Technical report
Juan Pablo Güitrón Leal
September 2018

Summary

The following document is a report of the first multi-core programming assignment. This first assignment consists of executing a matrix multiplication in CPU, CPU with threads, and GPU. This document shows an analysis and comparison between the execution times of each of the algorithms used.

Introduction:

Parallel programming is the use of computational resources to execute a task simultaneously. Parallel programming solves problems faster than sequential programming. However, there are many problems that can not be parallelized. The objective of this first assignment was to generate a code that could be executed in parallel and analyze the difference of running time between a sequential and a parallel algorithm.

The program consists of 6 files:

- CPU.cpp: Executes sequentially the matrix multiplication in the CPU.
- CPUWT.cpp: Performs the matrix multiplication in the CPU using threads.
- CUDA_1D.cu, CUDA_2D_1D.cu, CUDA_2D_2D.cu: Performs the matrix multiplication in the GPU.
- Common.h: Used to display error messages that occurred on the GPU.

Development:

A total of 180 tests were performed in 2:46 hours.

- **Characteristics of the system used:**
    - Device: Laptop
    - Model: HP Pavilion 15-cx0001la Notebook
    - Processor (CPU): Intel Core i5-8300H
        - Frequency: 2.3 GHz
        - Cores: 4
    - Video card (GPU): NVIDIA GeForce GTX 1050
        - Memory: 4 GB GDDR5
        - Cores: 640 cuda cores
        - Frequency: 1354 MHz

- **Selection of threads and GPU dimensions:**

To select the best dimensions and number of threads in the execution of the algorithm used by CUDA, several tests were performed to evaluate which one had the least execution time.

For the selection of the dimensions, 30 tests were carried out with matrices of 1000x1000 using 256 threads, in the following table we can see that a 2D 2D dimension has a shorter execution time.

Comparison between GPU dimensions (N= 1000, threads=256)			
Type	1D	2D 1D	2D 2D
	563.142700	1,763.236938	377.552032
	536.663330	1,749.408569	379.231445
	540.141479	1,820.285767	379.001007
	537.012146	1,782.862549	378.729034
	540.972717	1,883.993286	379.069763
	537.661987	1,779.980103	379.352264
	539.681030	1,851.887207	378.978394
	537.560974	1,847.251831	378.390472
	562.651550	1,772.345703	378.920227
Time (ms)	566.770813	1,805.569946	378.453278
Average	546.225873	1,805.682190	378.767792

For the selection of the number of threads, 60 tests were performed using 1000x1000 matrices varying the size of the dimensions of X and Y. The times of each test are shown in the following table. It can be observed in this table that when the dimension of Y is greater than the dimension of X, the execution time is less.

Comparison between GPU 2D 2D threads (N= 1000)						
Threads	x=32 y=32	x=16 y=16	x=2 y=256	x=16 y=64	x=256 y=2	x=64 y=16
	449.107971	374.413086	60.451370	370.380035	1,530.697632	431.274200
	428.429810	375.472870	60.241409	370.516174	1,548.359375	431.111176
	428.049133	374.572632	60.278282	370.785980	1,493.797729	431.149261
	428.669037	374.092865	60.147335	370.588684	1,541.272461	431.112366
	428.035217	374.207031	60.265541	370.953125	1,570.500122	431.271210
Time (ms)	428.303864	375.113556	60.286343	371.301788	1,505.011963	431.273926

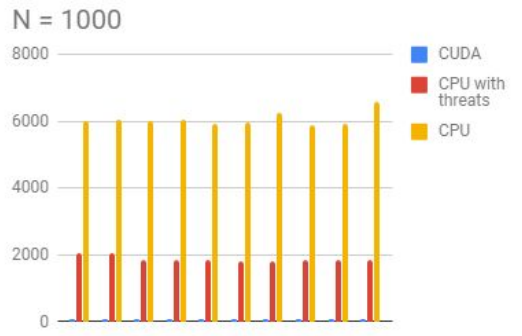
	428.462311	373.274292	60.123272	374.296417	1,527.695923	431.376923
	428.476654	374.722504	60.192356	374.365723	1,524.801147	431.165405
	428.179718	373.945831	61.459854	374.805969	1,534.717529	430.924774
	428.565033	372.954559	60.171268	374.457001	1,546.117554	431.379456
Average	430.427875	374.276923	60.361703	372.245090	1,532.297144	431.203870

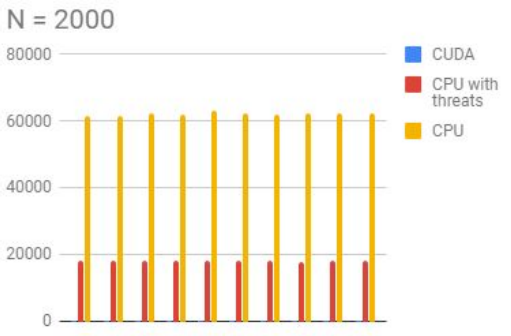
This configuration of dimensions and threads used is mainly due to two reasons:

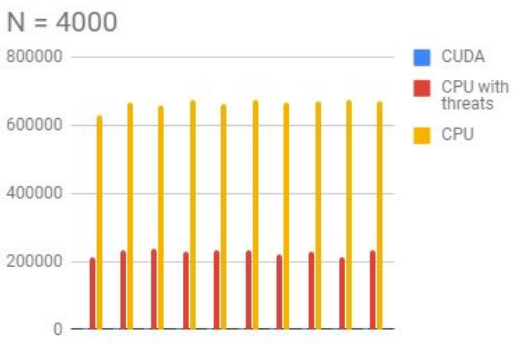
- Due to the nature of the problem, it is easier to go through the matrices using more dimensions, since the need for another cycle within each operation is avoided.
- Because of the way in which the problem is programmed, the execution of the program becomes slower if X is greater. This is due to the fact that when the result of the multiplication is stored in the result matrix a multiplication is made using X, so if multiplication is greater, the execution time increases.
-

Comparison of times (Times in milliseconds):

The following graphs show a comparison of execution times between the CPU and GPU (CUDA), changing the size of the multiplied matrices.







Comparison			
	CUDA	CPU with threads	CPU
1000	66.856436	1,883.551355	6,066.489941
2000	460.824149	18,222.521680	62,244.603125
4000	3,660.053858	227,893.415625	666,306.481250

- **Performance improvement factor**

  The following table shows the performance improvement factor (speed-up) which is defined as S (n) = T (1) / T (n).

Speedup		
	CUDA	CPU with threads
1000	90.739057	3.220772
2000	135.072355	3.415806
4000	182.048272	2.923764

Conclusions:

The results obtained previously demonstrate the capacity of parallel programming. Currently, parallel programming is something that is is being increasingly used. In this assignment, the GPU proved to be the best for parallel programming. However, having a dedicated GPU in a computer can be very expensive; so most of the current computers only use parallel programming in the CPU which has a lower number of threads.

In parallel programming it is important to know how to select the number of threads that will be used, because with a bad selection the performance could be affected in a negative way.

Referencias:

- Arnau, V. (2012). Introducción al paralelismo y al rendimiento. Septiembre 1, 2018, de Universidad de Valencia Sitio web: https://www.uv.es/varnau/OC_T4.pdf

- Blaise, B. (2018). Introduction to Parallel Computing. Septiembre 1, 2018, de Lawrence Livermore National Laboratory Sitio web: https://computing.llnl.gov/tutorials/parallel_comp/

- Notebook HP Pavilion 15-cx0001la. Septiembre 1, 2018, de HP Sitio web: https://www.hponline.com.mx/p/notebook-hp-pavilion-15-cx0001la-vesz54?gclid=EAIaIQobChMI-s-yusWb3QIVFLbACh2t0wIyEAQYAiABEgLqlfD_BwE