Table transpose for GPU databases

Gabriel Mateescu
Software Engineer, DIPF
mateescu@acm.org

Overview

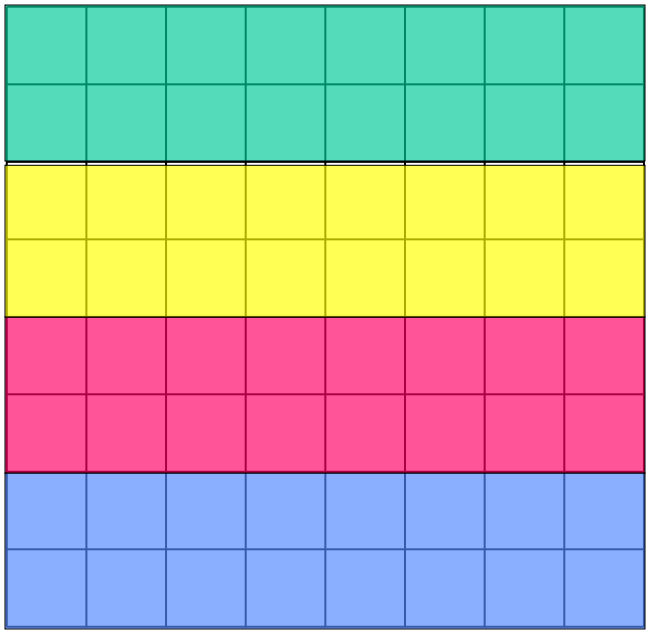
- In-memory column-oriented databases
- Memory architecture of accelerators
- Table transpose on GPU
- Scaling out with multiple accelerators
- Conclusion

In-memory columnar database

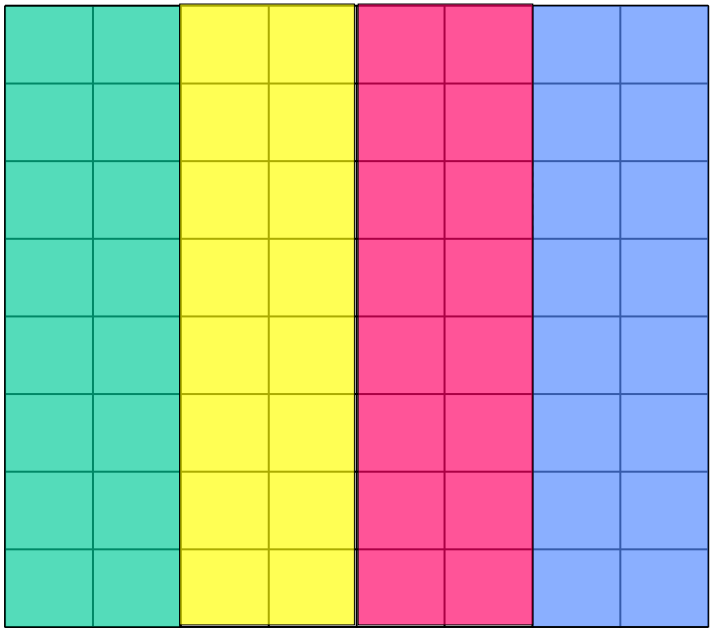
- Scalable *in-memory columnar databases*
- For scalability, table partitioning is needed
  - Horizontal partitioning (sharding)
    - Divide row set into ranges of rows
  - Vertical partitioning
    - Divide the column set: column-oriented database
- Partitioning is applied for all hardware architectures
  - Intel Xeon machines (hundreds of GB per node)
  - GPU machines (tens of GB per node)

Table partitioning

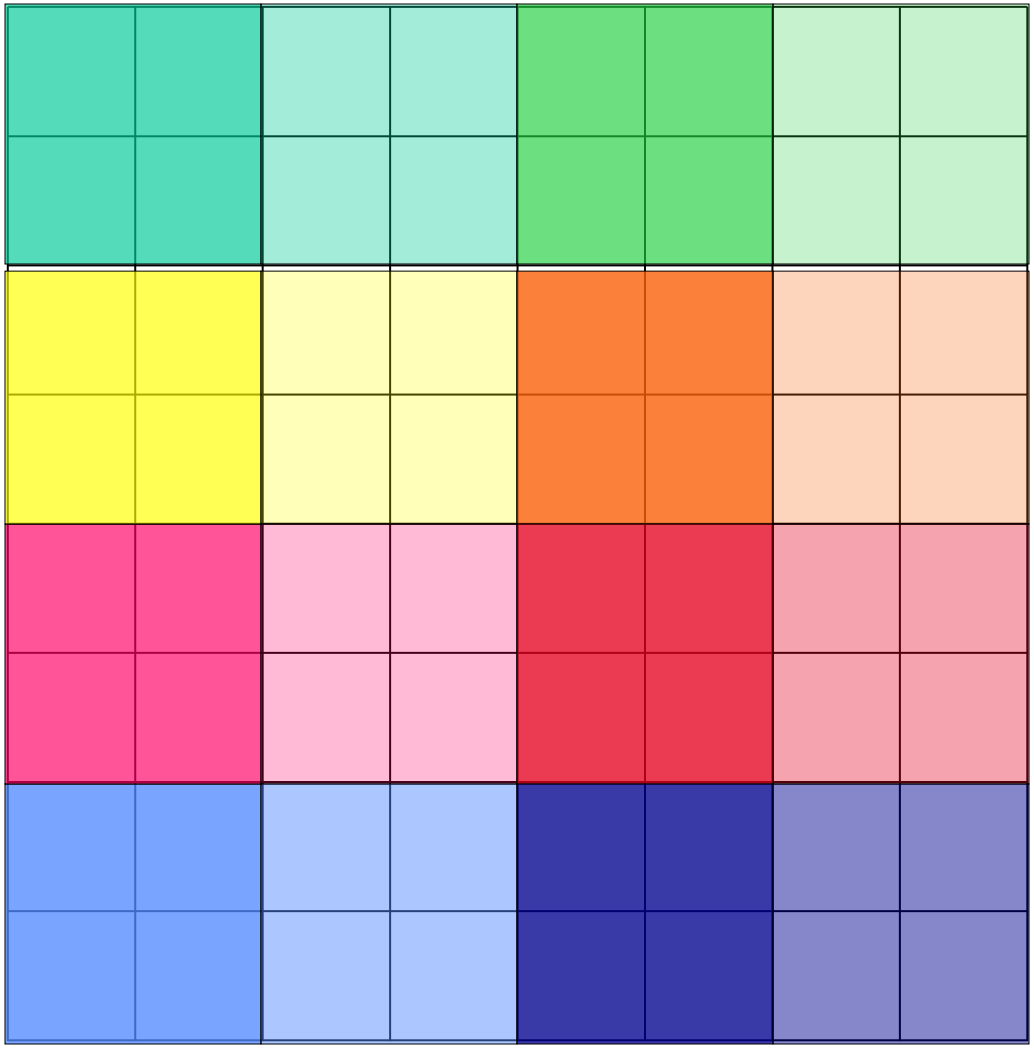
- Sharded table



- Columnar table



Sharded columnar table



In-memory Databases

- Table partitioning and scalable systems provide the basis for in-memory databases
- In-memory system can have very different memory profiles
  - Intel x86 systems: memory optimized for *low latency* using large, multi-level caches
  - GPU accelerators: memory optimized for *high bandwidth*, much higher than that of x86 systems
  - GPU memory (up to 12 GB) is smaller than x86 memory (hundreds of GB)

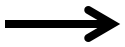
Table Access patterns

- Two of the access patterns that occur:
  - table transpose
  - table scan (possibly along with joins)
- Memory bandwidth limits the performance of both table scan and transpose
  - Machine architecture needs to be understood to reach the peak memory bandwidth
- Efficient table (matrix) transpose on GPU uses almost all memory bandwidth

Table Transpose Example

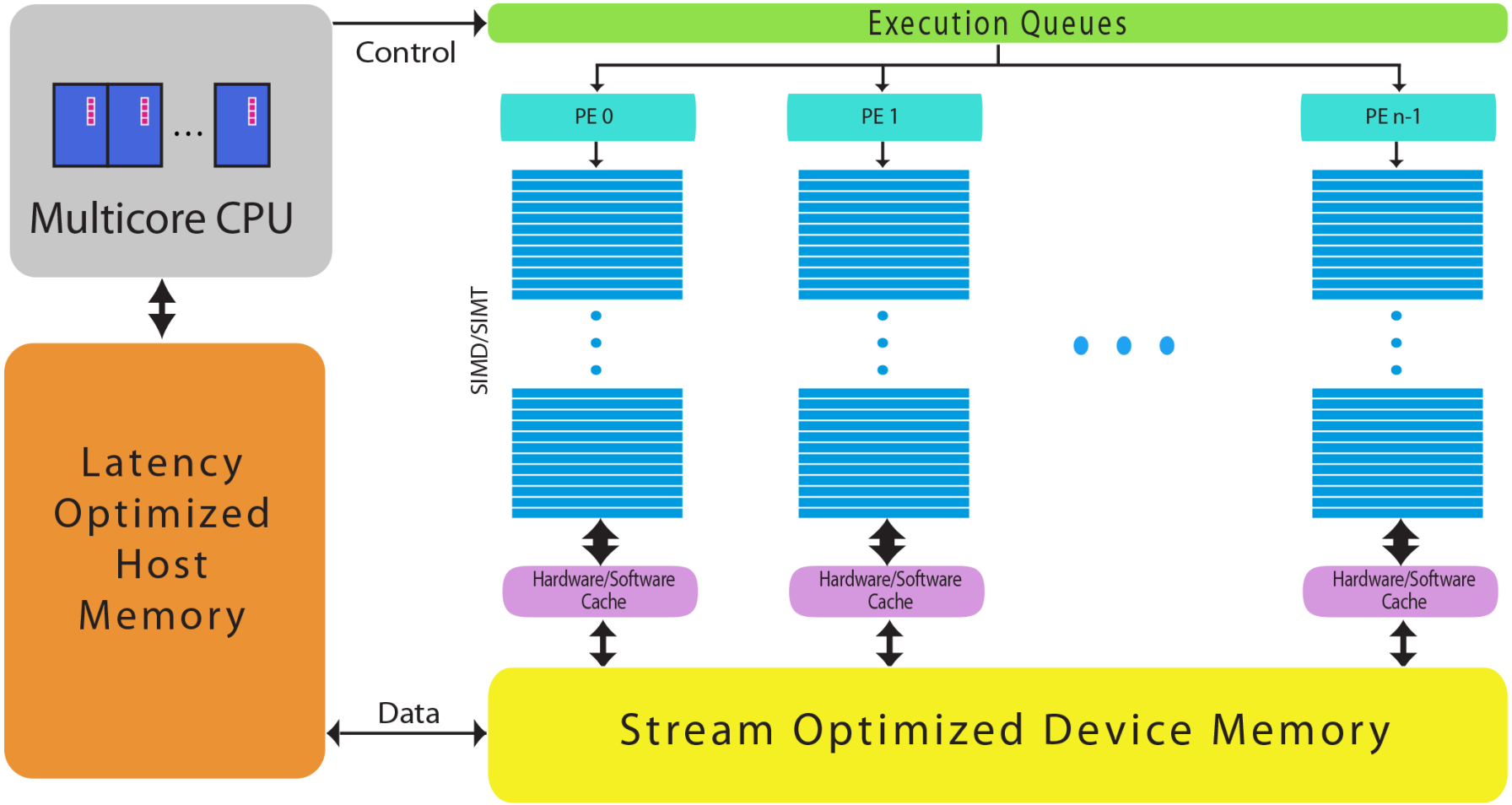
- Example of table view generated using matrix transpose

Time	Ka	Kb	Kc
T1	17.3	18.1	19.2
T2	21.9	20.5	22.1
T3	20.4	23.7	24.8



Kind	T1	T2	T3
Ka	17.3	21.9	20.4
Kb	18.1	20.5	23.7
Kc	19.2	22.1	24.8

Accelerator Architecture Model

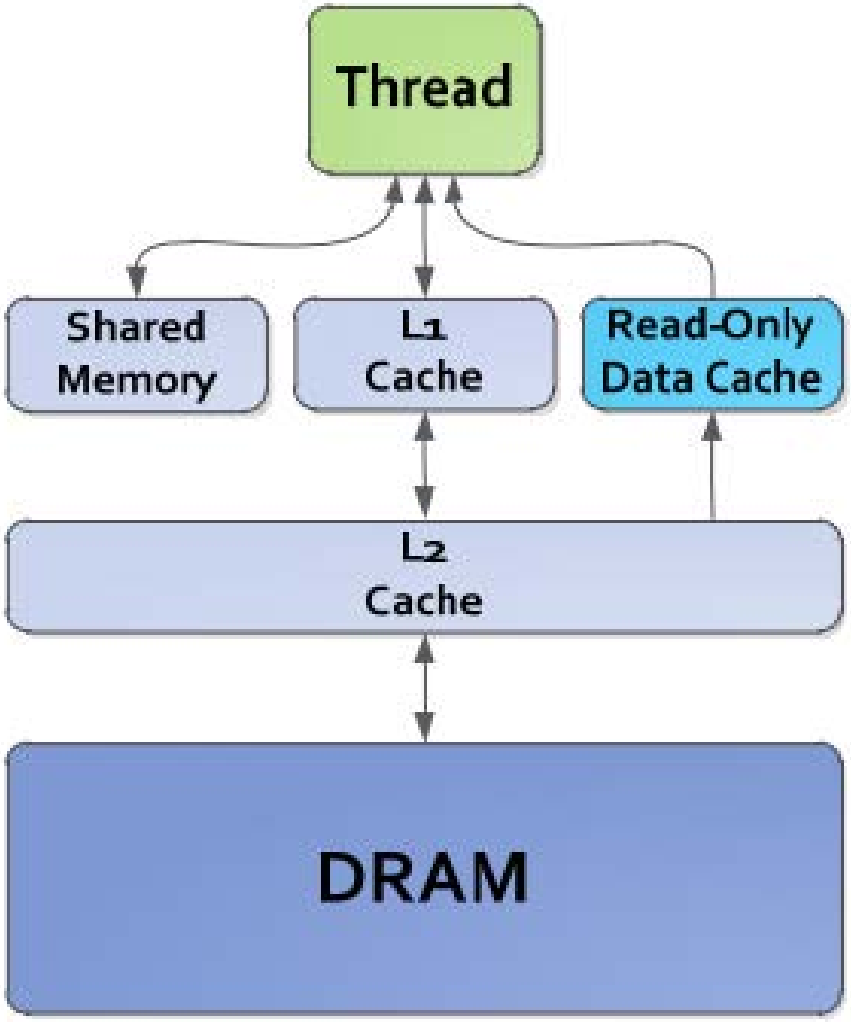


- Model by Michael Wolfe, NVIDIA PGI Group

GPU Accelerator Memory

- Device Memory aka Global Memory or DRAM
  - Data input/output for kernels
  - Data transfer between host and device
  - Optimized for bandwidth
    - Peak bandwidth reached for *stride-one accesses*
    - The GPU coalesces accesses issued by all threads in a warp (SIMD) into the minimum number of transactions
- Shared memory (user managed cache)
  - Shared by all threads in a thread block
  - Low latency (100x) and high bandwidth (10x)
  - *no penalty for strided access* to shared memory

Kepler GPU Memory Hierarchy



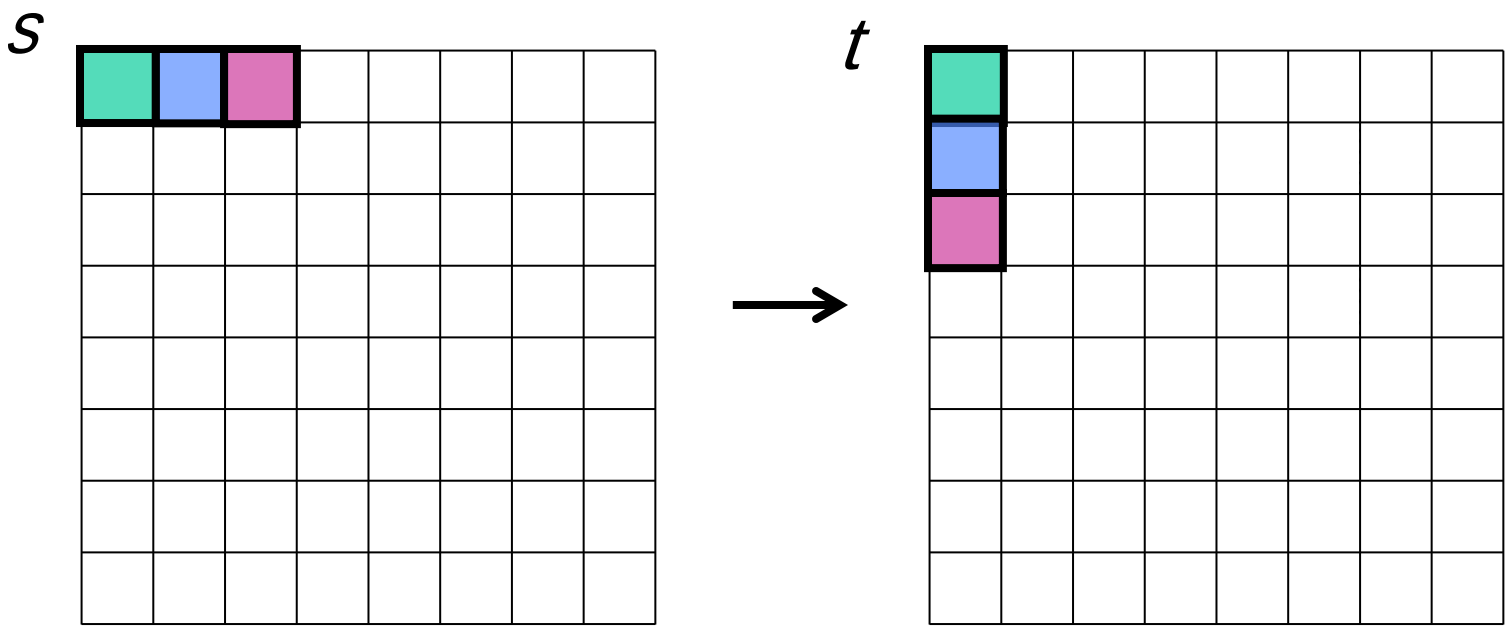
Tesla K20 and K40 feeds and speeds

Features	Tesla K40	Tesla K20X	Tesla K20
Number and Type of GPU	1 Kepler GK110B	1 Kepler GK110	
Peak double precision floating point performance	1.43 Tflops	1.31 Tflops	1.17 Tflops
Peak single precision floating point performance	4.29 Tflops	3.95 Tflops	3.52 Tflops
Memory bandwidth (ECC off)	288 GB/sec	250 GB/sec	208 GB/sec
Memory size (GDDR5)	12 GB	6 GB	5 GB
CUDA cores	2880	2688	2496

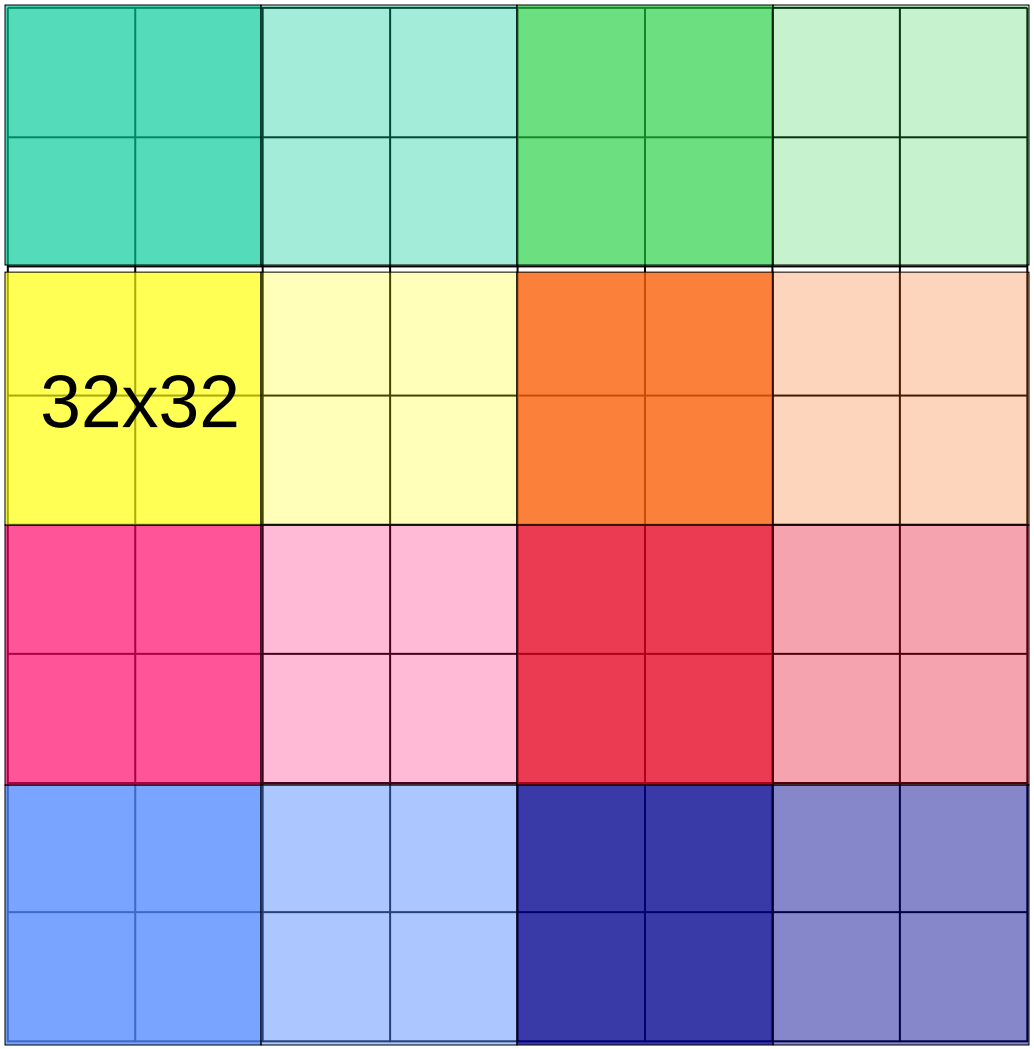
- ECC traffic consumes up to 15% of the memory bandwidth
- K20 memory bandwidth with ECC on is 177 GB/sec

Table Transpose on GPU

- Matrix *s* of SP-floats is transposed into matrix *t*
- Divide matrix *s* into tiles of size *32x32*
- Assign each tile to a *32x8* thread block
- Challenge: *avoid non-coalesced memory accesses*



Parallel matrix transpose



- The matrix of order *N* is divided into tiles of size *32x32* elements
- A tile is assigned to a thread block of *32x8* threads
- A *N/32xN/8* grid of thread blocks transposes the matrix

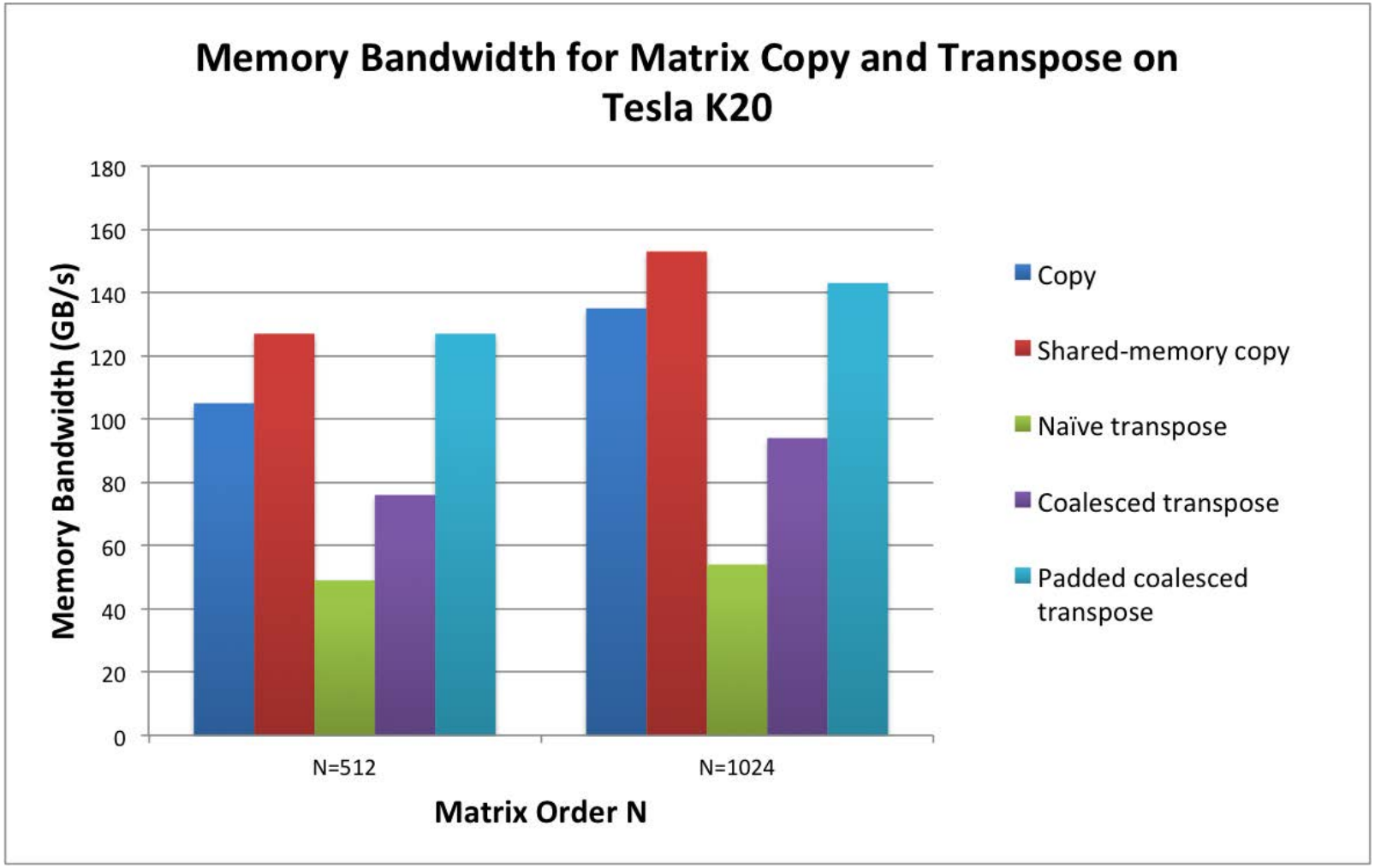
Using shared memory for write coalescing

- The threads in a *warp* read 32 elements of *s* with *stride one* and write in a *shared memory* buffer with *stride 32*
  - a 32x32 tile (4KB) is written by a thread block to the shared memory buffer
  - no penalty for strided access to shared memory
- After a full tile is written to the shared memory buffer, data is *read* from the buffer and *written* to global memory with *stride one*, building a 32x32 tile of the matrix *t*

Bank conflicts in shared memory

- Shared memory: 32 banks, 4-byte/bank
  - Accessed are issued per warp (32 threads)
  - Bank conflict if two or more threads in a warp access different words from the same bank
  - Conflicting accesses are serialized
- Avoid bank conflicts by using a shared memory buffer tile[32][33] instead of tile[32][32]
  - Array padding to avoid bank conflicts

Bandwidth for Matrix Transpose and Copy



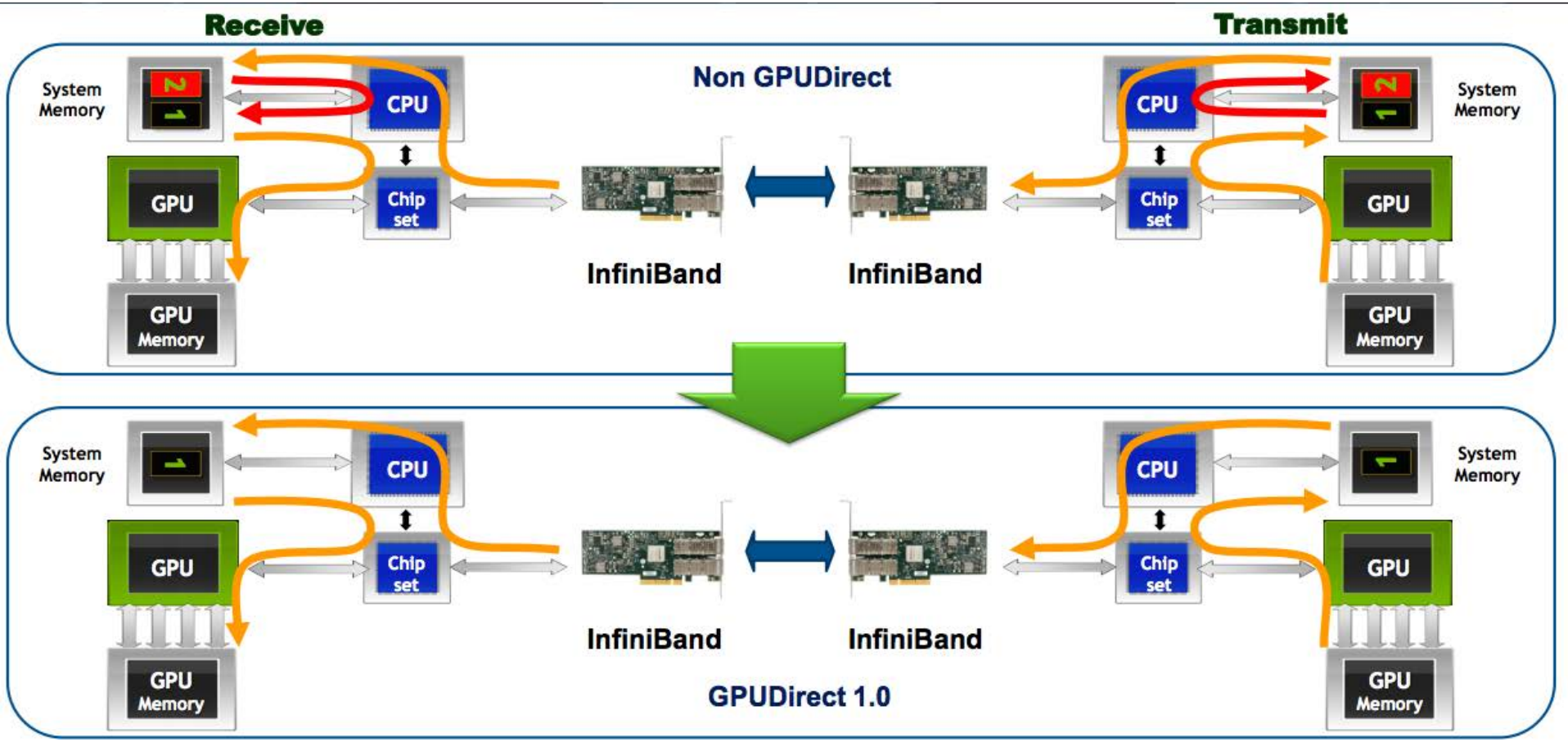
Transpose approach summary

- Efficient matrix transpose on GPU
  - Divide the matrix into tiles and assign one tile per thread block
  - The source matrix reads have stride-one and are coalesced
  - Use shared memory to get stride-one writes of the target matrix and get coalesced wirtes
  - Use array padding to avoid bank conflicts for the shared memory

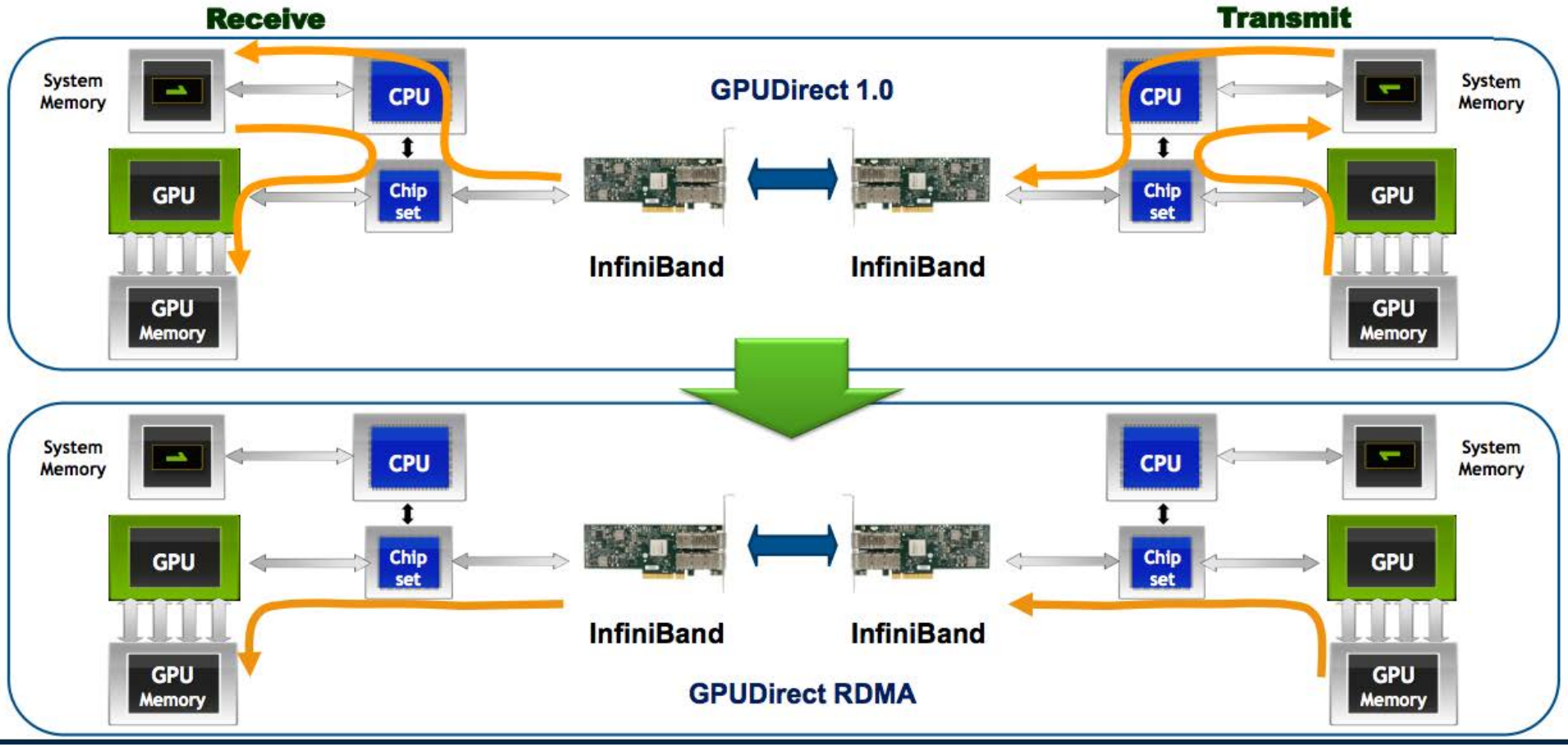
Scaling: multiple accelerators

- Multiple accelerators per server
  - E.g., 4 or 8 Tesla K20 GPU cards
- Multiple servers connected via InfiniBand
- GPUDirect with RDMA for inter-accelerator communication eliminates memory copy operations

GPUDirect


Receive
Transmit
Non GPUDirect
System Memory
CPU
Chip set
GPU
GPU Memory
InfiniBand
InfiniBand
System Memory
CPU
Chip set
GPU
GPU Memory
1
2
System Memory
CPU
Chip set
GPU
GPU Memory
InfiniBand
InfiniBand
System Memory
CPU
Chip set
GPU
GPU Memory
1
GPUDirect 1.0

GPUDirect with RDMA



PCI express bottleneck

- GPU device memory bandwidth: 200 GB/sec
- CPU socket memory bandwidth: 40 GB/sec
- PCI express bandwidth:

PCI-e version	Per-lane bandwidth	8x bandwidth	16x bandwidth
2.x	4 Gb/sec	4 GB/sec	8 GB/sec
3.x	8 Gb/sec	8 GB/sec	16 GB/sec

HPC vs Big Data Applications

- For both HPC and big data applications, efficient data manegement is essential for high performance
- HPC and Big data can have different requirements on data
  - the amount of input to be ingested
  - the amount of scratch data produced
  - the aount of output data produced
  - the at-rest or in-motion nature of input data

Conclusion

- Data has gravity

- Architect systems around data and reduce data movement to save time and energy
- When data movement is needed, organize it to fully exploit the available bandwidth