GPU Computing

Lab 7

Memoria zero-copy

Trasferimento dati CPU<-->GPU

Uso della host malloc

uso ‘classico’ della device memory

```
// part 1: using device memory
float *h_A, *h_B, *hostRef, *gpuRef;
h_A = (float *) malloc(nBytes);
h_B = (float *) malloc(nBytes);
hostRef = (float *) malloc(nBytes);
gpuRef = (float *) malloc(nBytes);

// initialize data at host side
. . .
// malloc device global memory
float *d_A, *d_B, *d_C;
CHECK(cudaMalloc((float**) &d_A, nBytes));
CHECK(cudaMalloc((float**) &d_B, nBytes));
CHECK(cudaMalloc((float**) &d_C, nBytes));

// transfer data from host to device
CHECK(cudaMemcpy(d_A, h_A, nBytes, cudaMemcpyHostToDevice));
CHECK(cudaMemcpy(d_B, h_B, nBytes, cudaMemcpyHostToDevice));

// set up execution configuration
int iLen = 512;
dim3 block(iLen);
dim3 grid((nElem + block.x - 1) / block.x);

sumArrays<<<grid, block>>>(d_A, d_B, d_C, nElem);
```

Uso della zero-copy

uso della zero-copy memory

ottiene il puntatore nello spazio indiriz. della GPU

esegue kernel nel modo usuale… con i riferimenti alla device memory

```
// part 2: using zerocopy memory for array A and B

CHECK(cudaHostAlloc((void **) &h_A, nBytes, cudaHostAllocMapped));
CHECK(cudaHostAlloc((void **) &h_B, nBytes, cudaHostAllocMapped));


// initialize data at host side
. . .

// pass the pointer to device
CHECK(cudaHostGetDevicePointer((void **) &d_A, (void *) h_A, 0));
CHECK(cudaHostGetDevicePointer((void **) &d_B, (void *) h_B, 0));


// add at host side for result checks
sumArraysOnHost(h_A, h_B, hostRef, nElem);


// execute kernel with zero copy memory
sumArraysZeroCopy<<<grid, block>>>(d_A, d_B, d_C, nElem);


// copy kernel result back to host side
CHECK(cudaMemcpy(gpuRef, d_C, nBytes, cudaMemcpyDeviceToHost));
```

Prestazioni

Il tempo di esecuzione del kernel **`sumArraysZeroCopy`** circa 3 volte più lento de kernel **`sumArrays`** che usa device memory

```
pascal[~/cuda-workspace/lab6-zeroCopy]->nvprof Release/lab6-zeroCopy
Using Device 0: Tesla P100-PCIE-16GB Vector size 4194304 power 22  nbytes   16 MB
==12471== Profiling result:
 Time(%)   Time     Calls       Avg       Min       Max  Name
 38.81%  3.7331ms         2  1.8666ms  1.8357ms  1.8974ms  [CUDA memcpy HtoD]
 30.96%  2.9781ms         1  2.9781ms  2.9781ms  2.9781ms  sumArraysZeroCopyOffset(. . .)
 29.26%  2.8146ms         1  2.8146ms  2.8146ms  2.8146ms  [CUDA memcpy DtoH]
  0.97%  93.632us         1  93.632us  93.632us  93.632us  sumArrays(. . . )
```

SoA vs AoS

confronti

Esercitazione

Elaborazione immagini:
Scrivere un programma CUDA per i kernel sotto definiti mettendo in evidenza vantaggi e svantaggi nell'organizzare i dati nel formato AoS o SoA .

Usare nell'analisi di prestazioni:
1. i tempi di esecuzione
2. Profili di storage con le metriche **global load efficiency** e **globalstore effi ciency**

```
/*
 * Riscala l'immagine al valore massimo [max] fissato
 */
__global__ void rescaleImg(<tipo immagine> *img, const int max, const int n)
```

```
/*
 * cancella un piano dell'immagine [plane = 'r' o 'g' o 'b'] fissato
 */
__global__ void deletePlane(<tipo immagine> *img, const char plane, const int n)
```

Trasposizione matrice

uso di SMEM

Matrice trasposta con SMEM

Dimensione 2D di blocco

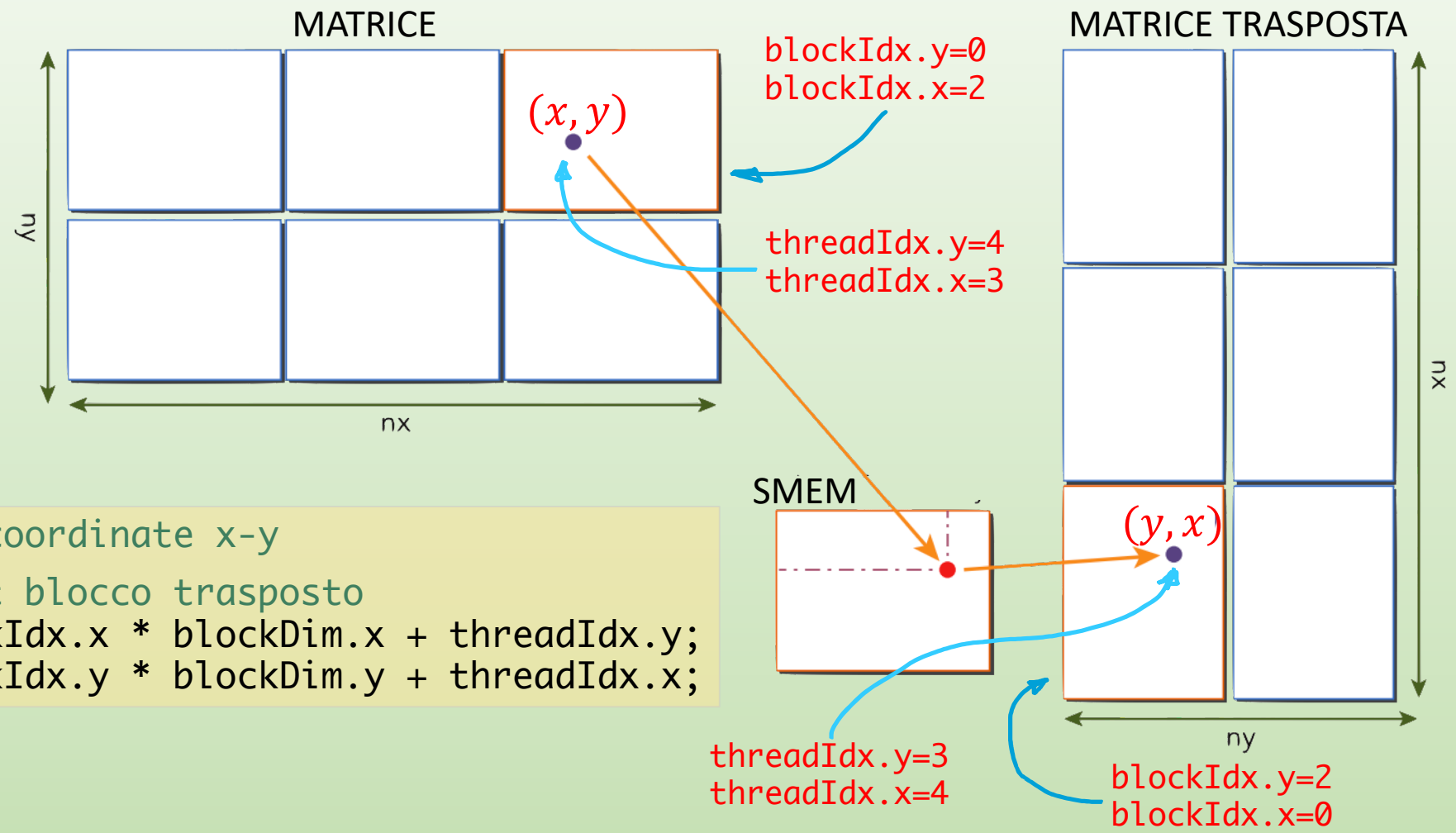
Dichiarazione mem shared di taglia pari al blocco thread

Il warp scrive i dati nella **shared memory** in row-major ordering evitando bank conflict sulle scritture
Ogni warp fa una letture coalescente dei dati in **global memory**

Sincronizzazione dei thread

```
// Dimensione del blocco
#define BDIMX 32
#define BDIMY 32
// macro x conversione indici lineari
#define INDEX(row, col, stride) (row * stride + col)

__global__ void transposeSmem(float *out, float *in, int nrows, int ncols) {
   // shared memory statica
   __shared__ float tile[BDIMY][BDIMX];

   // coordinate matrice originale: elem.  A_row,col
   unsigned int row = blockDim.y * blockIdx.y + threadIdx.y;
   unsigned int col = blockDim.x * blockIdx.x + threadIdx.x;

   // trasferimento dati dalla global memory alla shared memory
   if (row < nrows && col < ncols)
      tile[threadIdx.y][threadIdx.x] = in[INDEX(row, col, ncols)];

   // thread synchronization
   __syncthreads();
```

Dettagli nella trasposizione




```
// swap coordinate x-y
// offset blocco trasposto
y = blockIdx.x * blockDim.x + threadIdx.y;
x = blockIdx.y * blockDim.y + threadIdx.x;
```

```
out[y*nrows + x] = tile[threadIdx.x][threadIdx.y];
```

Matrice trasposta con SMEM

Indici di colonna e riga della matrice trasposta

```
        // offset blocco trasposto
y = blockIdx.x * blockDim.x + threadIdx.y;
x = blockIdx.y * blockDim.y + threadIdx.x;



// controlli invertiti nelle dim riga colonna
if (y < ncols && x < nrows)
    out[y*nrows + x] = tile[threadIdx.x][threadIdx.y];
```

Indice lineare della matrice trasposta

lettura della SMEM per colonna

Prestazioni

Esecuzione con block dim = (32 x 32), nrows = 8192, ncols = 8192

```
starting transpose at device 0: Tesla P100-PCIE-16GB
Matrice con nrows = 8192 ncols = 8192
naiveGmem elapsed 0.004381 sec
 <<< grid (256,256) block (32,32)>>> effective bandwidth 122.546929 GB
transposeSmem elapsed 0.001920 sec
 <<< grid (256,256) block (32,32)>>> effective bandwidth 279.622478 GB
SPEEDUP = 2.281758
```

```
$ nvprof --metrics gld_efficiency,gst_efficiency transpose
```

```
Kernel: naiveGmem(float*, float*, int, int)
          1         gld_efficiency      Global Memory Load Efficiency     100.00%     100.00%     100.00%
          1         gst_efficiency    Global Memory Store Efficiency      12.50%      12.50%      12.50%
Kernel: transposeSmem(float*, float*, int, int)
          1         gld_efficiency      Global Memory Load Efficiency     100.00%     100.00%     100.00%
          1         gst_efficiency     Global Memory Store Efficiency     100.00%     100.00%     100.00%
```

Prestazioni (unified mem)

Esecuzione con block dim = (32 x 32), nrows = 8192, ncols = 8192

```
starting transpose at device 0: Tesla P100-PCIE-16GB
Matrice con nrows = 8192 ncols = 8192
naiveGmem elapsed 0.315243 sec
 <<< grid (256,256) block (32,32)>>> effective bandwidth 1.703038 GB
transposeSmem elapsed 0.173379 sec
 <<< grid (256,256) block (32,32)>>> effective bandwidth 3.096517 GB
SPEEDUP = 1.818231
```

```
$ nvprof --metrics gld_efficiency,gst_efficiency transpose
```

```
Kernel: naiveGmem(float*, float*, int, int)
          1         gld_efficiency      Global Memory Load Efficiency     100.00%     100.00%     100.00%
          1         gst_efficiency    Global Memory Store Efficiency      12.50%      12.50%      12.50%
Kernel: transposeSmem(float*, float*, int, int)
          1         gld_efficiency      Global Memory Load Efficiency     100.00%     100.00%     100.00%
          1         gst_efficiency     Global Memory Store Efficiency     100.00%     100.00%     100.00%
```