NVIDIA.

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Memory Access Performance

DRAM Bandwidth
Memory Coalescing in CUDA

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
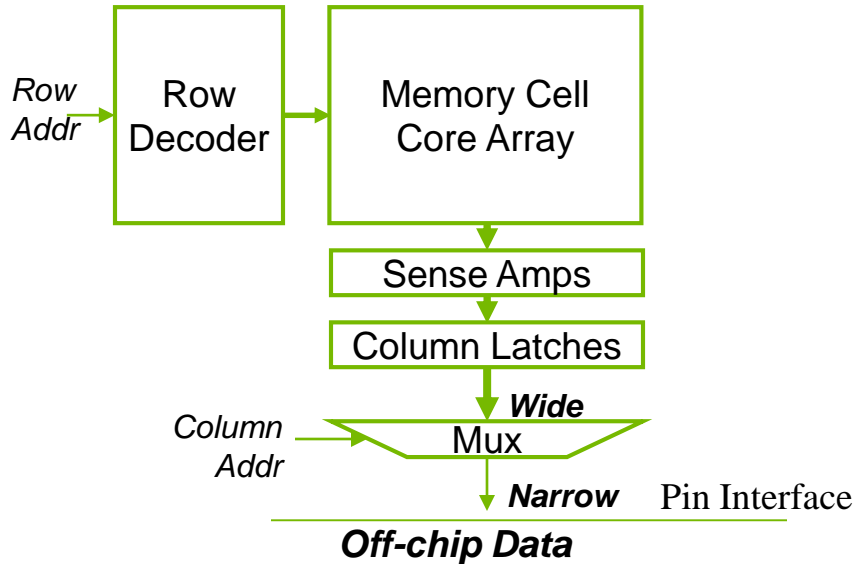
Global Memory (DRAM) Bandwidth
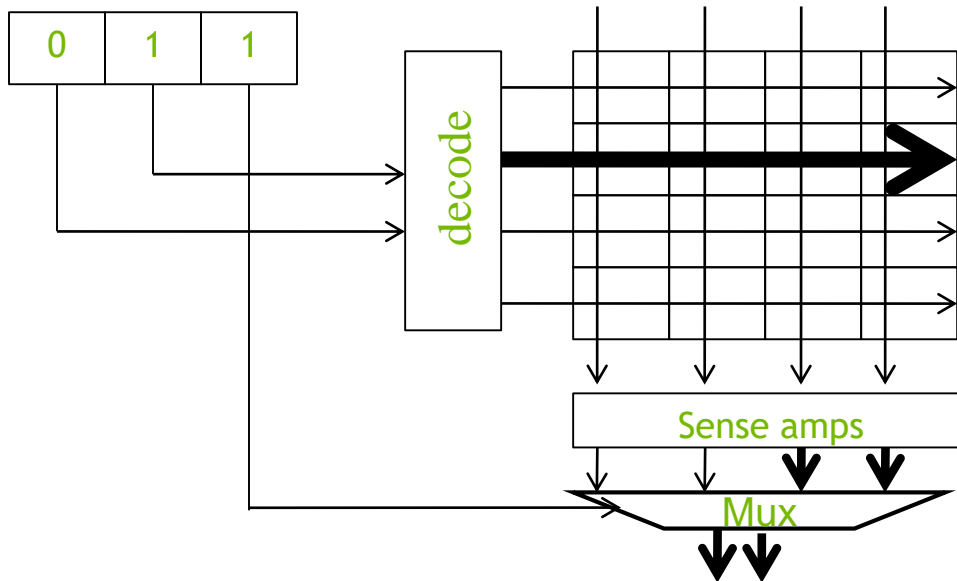
- Ideal

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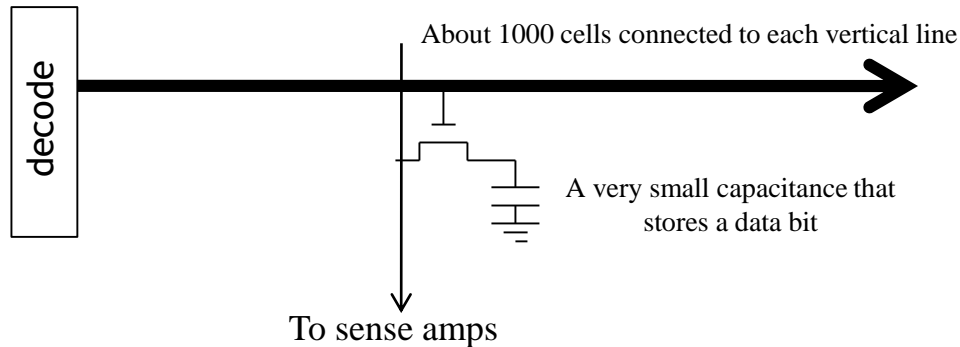
A very small (8x2-bit) DRAM Core Array

DRAM Core Arrays are Slow

– Reading from a cell in the core array is a very slow process
  – DDR: Core speed = ½ interface speed
  – DDR2/GDDR3: Core speed = ¼ interface speed
  – DDR3/GDDR4: Core speed = ⅛ interface speed
  – … likely to be worse in the future

About 1000 cells connected to each vertical line

decode
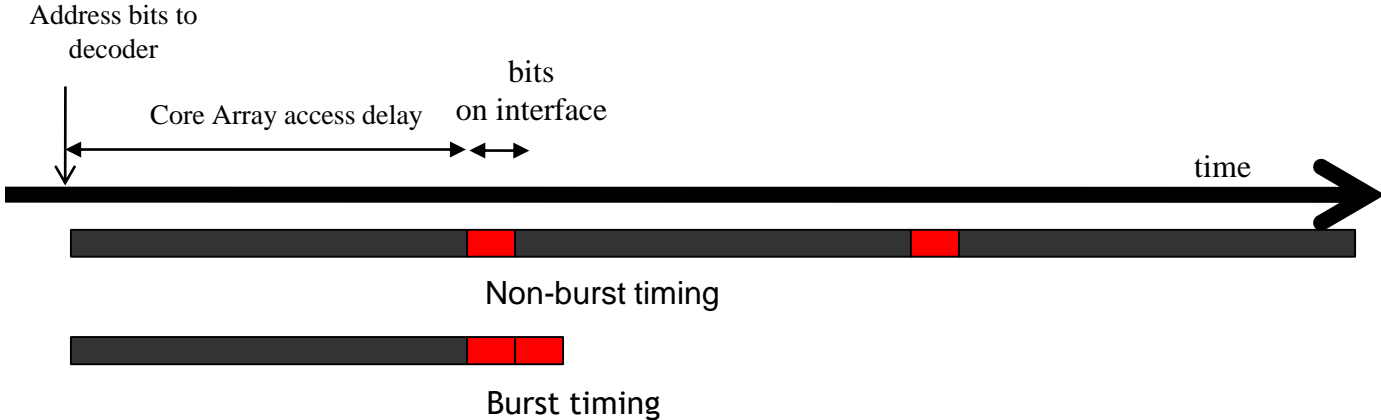
A very small capacitance that stores a data bit

To sense amps

DRAM Bursting

– For DDR{2,3} SDRAM cores clocked at 1/N speed of the interface:
  – Load (N × 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

Address bits to
decoder

Core Array access delay

bits
on interface

time

Non-burst timing

Burst timing

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
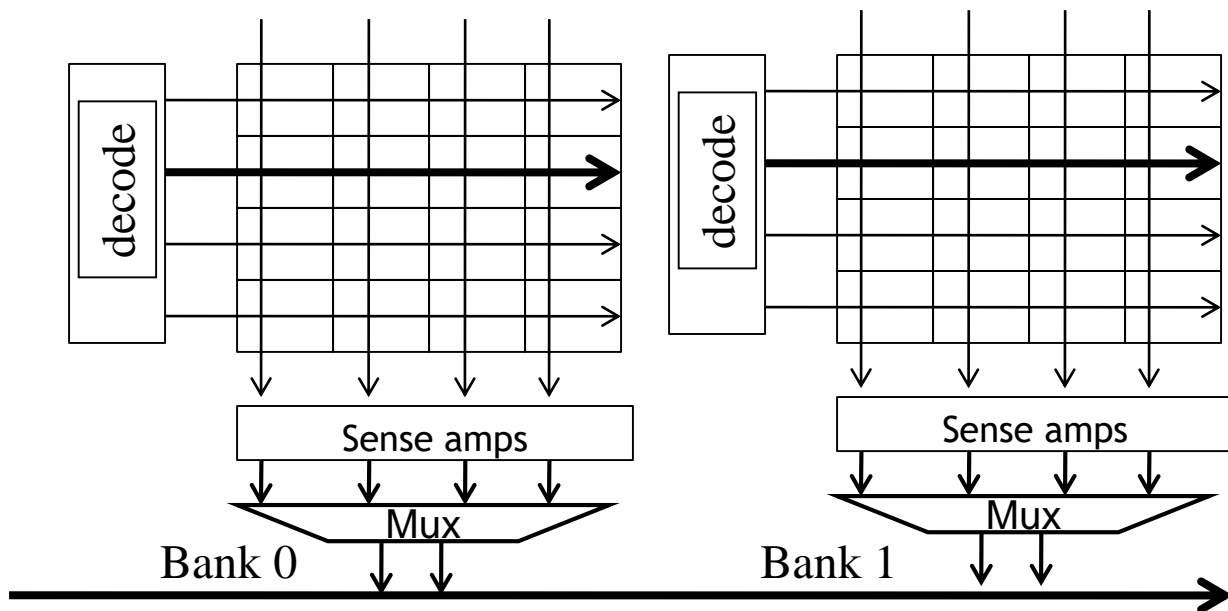
DRAM Bursting with Banking

Single-Bank burst timing, dead time on interface

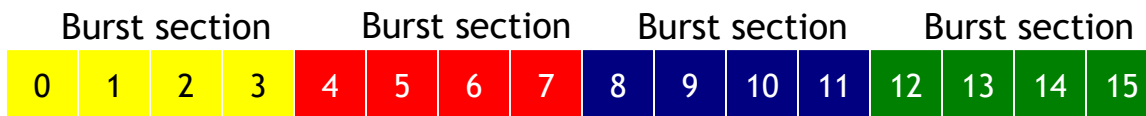Multi-Bank burst timing, reduced dead time

GPU off-chip memory subsystem

– NVIDIA GTX280 GPU:

    – Peak global memory bandwidth = 141.7GB/s

– Global memory (GDDR3) interface @ 1.1GHz

    – (Core speed @ 276Mhz)

    – For a typical 64-bit interface, we can sustain only about 17.6 GB/s (Recall DDR - 2 transfers per clock)

   – We need a lot more bandwidth (141.7 GB/s) – thus 8 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

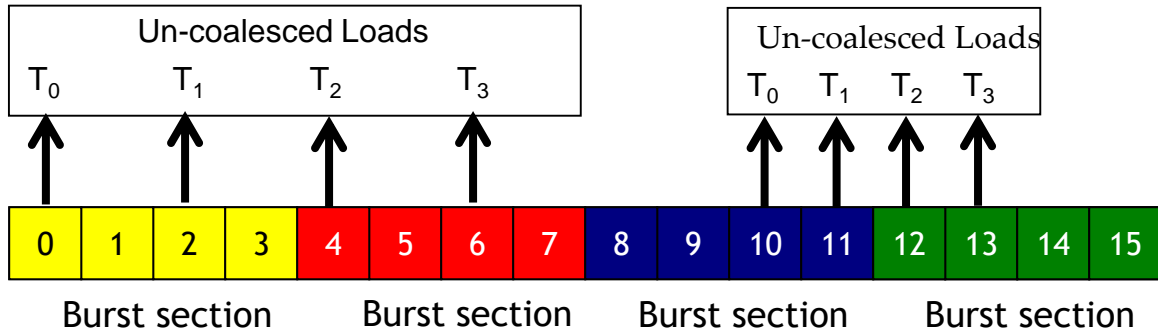| Burst section				Burst section				Burst section				Burst section			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

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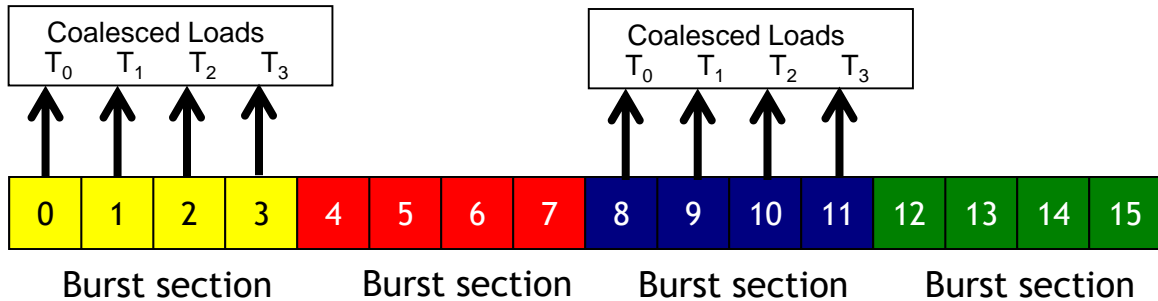
– When all threads of a warp execute a load instruction, if all accessed locations fall into the same burst section, only one DRAM request will be made and the access is fully coalesced.
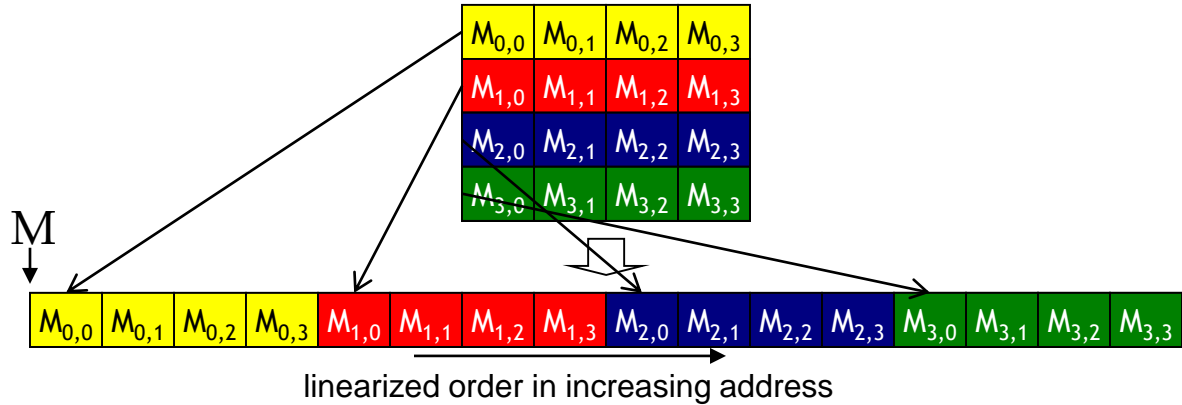
Un-coalesced Accesses

– 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
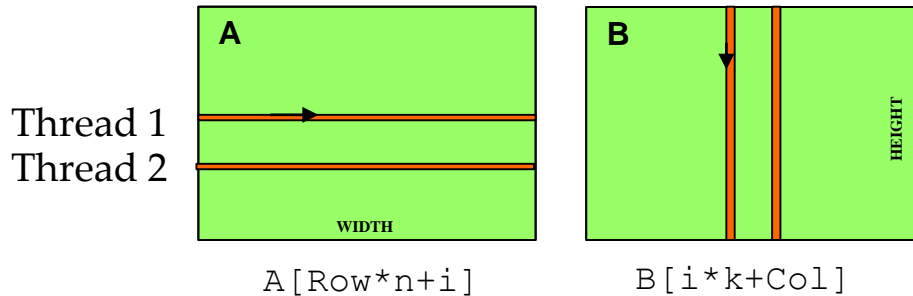
How to judge if an access is coalesced?

– Accesses in a warp are to consecutive locations if the index in an array access is in the form of

    – A[(expression with terms independent of threadIdx.x) + threadIdx.x];
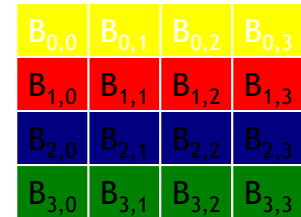
A 2D C Array in Linear Memory Space

linearized order in increasing address

Two Access Patterns of Basic Matrix Multiplication

A[Row*n+i]   B[i*k+Col]

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

A is m × n, B is n × k
Col = blockIdx.x*blockDim.x + threadIdx.x

B accesses are coalesced

$B_{0,0}$	$B_{0,1}$	$B_{0,2}$	$B_{0,3}$	$B_{1,0}$	$B_{1,1}$	$B_{1,2}$	$B_{1,3}$	$B_{2,0}$	$B_{2,1}$	$B_{2,2}$	$B_{2,3}$	$B_{3,0}$	$B_{3,1}$	$B_{3,2}$	$B_{3,3}$

$B_{0,0}$	$B_{0,1}$	$B_{0,2}$	$B_{0,3}$
$B_{1,0}$	$B_{1,1}$	$B_{1,2}$	$B_{1,3}$
$B_{2,0}$	$B_{2,1}$	$B_{2,2}$	$B_{2,3}$
$B_{3,0}$	$B_{3,1}$	$B_{3,2}$	$B_{3,3}$

A Accesses are Not Coalesced

Access direction in kernel code
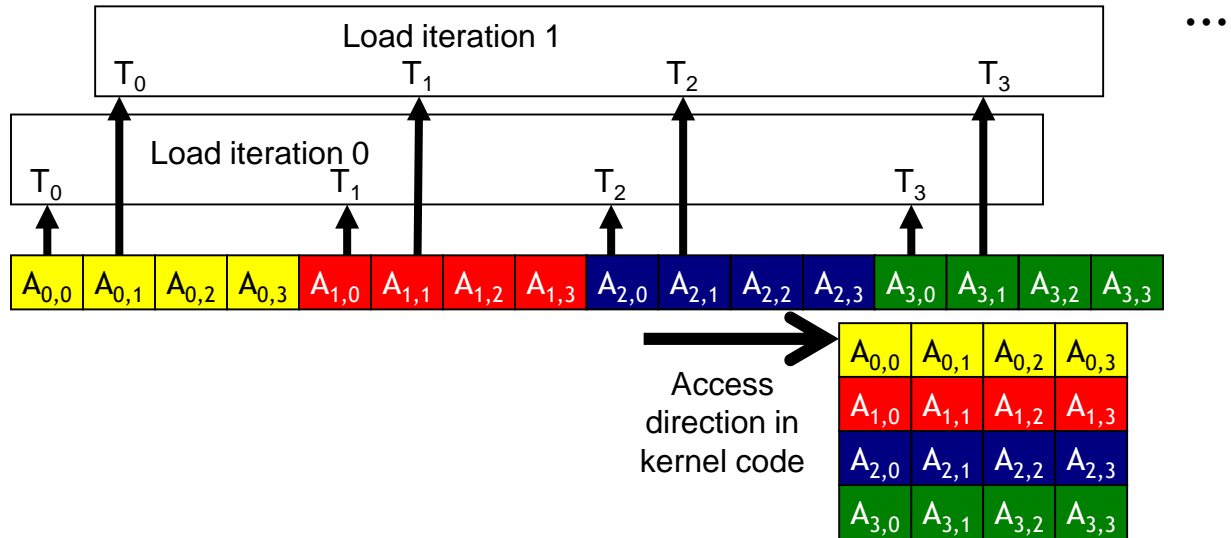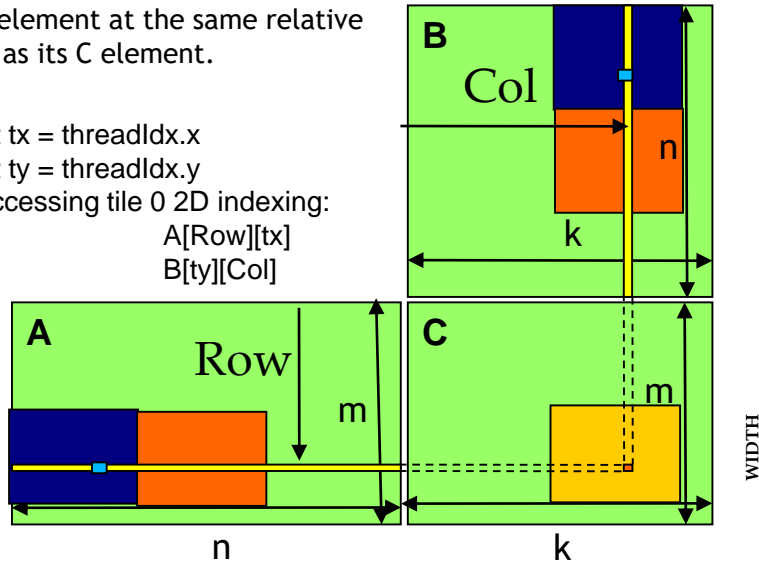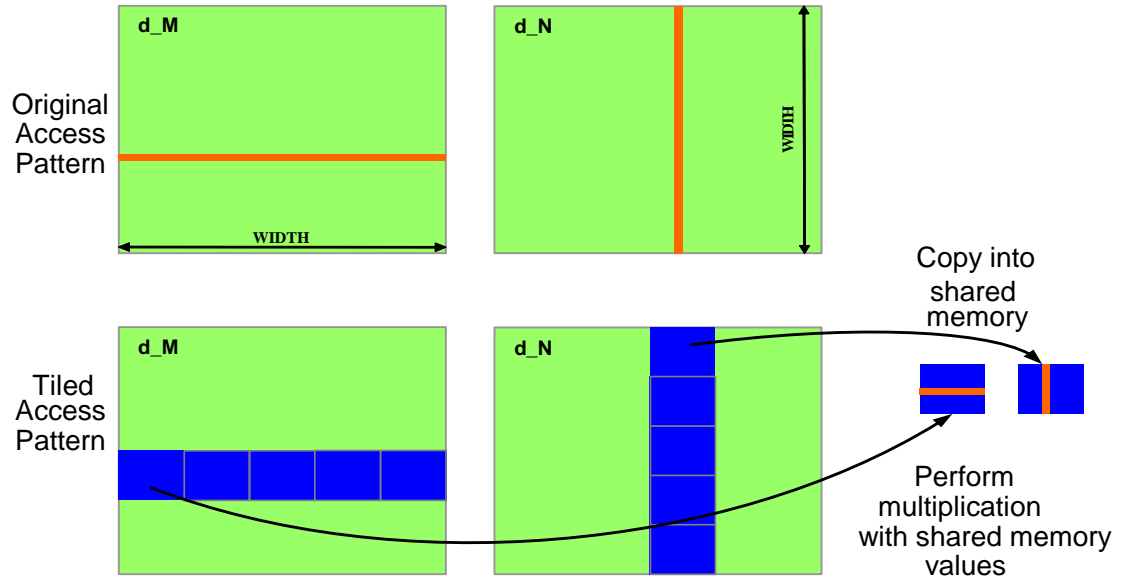
Loading an Input Tile

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]
```

B

Col

n

k

A

Row

m

n

C

m

k

WIDTH

Corner Turning

GPU Teaching Kit

Accelerated Computing

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA