



058165 - PARALLEL COMPUTING

Fabrizio Ferrandi

a.a. 2022-2023

Memory access performance

- Material from The GPU Teaching Kit .
- The GPU Teaching Kit is licensed by NVIDIA and the University of Illinois under the [Creative Commons Attribution-NonCommercial 4.0 International License.](#)

Objective

- To learn that memory bandwidth is a first-order performance factor in a massively parallel processor
 - DRAM bursts, banks, and channels
 - All concepts are also applicable to modern multicore processors

Global Memory (DRAM) Bandwidth

— Ideal

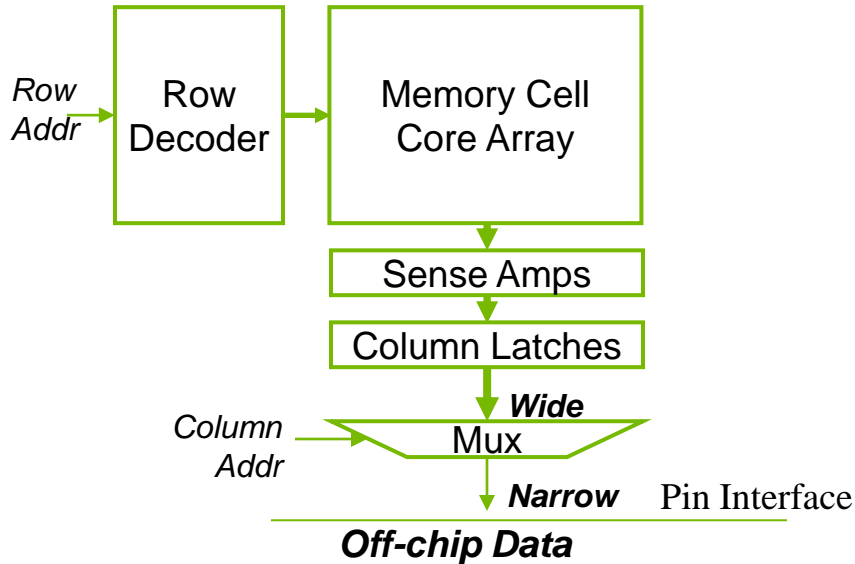


— Reality

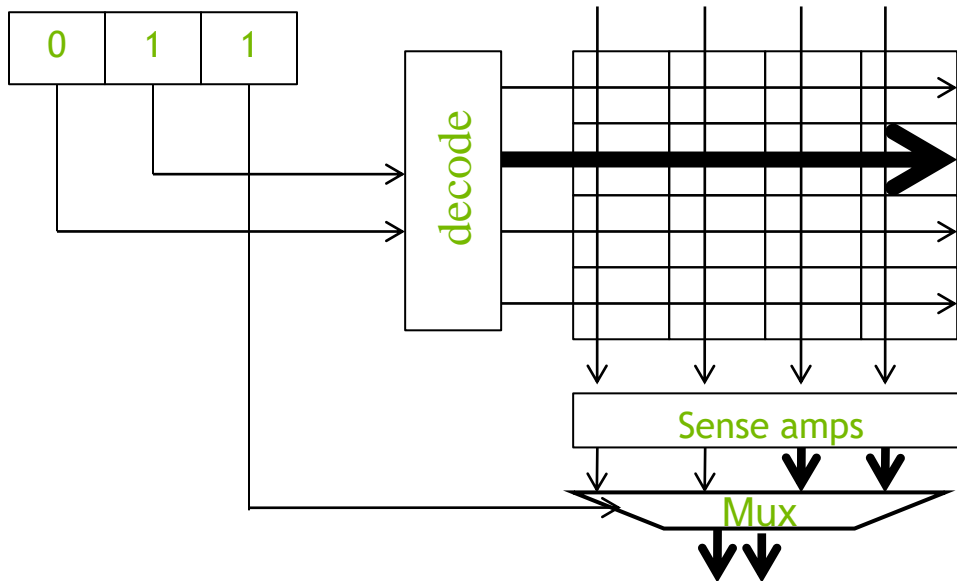


DRAM Core Array Organization

- Each DRAM core array has about 16M bits
- Each bit is stored in a tiny capacitor made of one transistor

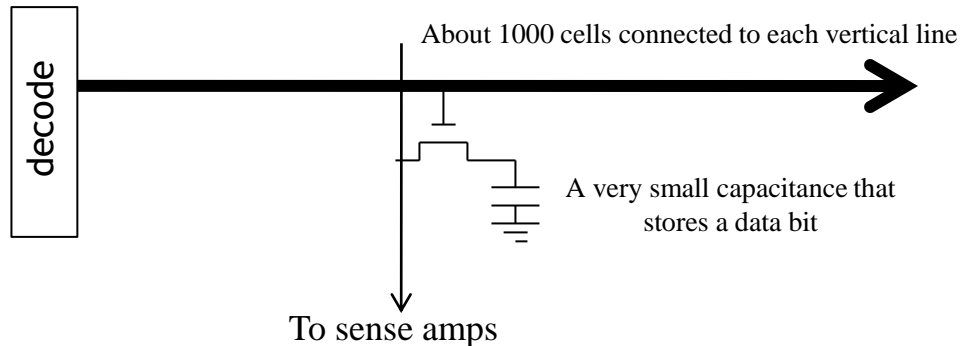


A very small (8x2-bit) DRAM Core Array



DRAM Core Arrays are Slow

- Reading from a cell in the core array is a very slow process
 - DDR: Core speed = $\frac{1}{2}$ interface speed
 - DDR2/GDDR3: Core speed = $\frac{1}{4}$ interface speed
 - DDR3/GDDR4: Core speed = $\frac{1}{8}$ interface speed
 - ... likely to be worse in the future

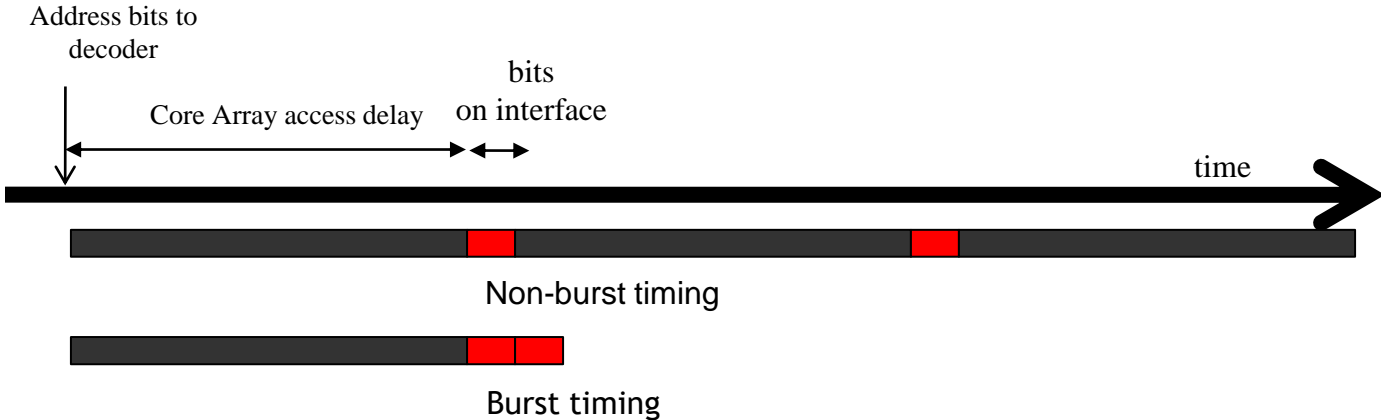


DRAM Bursting

- For DDR{2,3} SDRAM cores clocked at $1/N$ speed of the interface:
 - Load ($N \times$ interface width) of DRAM bits from the same row at once to an internal buffer, then transfer in N steps at interface speed
 - DDR3/GDDR4: buffer width = $8X$ interface width

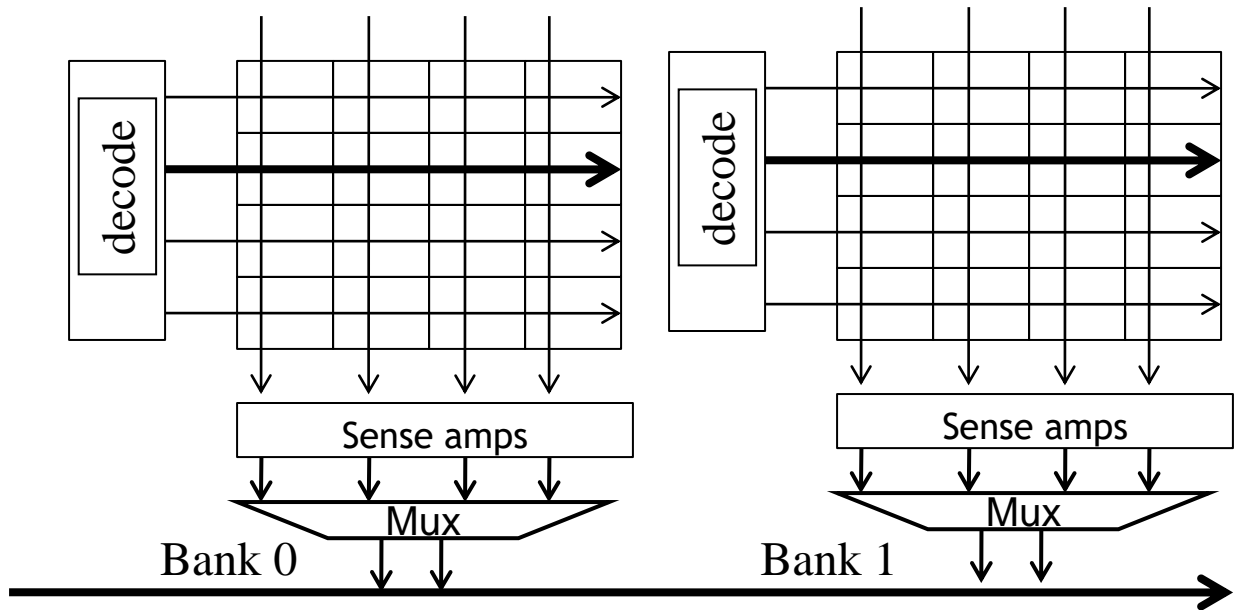


DRAM Bursting Timing Example



Modern DRAM systems are designed to always be accessed in burst mode. Burst bytes are transferred to the processor but discarded when accesses are not to sequential locations.

Multiple DRAM Banks



DRAM Bursting with Banking



Single-Bank burst timing, dead time on interface



Multi-Bank burst timing, reduced dead time

GPU off-chip memory subsystem

- NVIDIA RTX6000 GPU:
 - Peak global memory bandwidth = 672GB/s
- Global memory (GDDR6) interface @ 7GHz
 - 14 Gbps pin speed
 - For GDDR6 32-bit interface, we can sustain only about 56 GB/s
 - We need a lot more bandwidth (672 GB/s) – thus 12 memory channels

Objective

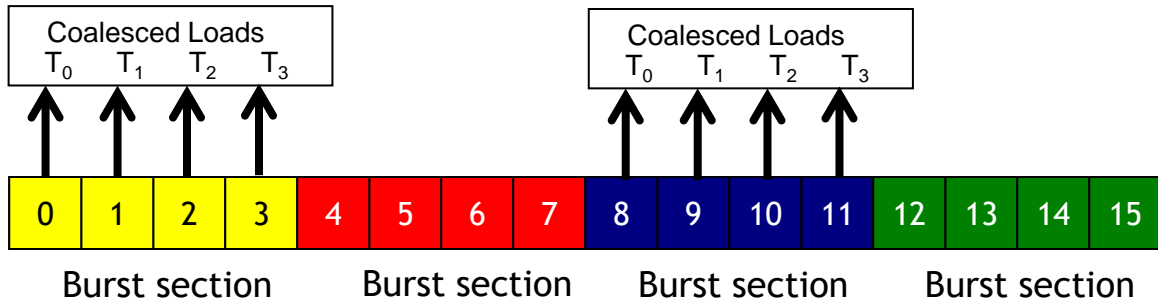
- To learn that memory coalescing is important for effectively utilizing memory bandwidth in CUDA
 - Its origin in DRAM burst
 - Checking if a CUDA memory access is coalesced
 - Techniques for improving memory coalescing in CUDA code

DRAM Burst – A System View



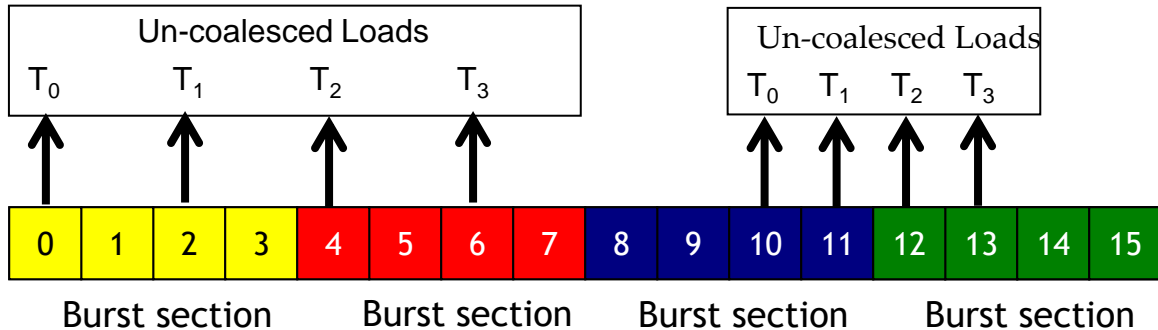
- Each address space is partitioned into burst sections
 - Whenever a location is accessed, all other locations in the same section are also delivered to the processor
- Basic example: a 16-byte address space, 4-byte burst sections
 - In practice, we have at least 4GB address space, burst section sizes of 128-bytes or more

Memory Coalescing



- When all threads of a warp execute a load instruction, if all accessed locations fall into the same burst section, only one DRAM request will be made and the access is fully coalesced.

Un-coalesced Accesses

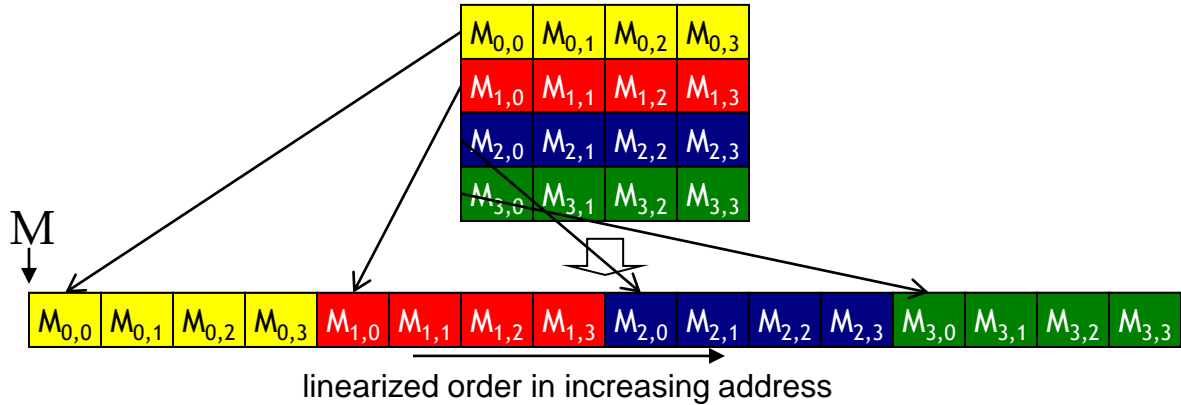


- When the accessed locations spread across burst section boundaries:
 - Coalescing fails
 - Multiple DRAM requests are made
 - The access is not fully coalesced.
- Some of the bytes accessed and transferred are not used by the threads

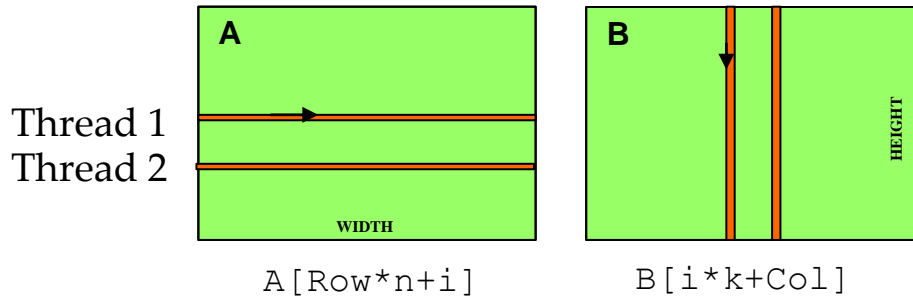
How to judge if an access is coalesced?

- Accesses in a warp are to consecutive locations if the index in an array access is in the form of
 - $A[(\text{expression with terms independent of threadIdx.x}) + \text{threadIdx.x}]$;

A 2D C Array in Linear Memory Space



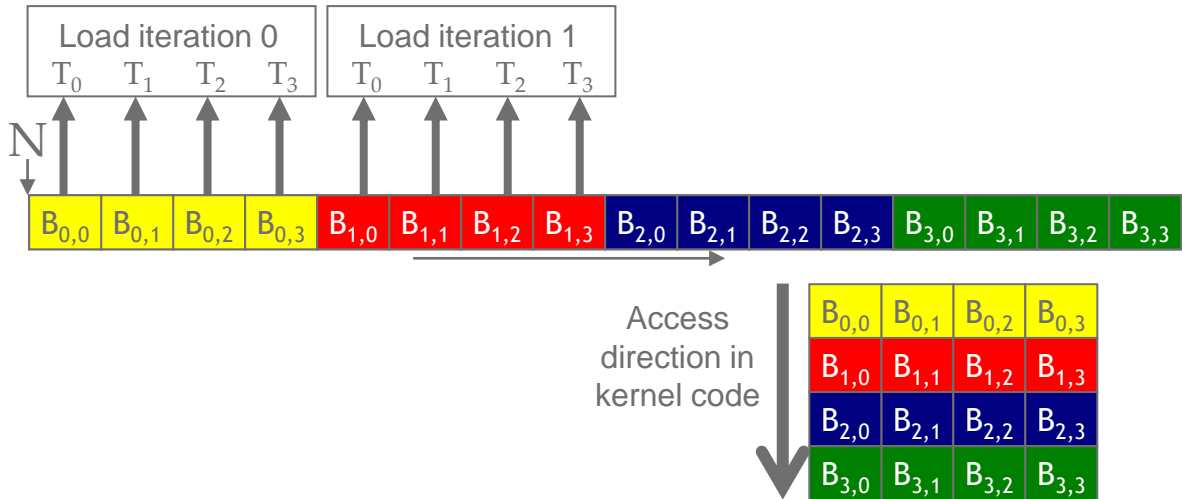
Two Access Patterns of Basic Matrix Multiplication



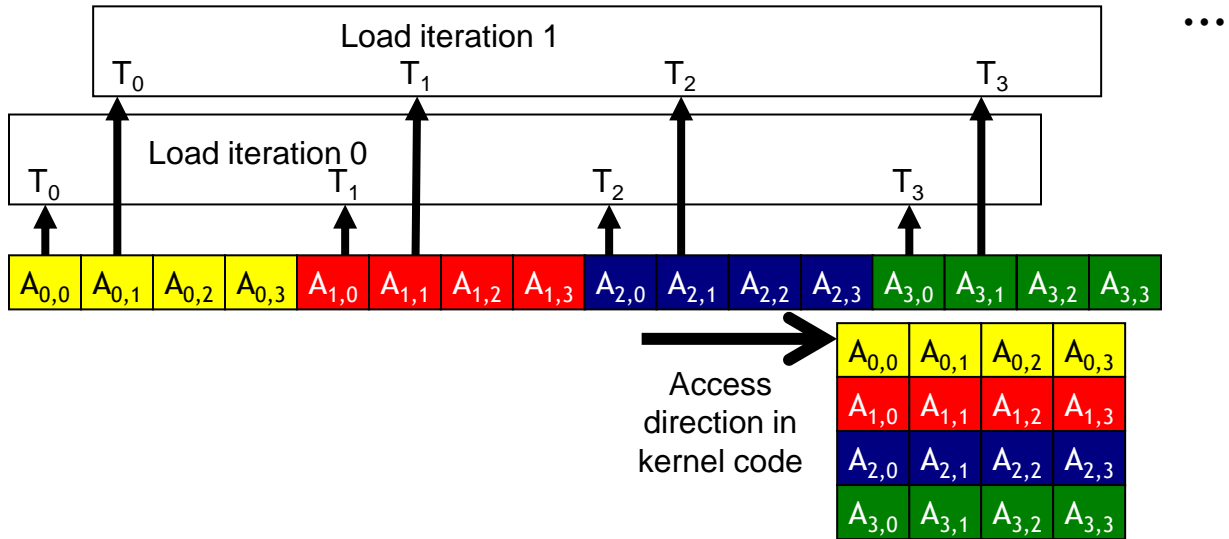
i is the loop counter in the inner product loop of the kernel code

A is $m \times n$, B is $n \times k$
 $\text{Col} = \text{blockIdx.x} \times \text{blockDim.x} + \text{threadIdx.x}$

B accesses are coalesced



A Accesses are Not Coalesced



Loading an Input Tile

Have each thread load an A element
and a B element at the same relative
position as its C element.

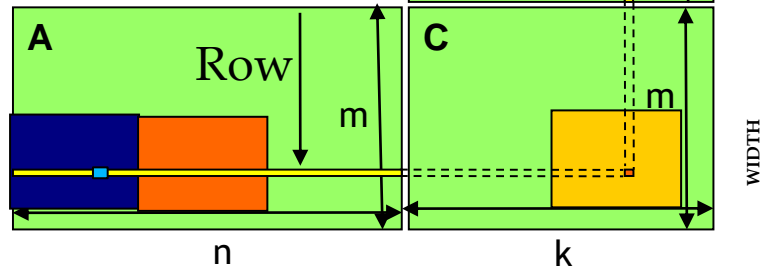
```
int tx = threadIdx.x
```

```
int ty = threadIdx.y
```

Accessing tile 0 2D indexing:

```
A[Row][tx]
```

```
B[ty][Col]
```



Corner Turning

