NMC_Lab

December 6, 2023

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

1 16 From Monte Carlo to nested Monte Carlo simulation

1.1 16.1 Objective

This is the second laboratory dedicated to Monte Carlo simulation on GPU. Monte Carlo simulation for linear problems is suitable for parallel architectures. It is therefore a simple and realistic exercise to check the assimilation of the different concepts introduced in this course. We first continue on the optimization of the previous Monte Carlo implementation and perform the reduction directly on the device. Afterwards, we perform a nested Monte Carlo simulation with two layers of parallelization granularities. When the parallel computations are equivalent, in terms of computational complexity and memory storage, it is better to serialize the computations on the inner finer granularity than on the outer coarser granularity.

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 16.2 Content

Copy your previous solution MC.cu, then compile it using

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

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

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

1.2.1 16.2.1 Monte Carlo simulation MC_k2, with a reduction on the device

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)_+. This time, their computation with an average is performed on the device. Thus, after simulating the realizations of the actualized payoff on the device, we perform the reduction on the device using shared memory.

a) Define MC_k2 which does the same operations found in MC_k1 plus the reduction part.

b) Would it be possible to print directly the values of sum and sum2 within MC_k2?

c) Does the reduction change significantly the execution time of the kernel MC_k2 when compared to MC_k1?

1.2.2 16.2.2 Nested Monte Carlo simulation, MC_nested_k and init_curand_nested_state_k

When we have many Monte Carlo simulations to perform at the same time, it is tempting to serialize with respect to the number of Monte Carlo simulations. However, when these Monte Carlo simulations are equivalent (like with nested Monte Carlo), it is more effective to parallelize first with respect to the number Monte Carlo simulations, then parallelize as much as possible with respect to the number of trajectories. This is exactly what will be done here.

a) How the number of Monte Carlo simulations m $>$ 65536 should be indexed? with blockIdx.x? with blockIdx.y? with threadIdx.x? A combination?

b) How should we index the number of simulated trajectories? What about the global memory limit? What should we serialize?

c) The number of computed sum and sum2 is equal to m. Use the unified memory for sum and sum2.

d) Define the kernels MC_nested_k and init_curand_nested_state_k, then check the result and the execution time.

[ ]: