M2-BIG DATA
GPGPU Chapter 3

Exercice

Réalisé par :
Gaby Maroun

Encadré par :
Dr. Etancelin JM

5 novembre 2020

Objectives

Lean a basic vector addition with basic thread block and grid dimensions specifications.

Instructions

We are interested on computing

$$C = A + B$$

where A, B and C are vectors of a given size n. To check the correctness of the program, A and B are initialized as follows : $A = (i)_i$ and $B = (n - i)_i$.
Complete the given code Chap3_Ex1_vectorAdd.cu to perform the following algorithm :

— Allocate data on host
— Initialize data on host
— Allocate data on device
— Copy data from host to device
— Compute thread block and kernel grid dimensions
— Invoke the CUDA kernel
— Copy results from device to host
— Verify the result correctness
— Free device memory

To understand the thread block and kernel grid dimensions, you will produce 3 different versions of the program :

1. Use only one block of threads
2. Use only one thread per block
3. Use several threads per block and several blocks

Make sure that your programm is correct for any vector size witout re-compiling.

1-vectorAdd.c

```c
#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>

// Initialize host vectors
void init(int *a, int *b, int n) {
  for (int i=0; i < n; ++i) {
    a[i] = i;
    b[i] = n-i;
  }
}

// Check result correctness
void check(int *c, int n) {
  int i = 0;
  while (i < n && c[i] == n) {
    ++i;
  }
  if (i == n)
    printf("Ok\n");
  else
    printf("Non ok\n");
}


// Cuda kernel
__global__ void add(int *a, int *b, int *c, int n) {
  //@TODO@ : complete kernel code
    int i = threadIdx.x + blockDim.x * blockIdx.x;
        if(i<n)
            c[i] = a[i] + b[i];


}

int main(int argc, char **argv)
{
  if(argc!=2) {
    printf("Give the vector size as first parameter\n");
    exit(2);
    }

  int n = atoi(argv[1]);
  printf("Vector size is %d\n",n);
  // host pointers
  int *host_a, *host_b, *host_c;
  // Device pointers
  int *dev_a, *dev_b, *dev_c;

  // Allocations on host
  //@TODO@ : complete here
  host_a = (int *) malloc (n * sizeof(int));
  host_b = (int *) malloc (n * sizeof(int));
  host_c = (int *) malloc (n * sizeof(int));

  // Initialize vectors
  init(host_a,host_b,n);

  // Allocations on device
  //@TODO@ : complete here
   cudaMalloc(&dev_a, n * sizeof(int));
   cudaMalloc(&dev_b, n * sizeof(int));
   cudaMalloc(&dev_c, n * sizeof(int));

  // Copy from host to device
  //@TODO@ : complete here
   cudaMemcpy(dev_a, host_a, n * sizeof(int), cudaMemcpyHostToDevice );
   cudaMemcpy(dev_b, host_b, n * sizeof(int), cudaMemcpyHostToDevice );


  // Invoke kernel
  //@TODO@ : complete here
  dim3 DimGrid((n-1)/256+1,1,1);
  dim3 DimBlock(256,1,1);
  add<<< DimGrid, DimBlock>>>(dev_a, dev_b, dev_c, n);

  // Copy result from device to host
  //@TODO@ : complete here
  cudaMemcpy(host_c, dev_c, n * sizeof(int), cudaMemcpyDeviceToHost );


  // Check result
  check(host_c,n);

  // Free device memory
  cudaFree(dev_a); cudaFree(dev_b); cudaFree(dev_c);
  // Free host memory
  free(host_a); free(host_b); free(host_c);
  return 0;
}
```

The execution result came as follow :

```
gmaroun@scinfe058:~/Bureau/stockage/Semestre 3/GPGPU/Chap3$ nvcc 1-vectorAdd.cu
gmaroun@scinfe058:~/Bureau/stockage/Semestre 3/GPGPU/Chap3$ nvprof --print-gpu-trace ./a.out 1000
Vector size is 1000
==22995== NVPROF is profiling process 22995, command: ./a.out 1000
Ok
==22995== Profiling application: ./a.out 1000
==22995== Profiling result:
   Start  Duration            Grid Size      Block Size     Regs*    SSMem*    DSMem*      Size  Throughput  SrcMemType  DstMemType
         Device   Context    Stream  Name
287.23ms  1.6320us                    -               -         -         -         -  3.9063KB  2.2827GB/s    Pageable      Device
Quadro RTX 4000         1         7  [CUDA memcpy HtoD]
287.24ms  1.2800us                    -               -         -         -         -  3.9063KB  2.9104GB/s    Pageable      Device
Quadro RTX 4000         1         7  [CUDA memcpy HtoD]
289.41ms  1.9200us              (4 1 1)       (256 1 1)        16        0B        0B         -           -           -           -
Quadro RTX 4000         1         7  add(int*, int*, int*, int) [112]
289.41ms  2.2720us                    -               -         -         -         -  3.9063KB  1.6397GB/s      Device    Pageable
Quadro RTX 4000         1         7  [CUDA memcpy DtoH]

Regs: Number of registers used per CUDA thread. This number includes registers used internally by the CUDA driver and/or tools and ca
n be more than what the compiler shows.
SSMem: Static shared memory allocated per CUDA block.
DSMem: Dynamic shared memory allocated per CUDA block.
SrcMemType: The type of source memory accessed by memory operation/copy
DstMemType: The type of destination memory accessed by memory operation/copy
```

Questions

1. How many floating operations are being performed in your vector add kernel ? EXPLAIN.
   The are n floating operations being performed as we indicate to be filled in the $argc$ *which means here we have a 1000.*
2. How many global memory reads are being performed by your kernel ? EXPLAIN.
   It's equal to n+n=2n for each host to device
3. How many global memory writes are being performed by your kernel ? EXPLAIN.
   It's equal to n for each device to host
4. Which version is the most efficient (use a size of n = 1000) ? Explain why. Use the NVIDIA profiler to get kernel execution time : nvprof –print-gpu-trace 1-vectorAdd 1000.
   As we can see from the execution, the most efficient is the second one with 287.24ms and 1.28us in addition to a throughput of 2.9GB/s

La fin.