W•SENSE
INTEGRATED CABLELESS SOLUTIONS

SAPIENZA
UNIVERSITÀ DI ROMA

Programmazione di sistemi multicore CUDA #2

Fabrizio Gattuso

Avvisi

12 Dicembre **NO LEZIONE PER IT MEETING**

Il secondo esonero sarà svolto insieme al pirmo appello

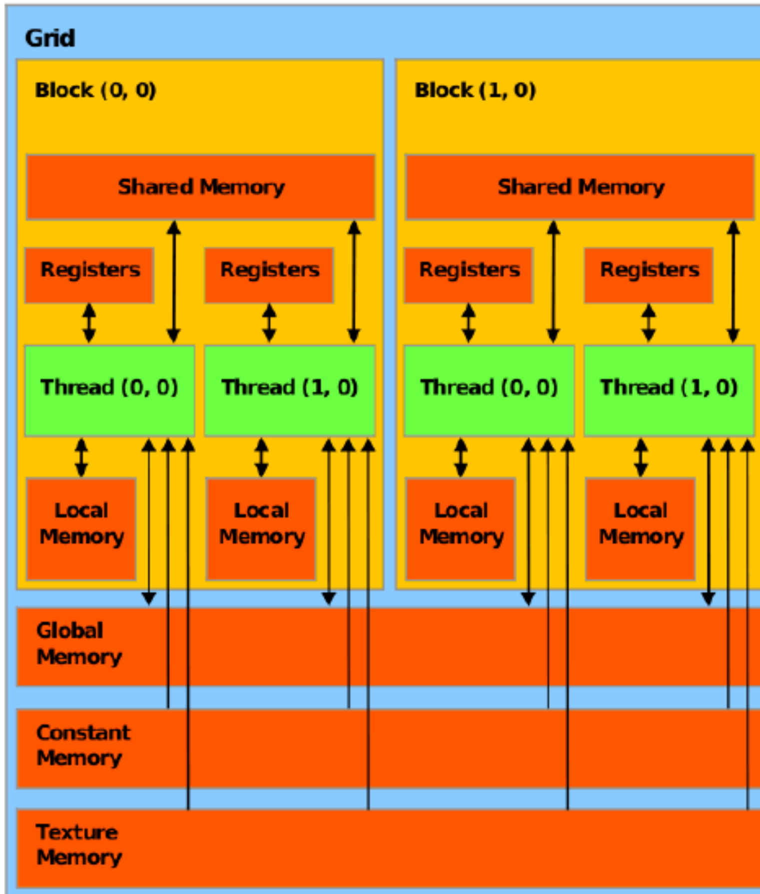
Date esame completo e primo esonero:

- **14 Gennaio** ore 16-18, Aula 2L - Via del Castro Laurenziano 7A
- **4 Febbraio** ore 16-18, Aula 2L - Via del Castro Laurenziano 7A

È uscito il codice OPIS *SIU975JZ* per la **valutazione del corso**

È importante che cominciate ad **inviare la mail con la richiesta materiale per il progetto**

Gestione della memoria



Variabile	Memoria	Scope
int var	register	thread
int array[10]	local	thread
__shared__	shared	block
__device__	global	grid
__constant__	constant	grid

Lavorare con i thread

I thread all'interno dello stesso blocco sono ***utili*** perché possono comunicare tra loro.

Attraverso la direttiva

```
void __synchthreads()
```

Tutti i thread saranno sincronizzati tra loro e solamente quando tutti avranno raggiunto la barriera il codice andrà avanti.

Lavorare con i thread (2)

Questo è utile anche per prevenire tre tipi di hazards noti:

- RAW (Read after write)
- WAR (Write after read)
- WAW (Write after write)

Fare sempre attenzione alle dipendenze temporali dei dati.

Lavorare con i thread (3)

Come già dovreste sapere la memoria utilizzabili per condividere dati tra due thread è quella:

1. **Global**
2. **Shared**

__shared__ per varabili su area di memoria condivisa

Non usare la global se non strettamente necessario, in quanto, è molto più lenta rispetto alla shared. Al Massimo utilizzare la memoria constant.

DimGrid e DimBlock

Come abbiamo visto se si avvia un kernel nel seguente modo:

myKernel<<<N, M>>> verrà lanciato con N blocchi da M threads.

Se si vogliono specificare più nel dettaglio tutte le dimensioni, tra cui la griglia, è necessario usare l’indirizzamento esteso.

dim3 grid(512, 1, 1) o dim3 grid(512)

Dim3 block(1024, 1024,

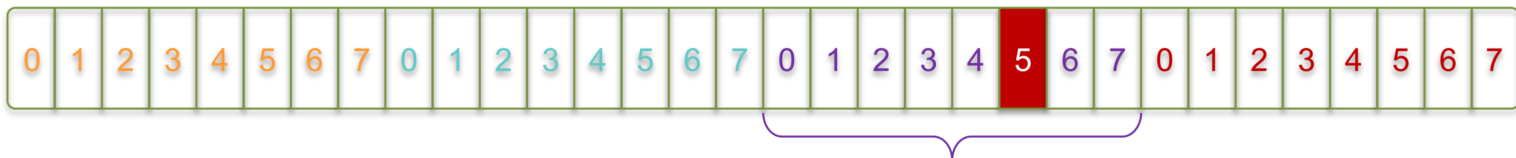
Indexing

Abbiamo già visto come si può ricavare l'id univoco per un'architettura 1D:

index = threadIdx.x + blockIdx.x * blockDim.x

`M = 8`

`threadIdx.x = 5`



Indexing (2)

Index2D = blockIdx.x * blockDim.x * blockDim.y + threadIdx.y * blockDim.x + threadIdx.x

Index3D = blockIdx.x * blockDim.x * blockDim.y * blockDim.z + threadIdx.z * blockDim.y * blockDim.x + threadIdx.y * blockDim.x + threadIdx.x

E questo solo per le griglie in 1D…

Gestione dei device (GPU)

Esistono delle comode funzioni per la gestione del device.

cudaError_t cudaGetDeviceCount (**int*** count)

Ritorna il numero di device disponibili.

cudaError_t cudaGetDevice (**int*** device)

Ritorna il device in uso.

cudaError_t cudaSetDevice (**int** device)

Imposta il device da utilizzare.

cudaError_t cudaGetDeviceProperties (**cudaDeviceProp*** prop, **int** device)

Ritorna la CC del device selezionato.

Gestione degli errori

Gestire gli errori è sempre una pratica essenziale per ogni sviluppatore. Nel calcolo parallelo è essenziale.

cudaError_t è l'enumeration utilizzata per rappresentare gli errori.

cudaSuccess = 0

cudaErrorInvalidValue = 1

cudaErrorMemoryAllocation = 2

cudaErrorInvalidConfiguration = 9

cudaErrorDuplicateVariableName = 43

Gestione degli errori (2)

cudaError_t cudaGetLastError (**void**)

Ritorna l'ultimo errore lanciato.

char* cudaGetErrorString(**cudaError_t** error)

Ritorna una stringa sull'errore passato in input.

Calcolo dello speedup

L'**incremento delle prestazioni** di un programma parallelo rispetto a uno sequenziale è detto **speedup**.

Si può calcolare come: **tempo sequenziale / tempo parallelo**

$$\text{Speedup}(N) = \frac{1}{(1-P)+\frac{P}{N}}$$

Serial part of job = 1 (100%) - Parallel part

Parallel part is divided up by N workers

Come scegliere i parametri da passare al kernel

1. Guardare sempre I limiti massimi dell'hw (**warp size**, **SM**, **max blocks**, **max threads per block**)
2. I thread all'interno del blocco dovrebbero essere multipli dello WARP size
3. Partire da numeri grandi di threads e blocks, calcolare lo speed up e provare a diminuire i parametri cercando la soluzione ideale

Compute Capability (CC)

<table>
<tr><th></th><th>FERMI GF100</th><th>FERMI GF104</th><th>KEPLER GK104</th><th>KEPLER GK110</th><th>KEPLER GK210</th></tr>
<tr><td>Compute Capability</td><td>2.0</td><td>2.1</td><td>3.0</td><td>3.5</td><td>3.7</td></tr>
<tr><td>Threads / Warp</td><td colspan="5">32</td></tr>
<tr><td>Max Threads / Thread Block</td><td colspan="5">1024</td></tr>
<tr><td>Max Warps / Multiprocessor</td><td colspan="2">48</td><td colspan="3">64</td></tr>
<tr><td>Max Threads / Multiprocessor</td><td colspan="2">1536</td><td colspan="3">2048</td></tr>
<tr><td>Max Thread Blocks / Multiprocessor</td><td colspan="2">8</td><td colspan="3">16</td></tr>
<tr><td>32-bit Registers / Multiprocessor</td><td colspan="2">32768</td><td colspan="2">65536</td><td>131072</td></tr>
<tr><td>Max Registers / Thread Block</td><td colspan="2">32768</td><td colspan="2">65536</td><td>65536</td></tr>
<tr><td>Max Registers / Thread</td><td colspan="3">63</td><td colspan="2">255</td></tr>
<tr><td>Max Shared Memory / Multiprocessor</td><td colspan="4">48K</td><td>112K</td></tr>
<tr><td>Max Shared Memory / Thread Block</td><td colspan="5">48K</td></tr>
<tr><td>Max X Grid Dimension</td><td colspan="2">2^16-1</td><td colspan="3">2^32-1</td></tr>
<tr><td>Hyper-Q</td><td colspan="3">No</td><td colspan="2">Yes</td></tr>
<tr><td>Dynamic Parallelism</td><td colspan="3">No</td><td colspan="2">Yes</td></tr>
</table>

Piccolo esercizio

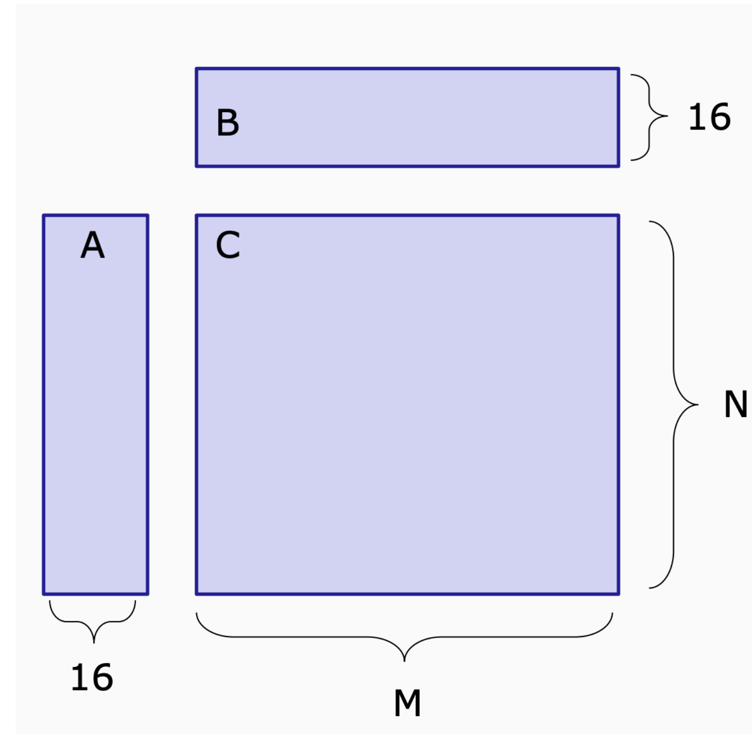
Calcolo parallelo del prodotto di due matrici

C = AB

A una matrice Nx16

B una matrice 16xM

M e N multipli di 16

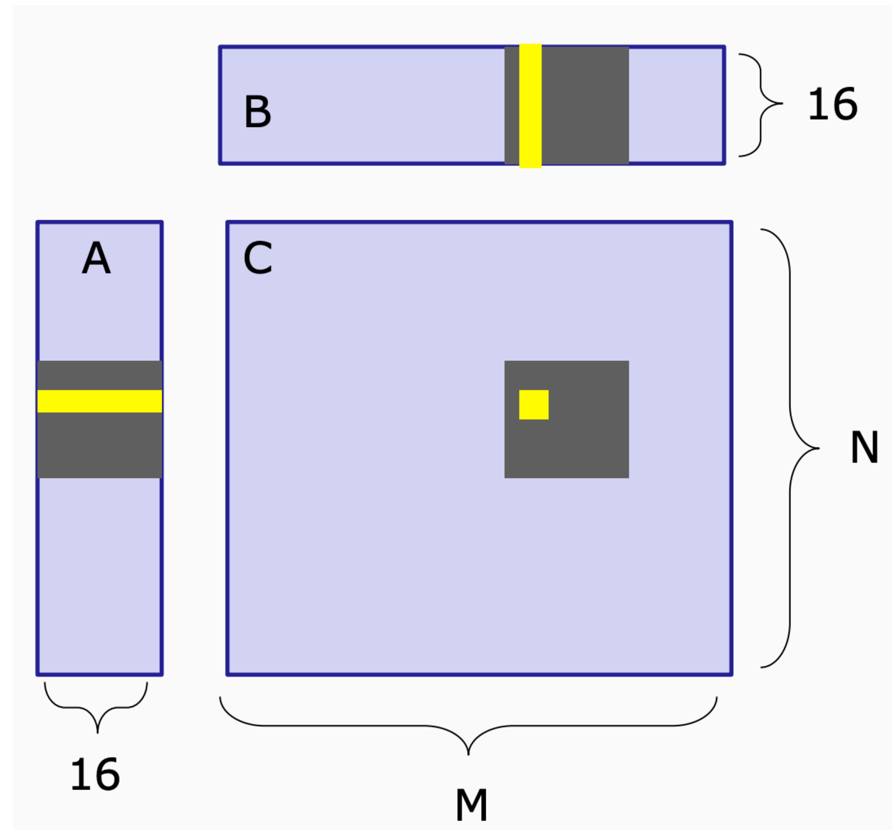


Piccolo esercizio (2)

La matrice viene divisa in tile da 16x16

Ogni tile viene passata a
un blocco di thread

Ogni thread calcola un elemento
della matrice risultante



Piccolo esercizio (3)

```
__global__ void simpleMultiply(float *a, float* b, float
*c, int N) {
    int row = blockIdx.y * blockDim.y + threadIdx.y;
    int col = blockIdx.x * blockDim.x + threadIdx.x;
    float sum = 0.0f;
    for (int i = 0; i < TILE_DIM; i++) {
        sum += a[row*TILE_DIM+i] * b[i*N+col];
    }
    c[row*N+col] = sum;
}
```