MC_Lab

December 6, 2023

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

1 8 Random number generators and Monte Carlo simulation

1.1 8.1 Objective

This is the first laboratory dedicated to Monte Carlo simulation on GPU. Monte Carlo simulation for linear problems is known to be suitable for parallel architectures. It is therefore a simple and realistic exercise to check the assimilation of the different concepts introduced during the course as well as in the other practical exercises. The first and most important element of Monte Carlo is the choice of the appropriate random number generator (RNG). RNGs are statically defined by a known inductive expression and dynamically specified by their state. In a parallel implementation, if we associate an RNG with each thread, the size of the state becomes an issue. Indeed, a large state multiplied by the number of threads occupies a significant cache memory space, and the latter becomes less available for other tasks.

The NP function is used to check the simulation results and thus it should not be modified.

As usual, do not forget to use CUDA documentation, especially:

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

Add to this,

3) the documentation of the CUDA random number generation library cuRAND Library.

1.2 8.2 Content

Compile MC.cu using

[ ]:
```
!nvcc MC.cu -o MC
```

Execute MC using (on Microsoft Windows OS ./ is not needed)

[ ]:
```
!./MC
```

As long as you did not include any additional instruction in the file MC.cu, the execution above is to return 0. At least, no compilation error is detected by the compiler!

1.2.1 8.2.1 (Pseudo) Random number generators, init_curand_state_k

Using cuRAND documentation, we need to find a good RNG and understand how it can be used for our simulation.

a) Which of HOST API or DEVICE API should we use for the random numbers generation?

b) Print the size of various state options (XORWOW, Mtgp32_t, and others) and explain why some of them must not be used.

c) Which RNG should you use by default? Explain how its initialization is performed.

d) Complete the syntax of init_curand_state_k

1.2.2 8.2.2 Monte Carlo simulation MC_k1, with a reduction on the host

sum and sum2 are two variables that should respectively contain the estimation of the first and the second moment of the actualized payoff: exp(-rT)*(x-K)_+. Their computation with an average is performed on the host. Thus, one needs only to simulate the realizations of the actualized payoff on the device.

a) Define the arrays of the actualized payoffs and perform the right copy.

b) Inspired by examples from cuRAND documentation, define the kernel MC_k1.

c) In which situation should we copy back the state to the global memory before exiting the kernel MC_k1?

d) See how the execution time changes with respect to n. What do you conclude about the scalability of Monte Carlo simulation?

[ ]: