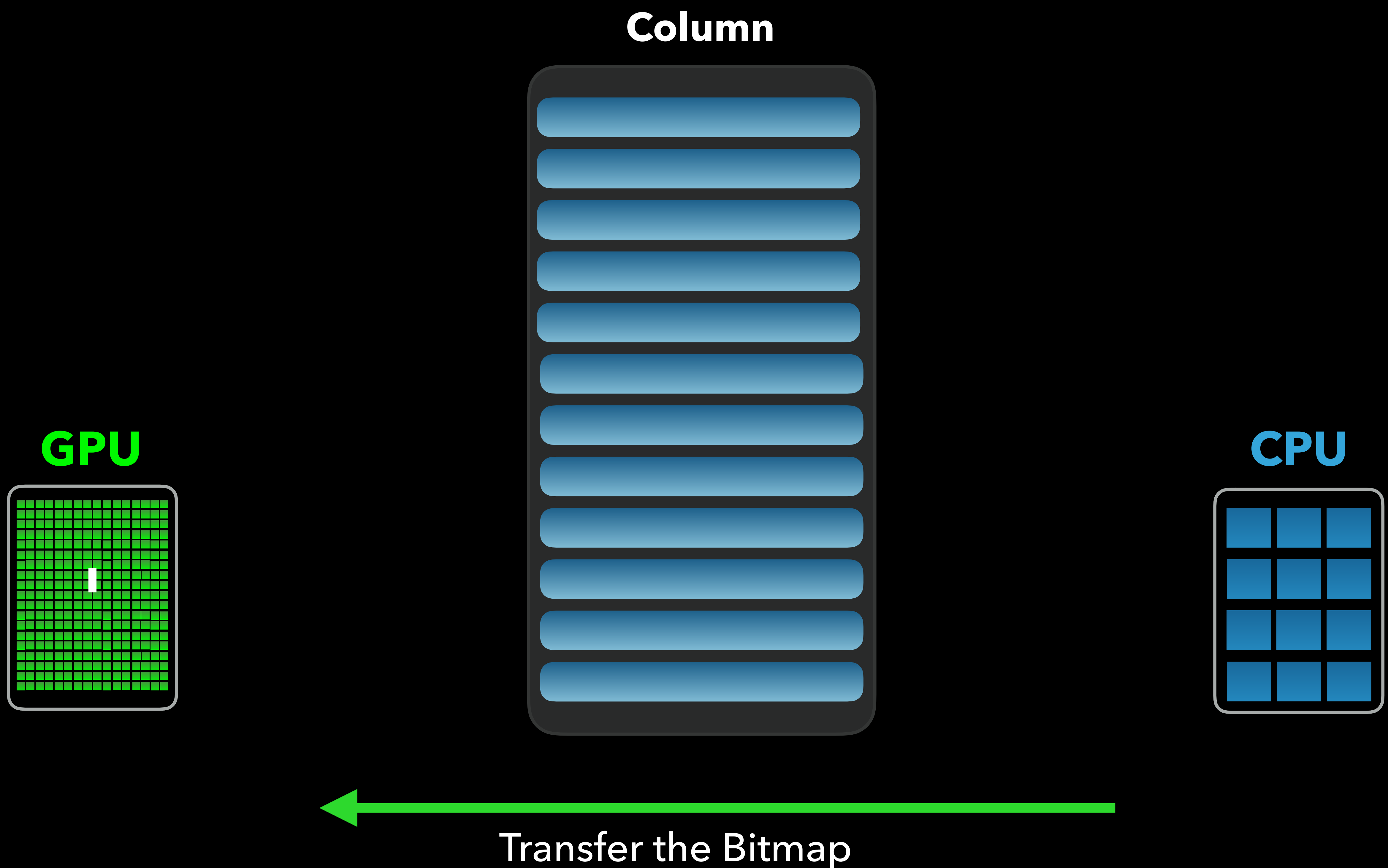
How Data parallel Co-processing Works?



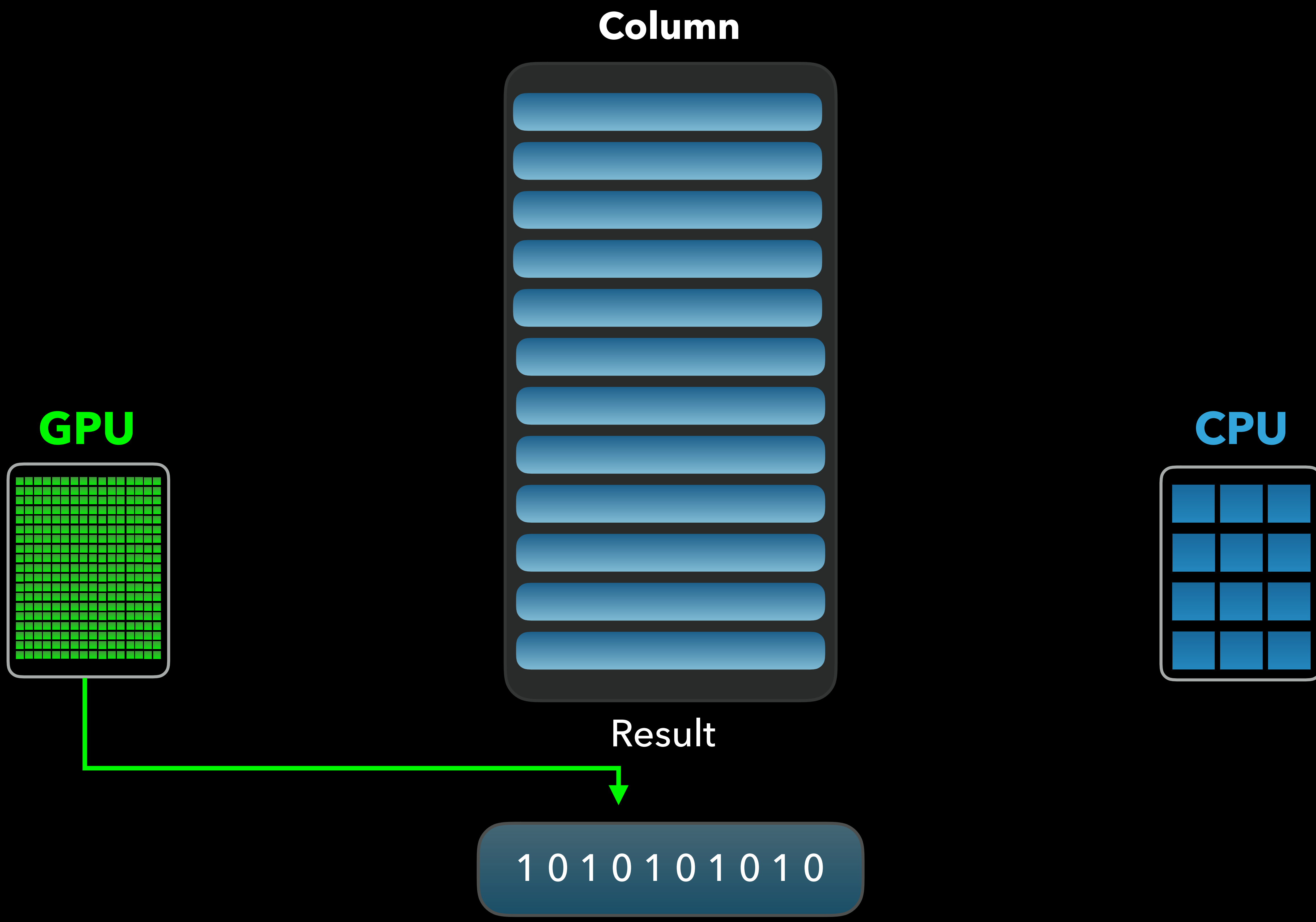
How Filter Pre-computation Works?



How Filter Pre-computation Works?



How Filter Pre-computation Works?



DESIGN SPACE - SUMMARY

Design Space	Implementation
Compression	27% of the original tuple = 5.625 GB (SF 100)
Group-by and aggregation	Hash based implementation
Hash table design	Collision avoidance with a perfect hash function
Hash table placement	Explore the whole GPU hierarchy
Co-processing	Data Parallel and Filter Pre-computation

OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING

EXPERIMENTS

EXPERIMENTS - IMPLEMENTATION FLAVORS

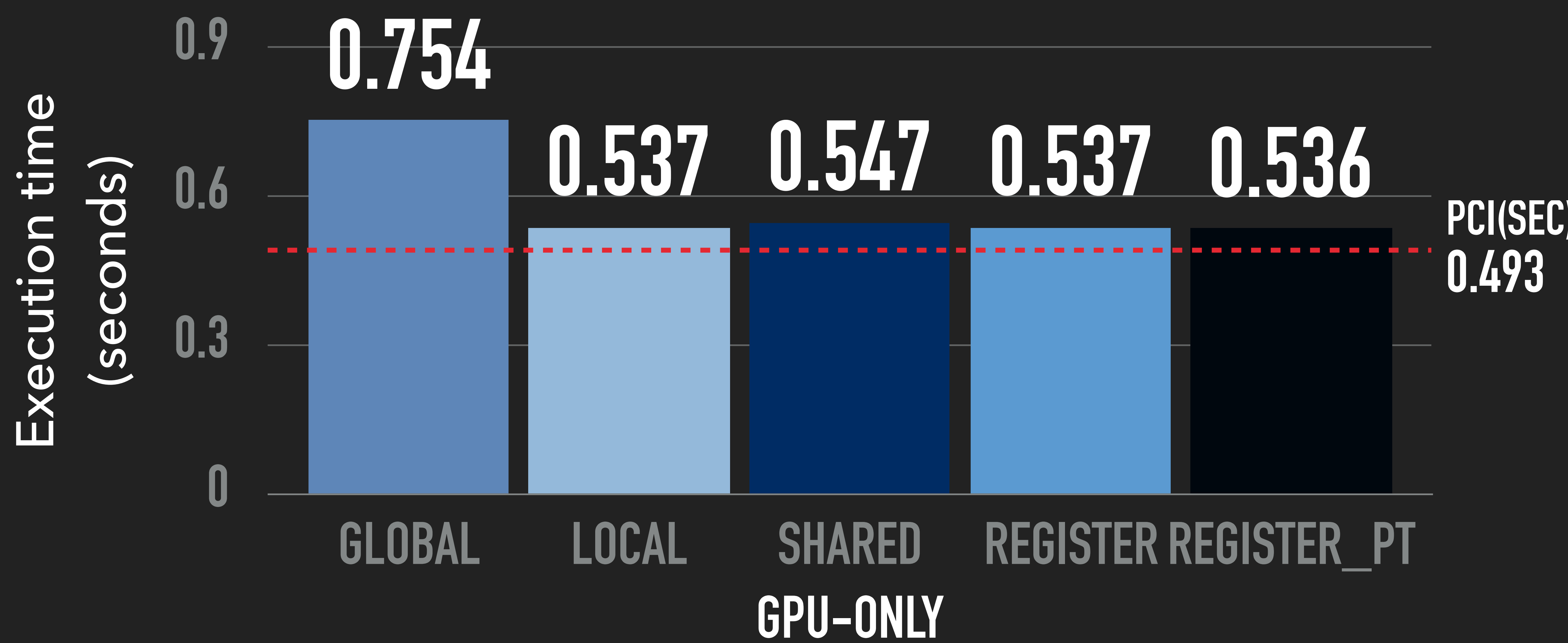
- Global - A single hash table in Global GPU memory
- Local - A hash table for each working thread in local memory
- Shared_memory - A hash table for each working thread in shared memory
- In_registers - A separate hash table for each working thread in its registers
- In_register_per_thread - A single table cell per working thread in register

EXPERIMENTS - CO-PROCESSING

- CPU implementation adapted from previous work
- We introduce Morsel-driven model of parallelized execution
- NUMA locality (One single socket)
- Co-processing with Filter Pre-computation and Data Parallelism.

Exploring Query Compilation Strategies for JIT, Vectorization and SIMD
Tim Gubner and Peter A. Boncz
ADMS@VLDB, 2017

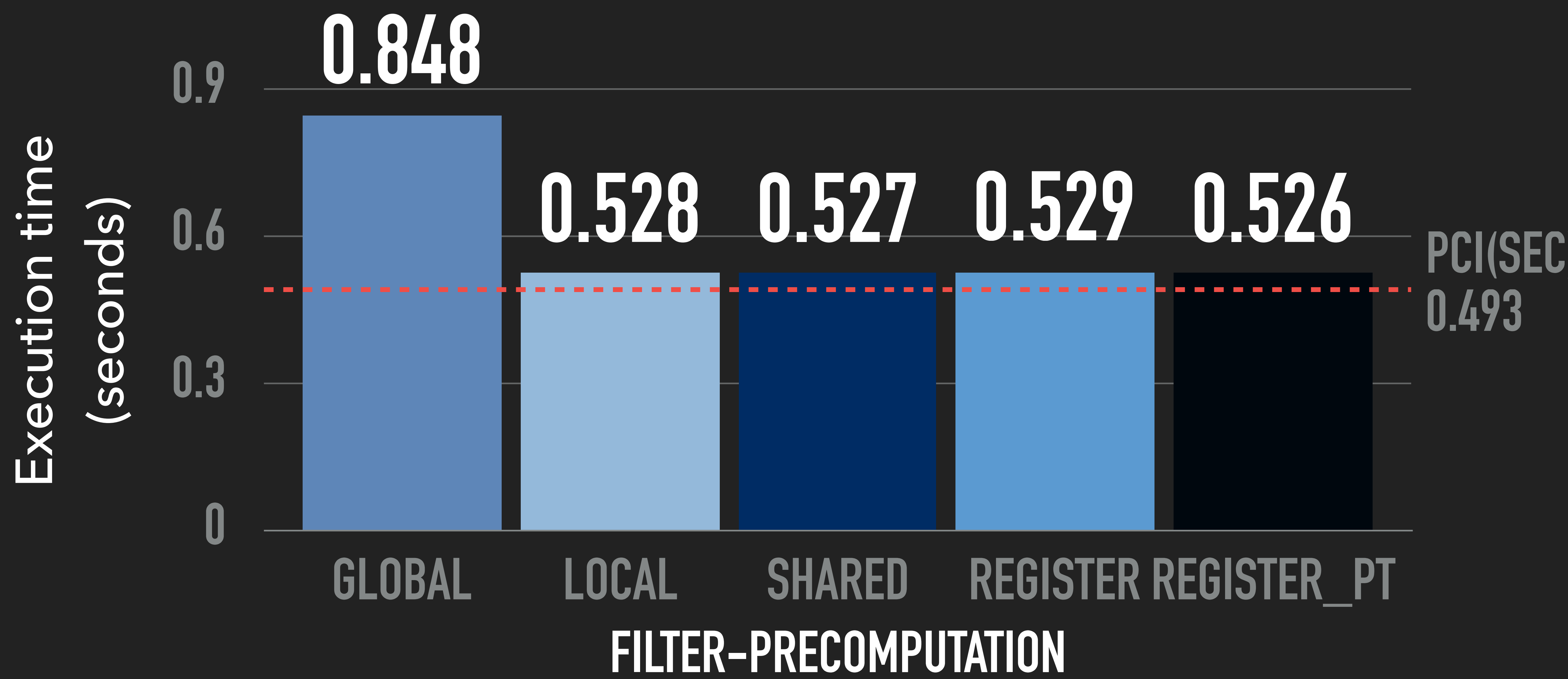
RESULTS TPC-H QUERY 01 - SF 100



Overlapping of computation and transfer (Asynchronous)

Global is penalized with atomic operations

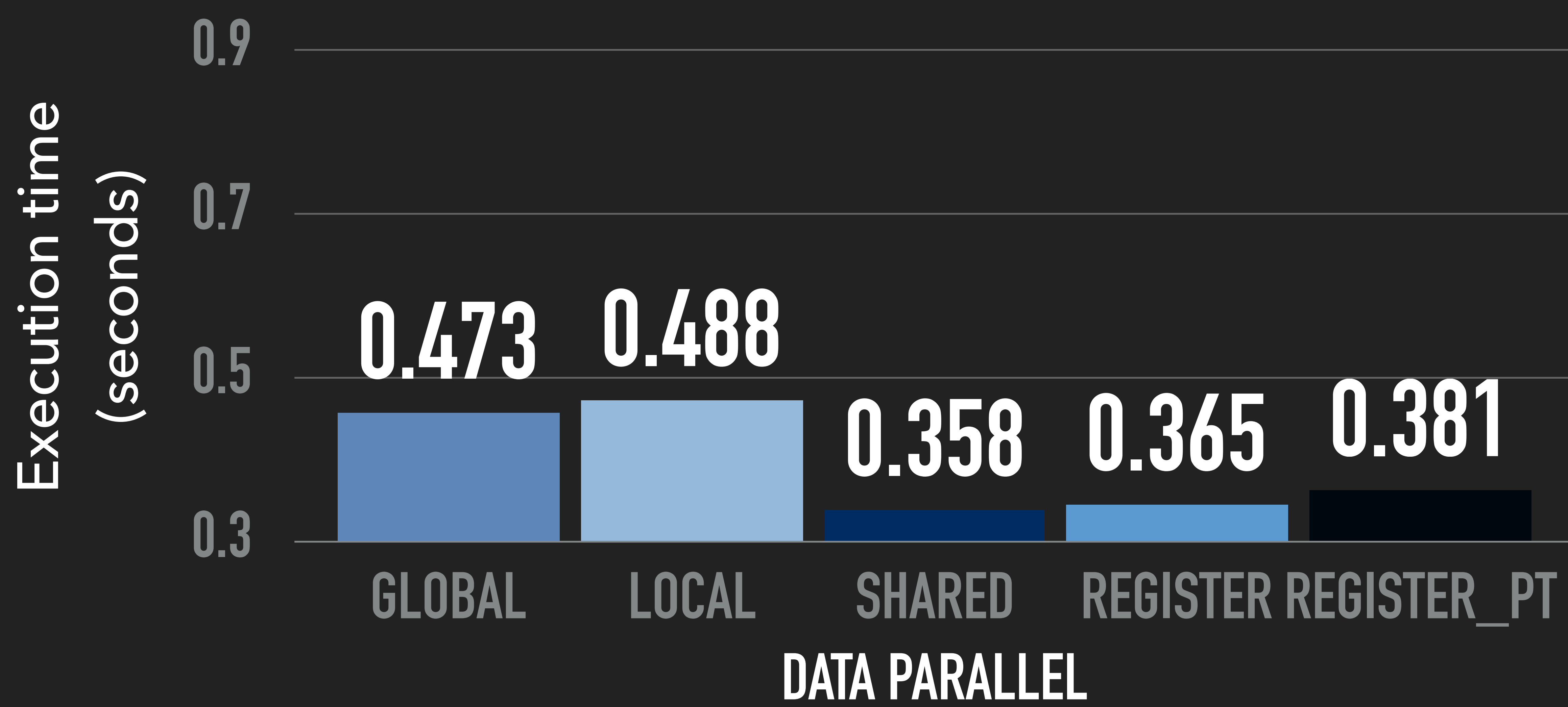
RESULTS TPC-H QUERY 01 – SF 100



Slightly speedup performance for local, shared and register

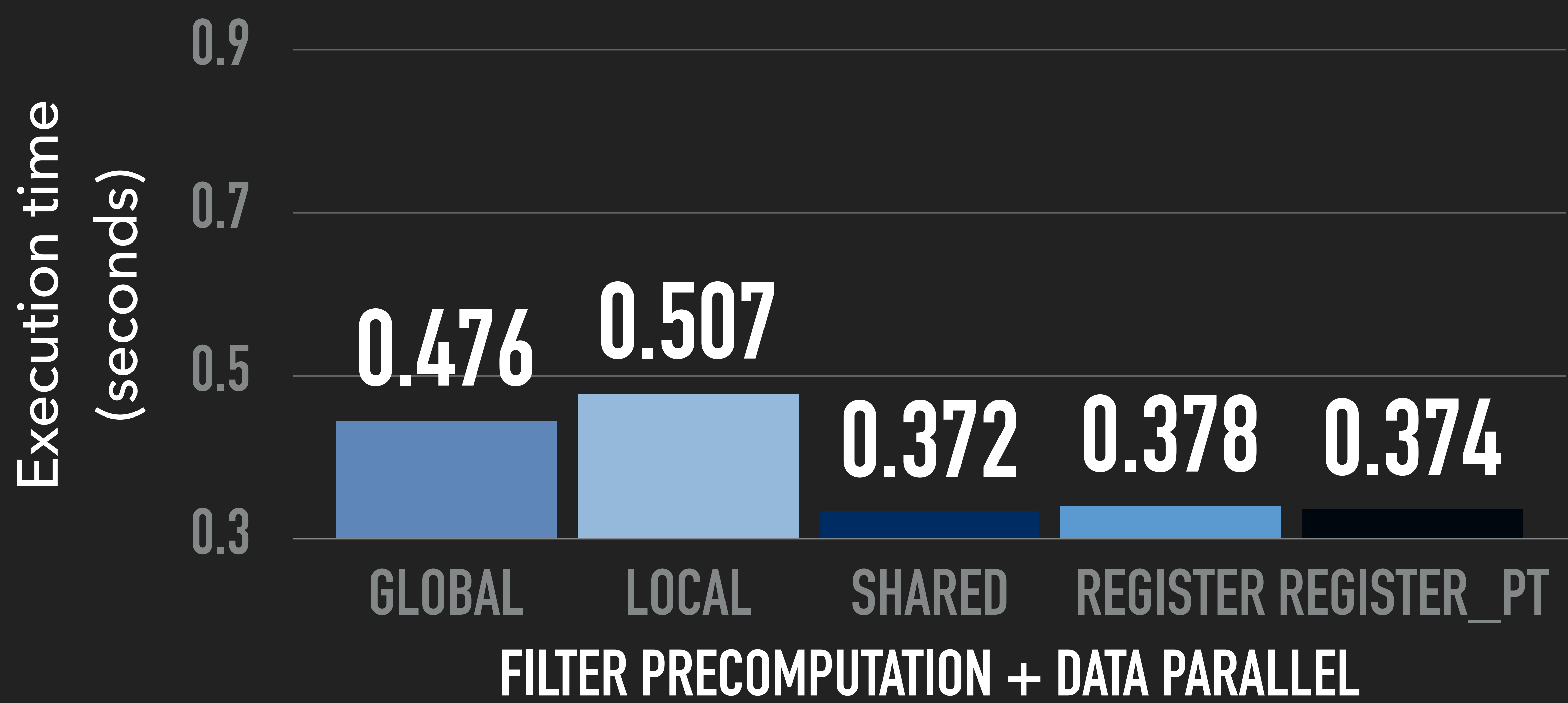
Time to filter on CPU is compensated by bitmap transfer

RESULTS TPC-H QUERY 01 – SF 100



Data parallel yields a 30% speedup over the best GPU-only implementation

RESULTS TPC-H QUERY 01 – SF 100



Slows computation down due to the filter/transfer overhead

RELATED WORK TPC-H QUERY 01

System	CPU/GPU	SF 100 (msec)
Red Fox	GPU	33000
Ocelot	GPU	3470
Voodoo	GPU	2940
AXE/GPU	GPU	2580
CoGaDB	GPU	3500
(This work)	GPU	536
PCI I/O Compressed	GPU	493
Red Fox	CPU	2.7E+06
Voodoo	CPU	1620
Hyper	CPU	1200
AXE/CPU	CPU	8380
(This work)	CPU	791
(This work)	CPU+GPU	385

OPTIMIZING GROUP-BY AND AGGREGATION USING GPU-CPU CO-PROCESSING

CONCLUSION

CONCLUSION

GPU-CPU Co-processing achieved significantly improved performance compared to a CPU-only baseline and GPU-only implementation

Future work include: larger number of groups, non-uniform data distribution, JIT compilation and a co-processing framework

https://github.com/diegomestre2/tpchQ01_GPU