

TELECOM
SudParis



Institut
Mines-Télécom

IA307 - GPU for Deep Learning

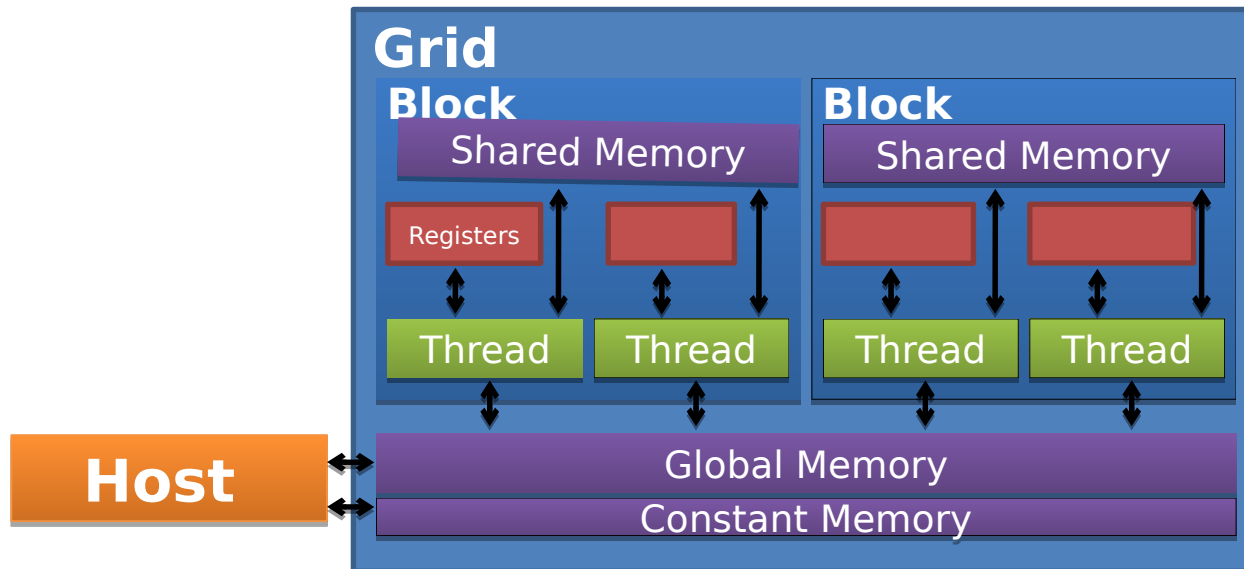
GPU Memory Architecture



Elisabeth Brunet

GPU Memory Architecture

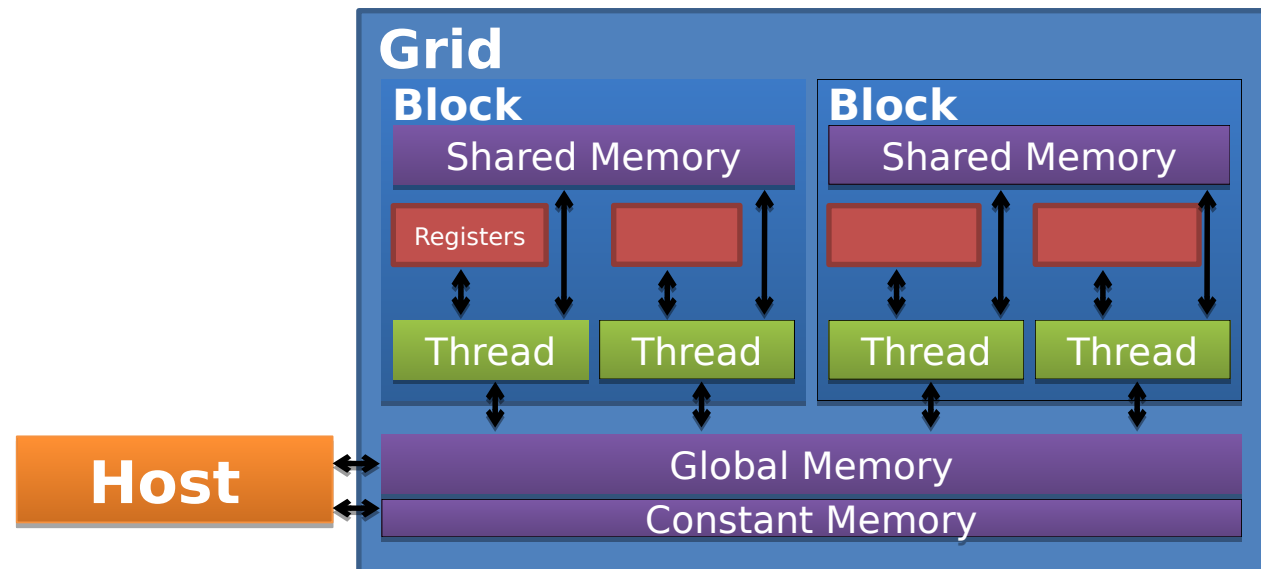
CPU and GPU memory spaces physically separated



- Explicit transfers between the two spaces
- Two entry points on the GPU
 - Global and constant memories

GPU memory hierarchy

- On GPU, 4 levels of memory [+ texture memory]
 - A) Global memory [__device__]
 - B) Constant memory [__device__] __constant__
 - C) Shared memory [__device__] __shared__
 - D) Registers



A) Global Memory

- Large, high latency, no cache
- Data
 - Accessible by all the threads of the grid
 - Lifespan : as required by the application
- From host,
 - Allocation/Free + copies in both ways
- Static declaration from the GPU with keyword `__device__`

Global memory management

- Allocation : **cudaMalloc**(void ** pointer, size_t nbytes)
- Desallocation : **cudaFree**(void* p)
- Cleaning : **cudaMemset**(void * p, int val, size_t nbytes)
- Copy of the data from host :
cudaMemcpy(void *dst, void *src,
size_t nbytes,
enum cudaMemcpyKind direction);

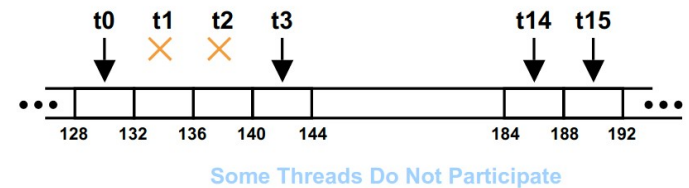
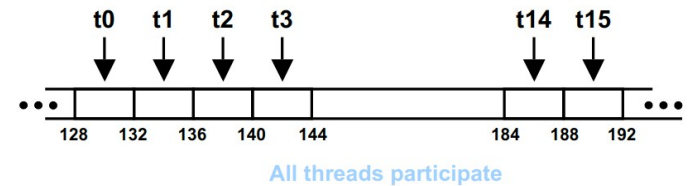
with enum **cudaMemcpyKind**

= { *cudaMemcpyHostToDevice*,
cudaMemcpyDeviceToHost,
cudaMemcpyDeviceToDevice }

Global Memory coalescing

- Multiple memory accesses into a single transaction

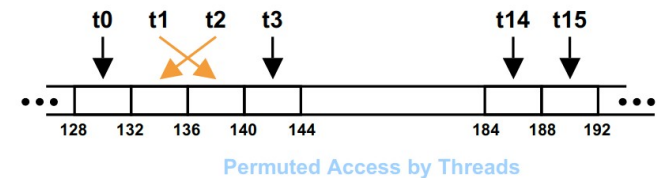
Coalesced Access:
Reading floats



- Uncoalesced load, ie serialized memory access, when memory accesses

- are not sequential
- are sparse
- are misaligned

Uncoalesced Access:
Reading floats



B) Constant Memory

- For data that will not change over a kernel execution
- Read-only, pretty small memory, slow, cached
 - The first read from constant memory costs one memory read from global memory ; after, costs one read from the constant cache
 - Cache for each multiprocessor very small
 - Optimized when warp of threads read same location
- Data accessible by all the threads of the grid

Constant memory management

- Declaration : `__constant__ float buffer [size];`
- Copy of the data from the host :

`cudaError_t cudaMemcpytoSymbol`

`(const char * symbol,
const void * src, size_t count ,
size_t offset=0,
enum cudaMemcpyKind)`

with enum `cudaMemcpyKind`

`= {cudaMemcpyHostToDevice,
cudaMemcpyDeviceToHost,
cudaMemcpyDeviceToDevice}`

C) Shared Memory

- Keyword `__shared__`
 - Separate space with very low latency
- Data
 - Accessible by all threads of the same block
 - Lifetime: kernel run
- Static allocation
 - From the GPU device
 - Static size given
 - at compile time (case a)
 - or at the kernel launch (case b)

```
// case a
__global__ void myKernel(){
    __shared__ int shared[32];
    ...
}
```

```
// case b
__global__ void myKernel(){
    extern __shared__ int s[];
    ...
}
int main() {
    int size= numThreadsPerBlock* sizeof(int);
    myKernel<<< dimGrid, dimBlock, size>>>();}
```

Shared memory management

- All operations on the device within a same kernel
- Static allocation from device : `__shared__ int tab[4] ;`
- Classic explicit initialization/modification in kernel
`for (int i = 0 ; i < 4 ; i++) tab[i]=i ;`



D) Registers

- Fast, only for one thread
- For local kernel variables
 - Allocation of scalar variables in registers
 - Allocation of arrays of more than 4 elements in the global memory
- No specific keyword