

Assignment #3: Report

Advanced Multicore Systems

Tudor Voicu – 4422066

Misael Hernandez – 4423615

May 26th, 2015

# 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).

# Question 1

The provided sequential C code implementation was benchmarked in two systems. The first one is a laptop with an Intel Core i7-4700HQ CPU running at 2.5 GHz with turbo-boost up to 3.4 GHz. The second one is the server to which we connect to, with an Intel Xeon processor. Table 1 and Figure 1 show the results for both, along with a comparison to the results in the thesis. The difference that we notice is mainly due to the clock speed of each processor.

Table 1. Execution time comparison

|  |  |  |  |
| --- | --- | --- | --- |
| Size | Laptop | Server | Thesis |
| 1 | 0.251 | 0.451 |  |
| 4 | 0.741 | 1.277 |  |
| 32 | 5.271 | 8.582 |  |
| 64 | 10.426 | 17.241 | 15 |
| 256 | 41.317 | 69.028 | 61 |
| 512 | 82.644 | 139.337 |  |
| 1024 | 164.641 | 272.317 | 242 |
| 2048 | 331.481 | 554.125 |  |
| 4096 | 657.824 | 1083.147 | 973 |

Figure 1. Execution time comparison

# Question 2

To speed up the sequential code, the thesis paper refers to using Cuda. To obtain the most speed up, the thesis writer refers to these methods:

* L1/L2 cache usage differences: 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 platform (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 memories: possibly the Write-Back mechanism is employed now and the L1 is only used for spilling registers and stack data usage.

* Texture memory usage: This application has a very specific memory access pattern. Each cell requires information only of its neighbors. Since it is a 2D array, a normal caching method by cache lines might cause a problem. The texture memory is another type of global memory that differs in that it is cached in a geographical way. This is better suited for the application. It is read-only memory for each kernel run, however, the information stored there is only valid for one kernel, so it is not a problem. Furthermore, it alleviates the border conditions, since accessing invalid addresses doesn’t cause an exception. This also helps to reduce divergent branching.
* Merging of memory transfers: In the original algorithm, the input data was sent one at a time, inside the first loop. To reduce the amount of memory transfers, this step was moved outside of the loop, so the whole input matrix is sent at the beginning, requiring no more transfers afterwards.
* Global variable: The counter for the time steps is kept in global memory, being managed by a single thread. This also reduces communication overhead.
* 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 are the following:

* 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).

# Conclusion

As mentioned in other labs, an efficient use of CUDA requires a broad knowledge on the underlying hardware. The thesis writer was able to achieve a lot of speed up due to this fact.

The most important limiting factor that he had to deal with was the memory bandwidth. Through the use of the texture memory, proper arrangement of the data and memory transfer management, he could overcome this problem.

However, the architecture of the GPU itself is also important. By knowing how the functional units in the GPU are organized, he was able to obtain an even higher speedup.