Introduction

The provided thesis “GPU-BASED SIMULATION OF BRAIN NEURON MODELS” takes a deep dive into the challenges posed by the implementation of a very-large-scale neuron model on the massive parallel architecture such as the CUDA GPU platform. Having explained why the SNN (Spiking Neural Networks) are recently more studied than ordinary Artificial Neural Networks, the author takes in account simulating the most complex model (the one that mimics best the biological specifications of an IO neuron): the Hodgkin-Huxley model. In the document, it is explained and highlighted that this model needs the most computational resources (in terms of FLOP/s) from the mentioned ones.

The purpose of the thesis is assessing the performance improvement of the original algorithm on different CUDA platforms, over their sequential counterpart. Furthermore, the author uses good analysis and understanding of the architectural details (of the specific CUDA generation) in order to further optimize the computation on both studied architecture generations (Fermi and Kepler).

Implementation details

-L1/L2 cache usage differences – Write-Through (problem with the scatter pattern access, due to L1). Because of the Write-Through characteristic of the L1/L2 cache system, performance can be impacted when the data has poor spatial (and temporal) locality. However, for the Fermi scientific plaftom (Tesla and GeForce) the benchmarking shows that this application does not exhibit the mentioned kind of data alignment problem. Therefore, as expected, the use of the cached version reduces the memory access times and allows for consistent speed-ups.

The Kepler implementation, using a low-budget graphics card (GT640), shows similar performance for both cached and non-cached runs, which can suggest different caching techniques than the other cards. The author suggests different utilization of the L1/L2 cache ensemble, suggesting different purposes for these memory: possibly the Write-Back mechanism is employed now and the L1 is only used for spilling registers and stack data usage.

-Texture memory usage

-Concurrent kernel execution   
-Synchronization

-USING efficient block size

In the specifications of the Tesla platform, it states that the warp size is 32 threads, therefore a block size should be a multiple of the warp size for maximum efficiency. Moreover, different block sizes show significant differences in performance because of SM resource utilization (registers per thread limited and also context switch bounds). Although it is suggested that “the higher number of threads per block, the higher the occupancy”, there is a threshold where the function shows a maximum in efficiency. It is noted that context switching or not enough number of registers per thread may impose the observed bottleneck.

The limitations of the application’s implementation on the targeted platforms

-All data computation of an IO cell (that is computed entirely in a kernel) needs to be available at the kernel launch, making the overlapping of memory transfer and processing impossible. Furthermore, making use of the fast shared memory is restricted because this type of memory is too small in comparison with the application requirements for normal network size (over 10000 cells).

-In contrast to the Tesla high-performance platform (which uses GDDR5 frame buffer memory), the GeForce GT640 makes use of DDR3 memory which significantly reduces performance as it constrains the throughput of the system (bandwidth is more than 5 times smaller than Tesla’s memory).

Code details:

-Big iterative loops