Lab 7a - CUDA base

https://elly2023.smfi.unipr.it/mod/page/view.php?id=5580

OBIETTIVO
Sviluppare ed eseguire applicazioni GPU utilizzando CUDA-Toolkit.

Teoria

Architettura GPU vs CPU

La GPU è specializzata per il calcolo «data parallel» e i transistor del chip sono dedicati maggiormente al processamento dei dati piuttosto che al caching o al controllo di flusso.

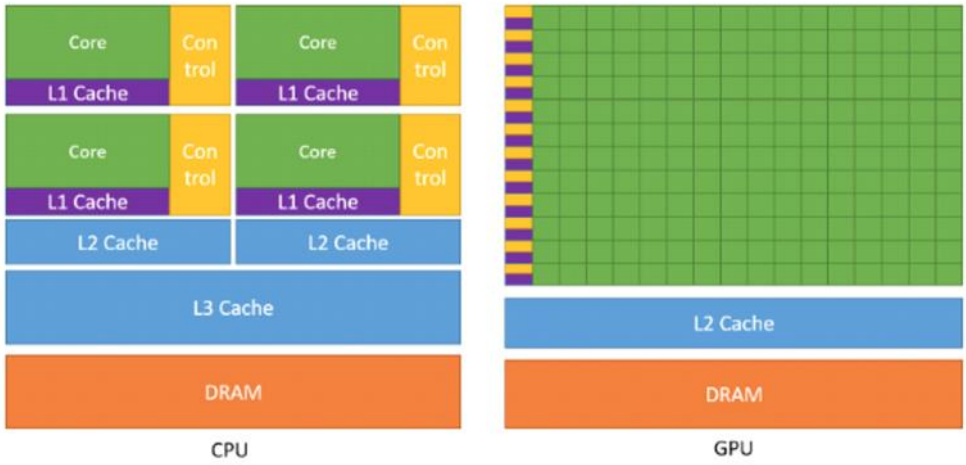
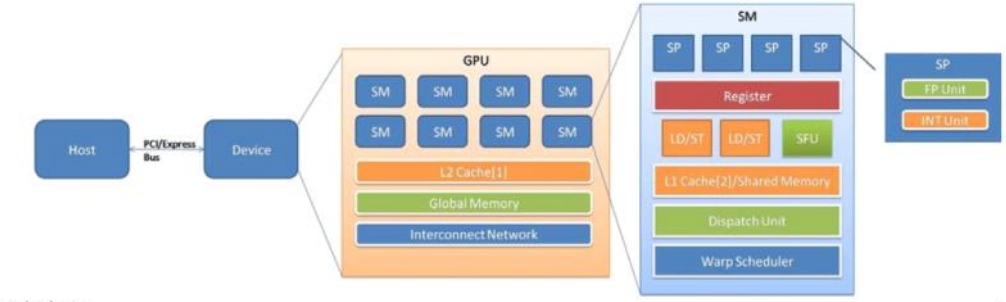


Figure 1: *The GPU Devotes More Transistors to Data Processing*

Architettura GPU: SP, Warp e SM

L'unità elementare di calcolo è il **Thread**. Un thread viene eseguito da un Cuda core (**Streaming Processor** – SP).
I thread vengono eseguiti in parallelo in gruppi di 32 (Warp) su SP gestiti da uno **Steaming Multiprocessor** (SM).
Ogni SM dispone di N SP (N=64 su P100, V100 e A100), un Warp scheduler, una memoria veloce (**Register**) privata per ogni thread, e una memoria veloce (L1 cache / **Shared memory**) condivisa tra i thread del SM.
Una GPU contiene un certo numero di SM (es. 56 su P100, 108 su A100) e un'ampia **memoria Globale** condivisa tra tutti i thread della GPU.



21/05/2024 5

Esercizi Base

- **Primo esercizio (dalla user guide del cluster HPC)**
  Ho copiato l'esercizio di esempio hello_cuda.cu, dopodiché l'ho compilato ed eseguito.
  In sintesi il programma modifica la stringa 'a' da "Hello " a "World!", sommando ad ogni carattere della prima un offset

per ottenere i caratteri della seconda. Stampando la stringa prima e dopo la modifica si ottiene in outpu "Hello World!"

hello_cuda.cu

```
/*
cp /hpc/share/samples/cuda/hello_cuda.cu .
module load cuda
nvcc hello_cuda.cu -o hello_cuda
*/

// This is the REAL "hello world" for CUDA!
// It takes the string "Hello ", prints it, then passes it to CUDA with an array
// of offsets. Then the offsets are added in parallel to produce the string "World!"
// By Ingemar Ragnemalm 2010

#include <stdio.h>

const int N = 7;
const int blocksize = 7;

// Kernel (memoria globale)
__global__
void hello(char *a, int *b)
{
    // threadIdx.x = indice thread nel blocco, va da 0 a 6
    a[threadIdx.x] += b[threadIdx.x];
}

int main()
{
    char a[N] = "Hello ";
    int b[N] = {15, 10, 6, 0, -11, 1, 0};

    // creo le strutture dati sull'host
    char *ad; // device a
    int *bd;  // device b
    const int csize = N*sizeof(char);
    const int isize = N*sizeof(int);

    printf("%s", a);

    // creo le strutture dati sulle GPU
    cudaMalloc( (void**)&ad, csize );
    cudaMalloc( (void**)&bd, isize );
    cudaMemcpy( ad, a, csize, cudaMemcpyHostToDevice ); //copia 'ad' host-GPU (bloccante)
    cudaMemcpy( bd, b, isize, cudaMemcpyHostToDevice ); //copia 'bd' host-GPU (bloccante)

    // singola dimensione del blocco di thread utilizzato per eseguire il kernel (singolo blocco fatto di 7 thread)
    dim3 dimBlock( blocksize, 1 );

    // configurazione della griglia di blocchi (una sola cella (blocco), sia lungo x che y)
    dim3 dimGrid( 1, 1 );

    // chiamata del kernel (asincrona)
    hello<<<dimGrid, dimBlock>>>(ad, bd);

    // copia 'ad' GPU-host (bloccante)
    cudaMemcpy( a, ad, csize, cudaMemcpyDeviceToHost );

    // libera la memoria allocata sulla GPU
    cudaFree( ad );

    printf("%s\n", a);
    return EXIT_SUCCESS;
}
```

Hello_cuda.slurm

```
#!/bin/sh
#< 1 node with 1 GPU
#SBATCH --partition=gpu
#SBATCH --qos=gpu
#SBATCH --nodes=1
#SBATCH --gres=gpu:p100:1
#SBATCH --time=0-00:30:00
#SBATCH --mem=4G
#< Charge resources to account
##SBATCH --account=<account>

module load cuda

nvcc hello_cuda.cu -o hello_cuda

echo "# SLURM_JOB_NODELIST  : $SLURM_JOB_NODELIST"
echo "# CUDA_VISIBLE_DEVICES: $CUDA_VISIBLE_DEVICES"

./hello_cuda > hello_cuda.dat
```

Output

```
[martina.genovese@ui01 base]$ sbatch hello_cuda.slurm
Submitted batch job 2191799
[martina.genovese@ui01 base]$ cat hello_cuda.dat
Hello World!
```

- **Altri esercizi**

  Ho copiato i file da "/hpc/home/roberto.alfieri/SHARE/cuda/base/", dopodiché ho testato tutti i programmi tramite lo script launch.slurm fornito

  - cuda_printf.cu # in un kernel <<<10,10>>> ogni thread stampa le proprie coordinate

```
[martina.genovese@ui01 base]$ cat cuda_printf.dat
#SLURM_JOB_NODELIST   : wn41
#CUDA_VISIBLE_DEVICES : 0
Hello from block 0, thread 0
Hello from block 0, thread 1
Hello from block 0, thread 2
Hello from block 0, thread 3
Hello from block 0, thread 4
Hello from block 0, thread 5
Hello from block 0, thread 6
Hello from block 0, thread 7
Hello from block 0, thread 8
Hello from block 0, thread 9
Hello from block 6, thread 0
Hello from block 6, thread 1
Hello from block 6, thread 2
Hello from block 6, thread 3
Hello from block 6, thread 4
Hello from block 6, thread 5
Hello from block 6, thread 6
Hello from block 6, thread 7
Hello from block 6, thread 8
Hello from block 6, thread 9
Hello from block 3, thread 0
Hello from block 3, thread 1
Hello from block 3, thread 2
Hello from block 3, thread 3
Hello from block 3, thread 4
Hello from block 3, thread 5
Hello from block 3, thread 6
Hello from block 3, thread 7
Hello from block 3, thread 8
Hello from block 3, thread 9
Hello from block 9, thread 0
Hello from block 9, thread 1
Hello from block 9, thread 2
Hello from block 9, thread 3
Hello from block 9, thread 4
Hello from block 9, thread 5
Hello from block 9, thread 6
Hello from block 9, thread 7
Hello from block 9, thread 8
```

```
Hello from block 9, thread 9
Hello from block 5, thread 0
Hello from block 5, thread 1
Hello from block 5, thread 2
Hello from block 5, thread 3
Hello from block 5, thread 4
Hello from block 5, thread 5
Hello from block 5, thread 6
Hello from block 5, thread 7
Hello from block 5, thread 8
Hello from block 5, thread 9
Hello from block 1, thread 0
Hello from block 1, thread 1
Hello from block 1, thread 2
Hello from block 1, thread 3
Hello from block 1, thread 4
Hello from block 1, thread 5
Hello from block 1, thread 6
Hello from block 1, thread 7
Hello from block 1, thread 8
Hello from block 1, thread 9
Hello from block 7, thread 0
Hello from block 7, thread 1
Hello from block 7, thread 2
Hello from block 7, thread 3
Hello from block 7, thread 4
Hello from block 7, thread 5
Hello from block 7, thread 6
Hello from block 7, thread 7
Hello from block 7, thread 8
Hello from block 7, thread 9
Hello from block 4, thread 0
Hello from block 4, thread 1
Hello from block 4, thread 2
Hello from block 4, thread 3
Hello from block 4, thread 4
Hello from block 4, thread 5
Hello from block 4, thread 6
Hello from block 4, thread 7
Hello from block 4, thread 8
Hello from block 4, thread 9
Hello from block 2, thread 0
Hello from block 2, thread 1
Hello from block 2, thread 2
Hello from block 2, thread 3
Hello from block 2, thread 4
Hello from block 2, thread 5
Hello from block 2, thread 6
Hello from block 2, thread 7
Hello from block 2, thread 8
Hello from block 2, thread 9
Hello from block 8, thread 0
Hello from block 8, thread 1
Hello from block 8, thread 2
Hello from block 8, thread 3
Hello from block 8, thread 4
Hello from block 8, thread 5
Hello from block 8, thread 6
Hello from block 8, thread 7
Hello from block 8, thread 8
Hello from block 8, thread 9
```

- sumArray.cu # 2 vettori di float sull'host vengono copiati su GPU, sommati e copiato da GPU a host il $

```
[martina.genovese@ui01 base]$ cat sum_array.dat
#SLURM_JOB_NODELIST   : wn41
#CUDA_VISIBLE_DEVICES : 0
Host> A: 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0
Host> B: 1.0 2.0 3.0 4.0 5.0 6.0 7.0 8.0 9.0 10.0
sommaarray<<<DimGrid,DimBlock>>>
Host> C: 2.0 4.0 6.0 8.0 10.0 12.0 14.0 16.0 18.0 20.0
```

- matMul.cu # 2 matrici NxN vengono copiate su GPU con 1 blocco 2D e moltiplicate

```
[martina.genovese@ui01 base]$ cat matMul.dat
#SLURM_JOB_NODELIST   : wn41
#CUDA_VISIBLE_DEVICES : 0

Host> A:
-1.0 0.0 1.0
2.0 3.0 4.0
5.0 6.0 7.0

Host> B:
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0

matMul<<<DimGrid,DimBlock>>>

Host> C:
6.0 6.0 6.0
42.0 51.0 60.0
78.0 96.0 114.0
```

- variabili.cu # float A[16] viene copiato sulla GPU che somma elementi (riduzione somma) e torna il risu$

```
[martina.genovese@ui01 base]$ cat variabili.dat
#SLURM_JOB_NODELIST   : wn41
#CUDA_VISIBLE_DEVICES : 0
variabili.cu:1:1: error: unterminated comment
 /*
 ^

Host> A:
-1.0 0.0 1.0
2.0 3.0 4.0
5.0 6.0 7.0

Host> B:
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0

matMul<<<DimGrid,DimBlock>>>

Host> C:
6.0 6.0 6.0
42.0 51.0 60.0
78.0 96.0 114.0
```

- managed.cu # il vettore Vett e' managed; la GPU calcola il quadrato di ogni elemento

```
[martina.genovese@ui01 base]$ cat managed.dat
#SLURM_JOB_NODELIST   : wn41
#CUDA_VISIBLE_DEVICES : 0
0: Vett*Vett = 0
1: Vett*Vett = 1
2: Vett*Vett = 4
3: Vett*Vett = 9
4: Vett*Vett = 16
5: Vett*Vett = 25
6: Vett*Vett = 36
7: Vett*Vett = 49
8: Vett*Vett = 64
9: Vett*Vett = 81
10: Vett*Vett = 100
11: Vett*Vett = 121
12: Vett*Vett = 144
13: Vett*Vett = 169
14: Vett*Vett = 196
15: Vett*Vett = 225
16: Vett*Vett = 256
17: Vett*Vett = 289
18: Vett*Vett = 324
19: Vett*Vett = 361
20: Vett*Vett = 400
21: Vett*Vett = 441
22: Vett*Vett = 484
23: Vett*Vett = 529
24: Vett*Vett = 576
25: Vett*Vett = 625
26: Vett*Vett = 676
27: Vett*Vett = 729
28: Vett*Vett = 784
29: Vett*Vett = 841
30: Vett*Vett = 900
31: Vett*Vett = 961
```

- race_condition.cu # int a=0 viene copiato su GPU <<<100,100>>>, ogni thread somma 1 (race). Timing

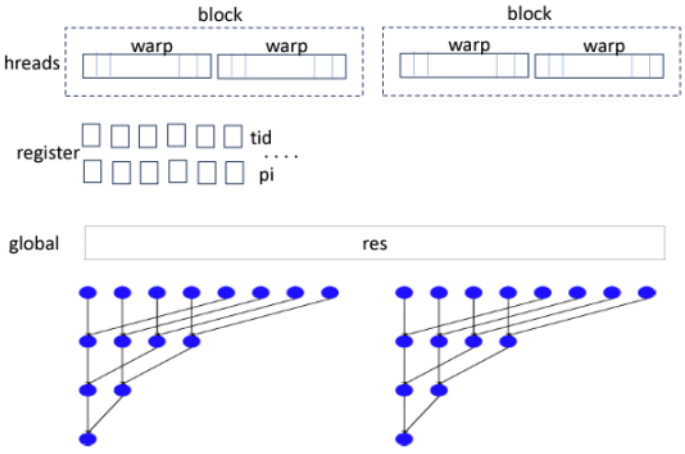
```
[martina.genovese@ui01 base]$ cat race_condition.dat
#SLURM_JOB_NODELIST   : wn41
#CUDA_VISIBLE_DEVICES : 0
GPUTime elapsed: 0.000046 seconds
a=1
```

- sezione_critica.cu # Sezione critica a livello warp o blocco con atomicCAS e atomicExch

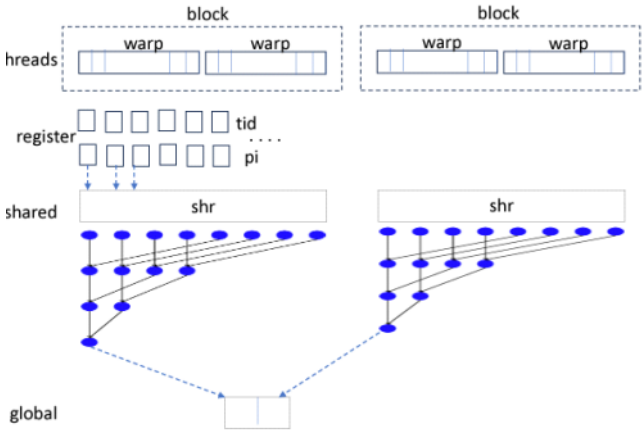
```
[martina.genovese@ui01 base]$ cat sezione_critica.dat
#SLURM_JOB_NODELIST   : wn41
#CUDA_VISIBLE_DEVICES : 0
Block 0 thread 0 entering critical section
Block 1 thread 0 entering critical section
finito: 2
```

Calcolo di Pigreco

cpi_gpu_global



cpi_gpu_shr



Ho eseguito tramite cpi_gp.slurm i programmi cpi_gpu_global e cpi_gpu_shr per il calcolo di pi-greco con GPU utilizzando la memoria global e la memoria shared.

cpi_gpu_global.cu

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>    //optarg
#include <time.h>

#define BILLION  1000000000L;

void options(int argc, char * argv[]) ;

int n=500000000;    // intervalli
int nblocks=128;    // numero blocchi
int  threadsPerBlock = 1024;
const double  PI = 3.14159265358979323846264338327950288 ;

__global__ void add( float *res1 ) {

       long int tid = threadIdx.x + blockIdx.x * blockDim.x;

        double h = 1.0 / (double)(gridDim.x * blockDim.x);
        double x = h * ((double)tid - 0.5);
        double pi1 = (1.0 / (1.0 + x*x)); // f1
     res1[tid] = pi1 * 4 * h;

         __syncthreads();
    // for reductions, threadsPerBlock must be a power of 2 // because of the following code
   int i = blockDim.x/2;
   while (i != 0) {
            if (threadIdx.x < i)
      res1[tid] += res1[tid + i];
      __syncthreads();
      i /= 2;
   }

}

/*****************************************/

int main(int argc, char **argv ) {

    options(argc, argv);  /* optarg management */

    n=nblocks*threadsPerBlock;

    float* res1=(float*)malloc(n*sizeof(float));
    float *dev_res1;
    cudaMalloc( (void**)&dev_res1, n*sizeof(float) );

    struct timespec t1,t2,t3;
    double wtime, ktime;
    clock_gettime( CLOCK_REALTIME ,          &t1) ;

    add<<<nblocks,threadsPerBlock>>>( dev_res1 );
    cudaDeviceSynchronize();
    clock_gettime( CLOCK_REALTIME ,          &t2) ;

    cudaMemcpy( res1, dev_res1, n*sizeof(float), cudaMemcpyDeviceToHost );

    float total1=0;
    for (long int i=0;i<n;i+=threadsPerBlock)  total1+=res1[i];

    clock_gettime( CLOCK_REALTIME ,          &t3 ) ;

    wtime = (double) ( t3.tv_sec  - t1.tv_sec )
          + (double) ( t3.tv_nsec - t1.tv_nsec )
            / BILLION;



    ktime = (double)  ( t2.tv_sec  - t1.tv_sec )
           + (double) ( t2.tv_nsec - t1.tv_nsec )
            / BILLION;



    fprintf(stderr,"#intervals blocks pi error wtime(s) ktime(s) \n");
    fprintf(stderr,"CUDA, %ld, %d, %d, %.10f, %.10e, %.4f,  %.4f \n",
          n, nblocks, threadsPerBlock, total1,  fabs(total1 - PI), wtime, ktime);
    cudaFree( dev_res1 );
```

```
    return 0;
}



/**********************************************/

void options(int argc, char * argv[])
{
  int i;
   while ( (i = getopt(argc, argv, "t:b:h")) != -1) {
        switch (i)
        {
        case 'b':  nblocks         = strtol(optarg, NULL, 10);  break;
        case 't':  threadsPerBlock = strtol(optarg, NULL, 10);  break;
        case 'h':  printf ("\n%s [-b blocks] [-h]",argv[0]); exit(1);
        default:   printf ("\n%s [-b blocks] [-h]",argv[0]);  exit(1);
        }
    }
}
```

cpi_gpu_shr.cu

```
#include <stdio.h>
#include <math.h>
#include <unistd.h>    //optarg
#include <time.h>

#define BILLION  1000000000L;

void options(int argc, char * argv[]) ;



int n=500000000;    // intervalli
int nblocks=128;    // numero blocchi
int   threadsPerBlock = 1024;
const int    max_threadsPerBlock = 1024;
const double  PI = 3.14159265358979323846264338327950288 ;



__global__ void add( float *res1 ) {
        __shared__ float shr1[max_threadsPerBlock];

       int tid = threadIdx.x + blockIdx.x * blockDim.x;

        double h = 1.0 / (double)(gridDim.x * blockDim.x);
        double x = h * ((double)tid - 0.5);
        double pi1 = (1.0 / (1.0 + x*x)); // f1
//        double pi2 = sqrt(1-x*x); // f2

     shr1[threadIdx.x] = pi1 * 4 * h;

          __syncthreads();
    // for reductions, threadsPerBlock must be a power of 2 // because of the following code
   int i = blockDim.x/2;
   while (i != 0) {
            if (threadIdx.x < i)
      shr1[threadIdx.x] += shr1[threadIdx.x + i];
      __syncthreads();
      i /= 2;
   }

   if (threadIdx.x==0)      res1[blockIdx.x] = shr1[threadIdx.x];
}

/**********************************************/

int main(int argc, char **argv ) {

    options(argc, argv);  /* optarg management */

    n=nblocks*threadsPerBlock;

    float* res1=(float*)malloc(nblocks*sizeof(float));
    float *dev_res1;
    cudaMalloc( (void**)&dev_res1, nblocks*sizeof(float) );



    struct timespec t1,t2,t3;
```

```
    double wtime,ktime;
    clock_gettime( CLOCK_REALTIME ,          &t1) ;

    add<<<nblocks,threadsPerBlock>>>( dev_res1);
    cudaDeviceSynchronize();

    clock_gettime( CLOCK_REALTIME ,          &t2) ;

    cudaMemcpy( res1, dev_res1, nblocks*sizeof(float), cudaMemcpyDeviceToHost );

    float total1=0;
    for (int i=0;i<nblocks;i++)    total1+=res1[i];

    clock_gettime( CLOCK_REALTIME ,          &t3) ;

    wtime = (double) ( t3.tv_sec  - t1.tv_sec )
            + (double) ( t3.tv_nsec - t1.tv_nsec )
            / BILLION;



    ktime = (double) ( t2.tv_sec  - t1.tv_sec )
            + (double) ( t2.tv_nsec - t1.tv_nsec )
            / BILLION;



   fprintf(stderr,"#intervals blocks pi error wtime(s)  ktime(s)\n");
    fprintf(stderr,"CUDA, %d, %d, %d,  %.10f, %.10e, %.4f,  %.4f \n", n, nblocks, threadsPerBlock, total1,  fabs(total1 - PI),
wtime, ktime);
    cudaFree( dev_res1 );

    return 0;
}

/*********************************************/

void options(int argc, char * argv[])
{
  int i;
   while ( (i = getopt(argc, argv, "t:b:h")) != -1) {
        switch (i)
        {
        case 'b':  nblocks  = strtol(optarg, NULL, 10);  break;
        case 't': threadsPerBlock    = strtol(optarg, NULL, 10);  break;
        case 'h':  printf ("\n%s [-b blocks] [-h]",argv[0]); exit(1);
        default:   printf ("\n%s [-b blocks] [-h]",argv[0]);  exit(1);
        }
    }
}
```

File di output

```
[martina.genovese@ui01 cpi]$ more cpi_gpu.slurm.o2191772
#SLURM_JOB_NODELIST   : wn41
#CUDA_VISIBLE_DEVICES : 0
#intervals blocks pi error wtime(s) ktime(s)
CUDA, 1048576, 1024, 1024, 3.1415929794, 3.2584135923e-07, 0.0026,  0.0001
#intervals blocks pi error wtime(s)  ktime(s)
CUDA, 1048576, 1024, 1024,  3.1415929794, 3.2584135923e-07, 0.0002,  0.0002
```

Ho poi creato lo script cpi_gpu_scaling.slurm in modo da valutare la scalabilità dei parametri significativi.

cpi_gpu_scaling.slurm

```
#!/bin/sh

#SBATCH --output=%x.o%j
#SBATCH --partition=gpu
#SBATCH --qos=gpu
#SBATCH --gres=gpu:p100:1
#SBATCH --time=0-00:05:00
#SBATCH --mem=12G

#stampa il nome del nodo assegnato e argomenti
echo "#SLURM_JOB_NODELIST   : $SLURM_JOB_NODELIST"
echo "#CUDA_VISIBLE_DEVICES : $CUDA_VISIBLE_DEVICES"

module load cuda

threadsPerBlock=1024
# nblocks=1024
```

```
nvcc cpi_gpu_global.cu -o cpi_gpu_global

for nblocks in 256 512 1024 2048 4096 8192
do
    cpi_gpu_global -b $nblocks -t $threadsPerBlock 2>> cpi_gpu_scaling_global.dat
done



nvcc cpi_gpu_shr.cu -o cpi_gpu_shr

for nblocks in 256 512 1024 2048 4096 8192
do
    cpi_gpu_shr -b $nblocks -t $threadsPerBlock 2>> cpi_gpu_scaling_shr.dat
done
```

Ho infine creato il progrmma in python cpi_gpu_scaling.py per confrontare le prestazioni delle varie implementazioni del programma.

cpi_gpu_scaling.py

```
#!/usr/bin/env python2

import matplotlib
matplotlib.use('Agg')   # Backend per PNG
import matplotlib.pyplot as plt
import pandas as pd

dfG = pd.read_csv("cpi_gpu_scaling_global.dat", names=["mem", "intervals", "nblock", "threadxblock", "pi", "error", "wtime",
"ktime"])
dfS = pd.read_csv("cpi_gpu_scaling_shr.dat", names=["mem", "intervals", "nblock", "threadxblock", "pi", "error", "wtime", "ktime"])

print(dfG)
print(dfS)

plt.subplot(2,1,1)

plt.title('CPI GPU Scaling')
plt.grid()
plt.xlabel('blocks')
plt.ylabel('time')
plt.yscale('log')
plt.plot(dfG.nblock, dfG.wtime, '-o', label="GPU Global")
plt.plot(dfS.nblock, dfS.wtime, '-o', label="GPU Shared")
plt.legend(shadow=True,loc="best")

plt.subplot(2,1,2)

plt.grid()
plt.xlabel('blocks')
plt.ylabel('speedup')
plt.plot(dfG.nblock, dfG.wtime.iloc[0] / dfG.wtime, '-o', label="GPU Global")
plt.plot(dfS.nblock, dfS.wtime.iloc[0] / dfS.wtime, '-o', label="GPU Shared")
plt.plot(range(1,60),range(1,60),'-r', label='Ideal')

plt.legend(shadow=True,loc="best")

plt.savefig('cpi_gpu_scaling.png')
```

cpi_gpu_scaling.png

…