Array_Add_Lab

December 6, 2023

Massive parallel programming on GPUs and applications, by Lokman ABBAS TURKI

1 3. Add arrays

1.1 3.1 Objective

The main purpose of this lab is to familiarize students with the CUDA API through the implementation of vector addition. Students will gain hands-on experience by writing both GPU kernel code and the corresponding host code for array addition. The host code for memory allocation on GPU and copies from CPU to GPU and from GPU to CPU will serve as examples for future labs. We also introduce the use of unified memory.

Students are encouraged to use the CUDA documentation, enabling them to discover:

1) the specifications of CUDA API functions within the CUDA_Runtime_API.
2) the examples of how to use the CUDA API functions in CUDA_C_Programming_Guide

1.2 3.2 Content

Your directory has to contain Add.cu file as well as the header file timer.h

Compile Add.cu using

[ ]:
```
!nvcc Add.cu -o ADD
```

Execute DQ using (on Windows machine ./ is not needed)

[ ]:
```
!./ADD
```

As long as you did not include any additional instruction in the file Add.cu, the execution above is supposed to return

(999999500): 999999500 (999999510): 999999510 (999999520): 999999520 (999999530): 999999530 (999999540): 999999540 CPU Timer for the addition on the CPU of vectors: 0.126

Of course, the execution time changes at each call and depends on the host's performance.

1.2.1 3.2.1 Addition operation on the device with explicit data transfer and CPU timer

a) Allocate aGPU, bGPU, cGPU on the GPU using cudaMalloc.

b) Transfer the values of a, b to aGPU, bGPU using cudaMemcpy.

c) Write the kernel addVect_k that adds aGPU to bGPU and puts the result in cGPU

d) Transfer the values of cGPU to c using cudaMemcpy.

e) Free the GPU memory using cudaFree.

f) Call the kernel addVect_k instead of the function addVect.

g) Do not forget to use cudaDeviceSynchronize after calling the kernel.

1.2.2 3.2.2 Few Optimizations and GPU timer

a) Change the CPU timer with the GPU timer using cudaEvent (in milliseconds).

b) Check for yourself that using threadIdx.x*gridDim.x + blockIdx.x to access the global memory is a very bad choice.

c) What if you compute the execution time of both calling addVect_k and transferring data?

d) Profile further your code using: !nvprof ./ADD

e) Fix the slow transfer of a, b to aGPU, bGPU using the initialization on the device. Now, we can remove the arrays a, and b.

f) Allocate aGPU, bGPU, and cGPU using cudaMallocManaged on the unified memory. Now we can also remove the array c.

g) What if we kept the initialization on the host but with all arrays in the unified memory?

h) You can speed up solution g) using cudaMemPrefetchAsync of a and b before calling addVect_k.

[ ]: