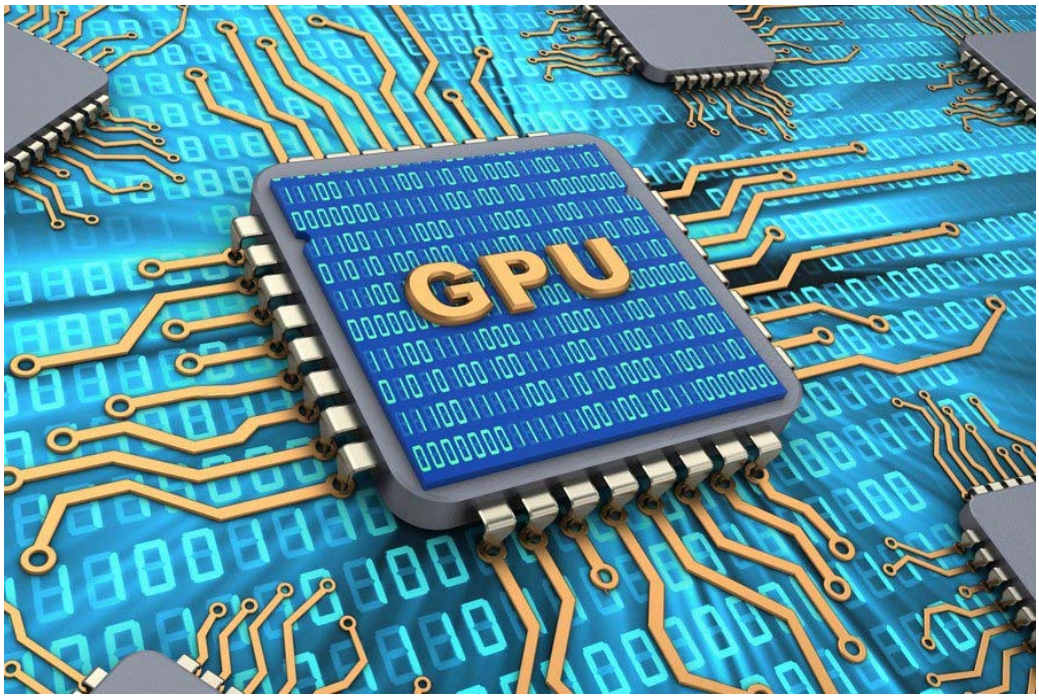

CAOS RESEARCH REPORT

By Jeremy Johnathan Williams

UNIVERSITAT AUTÒNOMA DE BARCELONA

CAOS Research Report

A Research Report in the Internship Fellowship Programme in the
Department of Computer Architecture and Operating Systems (CAOS)

By

Jeremy Johnathan Williams

Master's Degree in Modelling for Science and Engineering, July 4th, 2019

CAOS RESEARCH PLAN

NAME OF THE PERSON RESPONSIBLE FOR THE AGREEMENT : **Miquel Angel Senar**	
POSITION OF THE PERSON RESPONSIBLE FOR THE AGREEMENT: Department Director	
COMPANY'S NAME: Department of Computer Architecture and Operating Systems, UAB	
CIF :	
ADDRESS / STREET AND NUMBER : C/ Sitges S/N	
TOWN : Cerdanyola del Vallés	POSTAL CODE: 08193
TELEPHONE:	FAX:
E-MAIL ADDRESS:	
TUTOR APPOINTED BY THE COMPANY: **Juan Carlos Moure López**	
DATES FOR IN-COMPANY PRACTICUMS PERFORMANCE: October 2018 until June 2019	
DAYS OF WEEK: A) October 2018 to December 2018 (3 Months) -Monday, Wednesday, Thursday and Friday B) January 2019 to June 2019 (6 Months) - Monday, Tuesday, Wednesday, Thursday and Friday	TIMETABLE: A) 16:00 to 20:00 (4hrs) B) 16:00 to 20:00 (4hrs)

WORKING PLAN:

We want to accelerate a typical data analytics process using GPUs. The central algorithm of the analysis process is the k-means clustering algorithm, which is a classical algorithm for unsupervised machine learning.

The first goal is to design and evaluate a GPU-accelerated implementation of the k-means clustering algorithm. The input data will be generated randomly and will fit into the GPU memory. A performance engineering methodology will be used to develop an optimized version.

The second goal is to execute and optimize a data analytics problem on a GPU-accelerated computer system, using real data. The performance of the analysis task on the GPU-accelerated system will be compared with the performance on the original baseline system.

Finally, we will study the scalability of the algorithm in order to use GPUs to solve the problem, either attached to the same node or to separate nodes, and in order to combine computation between GPUs and CPUs in a heterogeneous system.

CAOS HISTORY AND LEADERSHIP

The CAOS was created by agreement of the Governing Council of the UAB on January 27, 2005. Their roots come from 1986 as a unit of teaching and research within the old CAOS.

The Department teaches at the School of Engineering (in the Bellaterra headquarters and Sabadell) and in other centers such as the Faculty of Sciences of the Universitat Autònoma de Barcelona (UAB). The academic activity is basically focused on the Degree in Computer Engineering, the Master's Degree in Computer Engineering and Doctorate in Computing in the line of High Performance Computing.

The Department has two research groups that focuses on High Performance Computing on platforms such as Clusters and Cloud for both multicore and GPU architectures.

At present, the Head of the Department is Dr. Miquel Angel Senar Rossell and between academic staff, administration and services and research fellows; consisting of almost 40 members.

Leadership of the department are as followed:

- Director: **Miquel Àngel Senar Rosell**
- Secretary: **Eduardo César Galobardes**
- Coordinator Doctorate / Master's Degree: **Anna B. Sikora**
- Coordinator Section: **Juan Carlos Moure López**
- Coordinator for Academic Affairs and Quality (School of Engineering): **Remo Suppi**

ACKNOWLEDGEMENTS

I would like to express my gratitude to my research paper advisor, Dr. Juan Carlos Moure Lopez for the useful comments, remarks and engagement through the learning process of this master thesis. I would like to thank the director and the selection committee of Department of Computer Architecture and Operating Systems (CAOS) organized by the School of Engineering, UAB for allowing me the opportunity to develop my research training in the area of Architecture and Computer Technology, in particular, Structure of Computers, Computer Architecture, Operating Systems, Distributed, Parallel and High Performance Processing and its applications leaded by my research paper advisor, Dr. Juan Carlos Moure Lopez. Furthermore, I would like to thank all of the MMSE Professors, Committee Members and Coordinator, Professor Anna Cima, for helping me with the selection of my training, educational development, progression of my topic, research experience and thesis structure in the Department of Computer Architecture and Operating Systems during October 2018 until June 2019. Also, I would like to thank my loved ones, who have supported me throughout entire process, both by keeping me harmonious and helping me put the pieces of this master thesis together. I will be forever grateful to you all.

CONTENTS

1. INTRODUCTION

SINCE the beginning of time, business and technology have been disengaged as two completely different regions of capable undertakings and futuristic developments. These anomalies inspired essential improvements in industry, research and extraordinary findings over the last few centuries; towards what we know today as the 21st century.

For nine (9) month from October 2018 until June 2019, I did a research internship at CAOS in UAB. I was really very privileged to be a part of the great team; where I directly worked with the coordinator section, Dr. Juan Carlos Moure López.

During my stay at CAOS, I have learnt how to use a GPU (graphics processing unit) to accelerate an application running on the CPU (central processing unit) by offloading some of the compute-intensive and time consuming portions of the code. In this case, the GPU will become a co-processor during the execution of the accelerated application.

To me, an accelerated application runs faster than any application executing with the CPUs only. This is because it uses the GPU´s massively parallel processing power to boost performance.

Thanks to the wonderful, Dr. Juan Carlos Moure López, who took his time to teach me everything about Architecture, Computer Technology and GPU Computing. In which, I was able to develop my first ever GPU-accelerated implementation with an official real dataset.

However, I have learned that to have extremely good performance and higher memory usage, with the help of the GPU as a co-processor, it is recommended to have a good balance between the selection variables (in any application default settings) and a very large dataset (i.e. BIG DATA).

The research conducted, during my stay, was an experimental study with a motivation to successfully design, optimize and evaluate a GPU-accelerated implementation of the k-means clustering algorithm. The outline of our research approach with the understanding of the application designs and the data visualizations of the GPU-accelerated application performance will now be deliberated.

2. APPLICATION DESCRIPTION AND METHODS

The k-means clustering algorithm has been described as one of the unsupervised learning - data mining alogrithms. The objective of this algorithm is to find collections (centroids) in the data, with the number of sets denoted by the variable “k”.

There are **three (3)** application design approaches that was used to develop a GPU-accelerated implementation of the k-means clustering algorithm. They are as followed:

1) **CPU** – Baseline Version
   i. Random Dataset
   ii. Input Dataset
2) **CPU** – Optimized Version
3) **GPU** – Accelerated Verson

2.1 CPU – Baseline Versions

We begin with two (2) types of baseline versions, **Random** and **Input Dataset**.

2.1.1 Random Dataset

Firstly, the description and structure of our initial version defined as **“kmeans_baseline.c”.**
The source code **“kmeans_baseline.c”** includes 5 functions including main() function and random data.
Screenshot of running this program is as followed:

```
K-means:
        Points: 1000
        Dimension: 3
        Centroides: 4
Points space is from 1000 x 3

Generated 1000 x 3 random data.

Centroides:
       777.0   474.0   61.0
       698.0   499.0   59.0
       225.0   11.0    899.0
       556.0   709.0   62.0

Number of iterations needed: 10000

New centroides:
       783.6   364.8   663.4
       331.6   299.5   241.5
       258.6   592.6   750.0
       624.5   785.2   311.0
```

Figure 1: Sample execution of the program “**kmeans_baseline.c**”

2.1.2 Input Dataset

This program is a restructured version of **"kmeans_baseline.c"** to develop **"kmeans_baseline_input.c"**. The difference of them are reading the selected input dataset from **csv** (comma-separate value) file instead of randomly generating a dataset, as required.

Below is a screenshot of running this input program (with a sample real data set) is as followed:

```
Usage: ./KMI1_CPU1<input_data.csv>, <K=Number of Clusters>, <N=Number of Rows>, <D=Number of Columns>
We now use default parameters with InputData.csv, K=3, N=3010, D=8
K-means:
        Points: 3010
        Dimension: 8
        Centroides: 3
reading input data from InputData.csv

Read 3010 x 8 points data from InputData.csv.

Centroides:
       43.0    50.8    3.9     2.3     48.7    18.1    25.8    2.7
       9.4     41.9    27.9    18.8    31.6    15.0    4.6     2.3
       18.4    52.7    19.4    4.7     28.9    26.6    18.5    1.1

Number of iterations needed: 10000

New centroides:
       48.5    25.4    15.3    7.8     36.0    21.2    23.8    4.5
       22.0    35.8    24.5    13.4    36.5    17.9    14.4    8.2
       38.3    31.2    18.4    9.7     33.5    51.1    25.9    6.2
```

Figure 2: Sample execution of the program "**kmeans_baseline_input.c**"

2.2 CPU – Optimized Version

At this point, the program has fulfilled execution requirements needed for the problem at hand. This program is a restructured version of **"kmeans_baseline_input.c"** that was developed as an optimized version called **"kmeansFINAL.c"**. The difference between them is that the program transformation technique called "code optimization" was be used to improve the intermediate "**baseline input version"** by developing it to consume fewer resources (i.e. CPU, Memory); so that it improves the speed and performance of the program.

The optimized program improvements were designed in three (3) parts. They are as followed:

1) **Input** (to generates N points with D - dimensions randomly and makes the choice of each centroid),
2) **Computation** (to update the center for all the centroids, returns the position (centroid number) where the minimum value of a vector is found and assign each point to the nearest centroid)
3) **Output/Main** (to preform the output results via the kmeans algorithim).

2.3 GPU – Accelerated Version

At this point in the research, the program has fulfilled all the optimized and relevant parallelism requirements needed for data transfer to the GPU. This program is a restructured version of **"kmeansFINAL.c"** that was altered as a GPU-accelerated implementation called **"kmeansACC.c"**. The difference of them is the insertion of the OpenACC Directives in selected parts of the program. OpenACC works with three levels of parallelism via gang, worker and vector. When executing a selected computational region on the GPU, one or more gangs are launched, each with one or more workers, where each worker uses the SIMD (Single instruction, multiple data) vector execution ability with one or more vector line of data flow. Therefore, we developed the relevant parts of the program that completes the data pipeline, performance optimization techniques and simplify parallel programming process (i.e. OpenACC Directives) of heterogeneous CPU/GPU system.

We will now explain the GPU-accelerated implementation, the OpenACC Directives that were inserted in the most computationally expensive parts of **"kmeansFINAL.c"** for high-level data management and optimizing loops in **"kmeansACC.c"**.

Firstly, the "**kmeansFinal.c**" code was modified to include OpenACC directives. Analytically, the **"#pragma acc data copyin(Punts[N*D]) copyout(Labels[N]) copy(Centroides[K*D])**" and **"#pragma acc data create(New_Centr[K*D]) copyout(Sep[K])**" were inserted inside the "main" function before the "do loop", in order to allocate memory (where necessary), move the contents of reassignment each points to nearest centroids from host to GPU and copies data to the host when exiting to the function region.

Furthermore, the "**#pragma acc data present(Punts,Labels,Centroides,New_Centr,Sep) copy(pR[1])"** was inserted inside the "**PointsToCentroides**" function before the first loop to inform the compiler that data is already present on the device with allocated memory on GPU, copy data from host to GPU (device) and copy data to the host when exiting to the function region. The "**#pragma acc parallel loop gang vector_length(128)"** was put in front of the outmost loop and two individual internal outmost loops. These parallel regions will utilize thread blocks, each with 128 threads, where each thread executes one iteration of the loops.

Also, the **"#pragma acc loop vector**" was place before the innermost for loops of each individual **"#pragma acc parallel loop gang vector_length(128)**". The use of **acc parallel** provides massive parallelization and identifies each loop to run in parallel. This directive basically tells the compiler to analyze all the code that is assigned to. As a result, the compiler "understands" the code in the annotated accelerated region and does all the parallelizing of selected parts of the code provided by us.

These change prepares the GPU-accelerated program "**kmeansACC.c**" for our official input data.

3. DATA MODIFICATION AND OVERVIEW

3.1 Data Description

As discussed earlier, input data will be an official real dataset that was extracted using a Data Extraction System and will fit on the GPU memory.

The USCensus1990raw data set was collected as part of the 1990 census from the (U.S. Department of Commerce) Census Bureau website at http://www.census.gov.

Data Set Characteristics:	Multivariate	**Number of Instances:**	2458285	**Area:**	Social
Attribute Characteristics:	Categorical	**Number of Attributes:**	68	**Date Donated**	N/A
Associated Tasks:	Clustering	**Missing Values?**	N/A	**Number of Web Hits:**	134120

Table 1: US Census Data (1990) Data Set Description, Retrieved Web. 19 Apr. 2019. **[29, 30]**

The data set was extracted using the Data Extraction System found at http://dataferrett.census.gov.

The data set is a 1% sample of the Public Use Microdata Samples (PUMS) person records drawn from the full 1990 census sample.

Table 1 shows that the data set contains 68 categorical attributes with a sequence of operations via Randomization and Selection of Attributes. It also contains a total of 2,458,285 number of instances or data points.

3.2 Data Layout

Below shows the original dataset layout:

caseid,dAge,dAncstry1,dAncstry2,iAvail,iCitizen,iClass,dDepart,iDisabl1,iDisabl2,……

10000,5,0,1,0,0,5,3,2,2,1,0,1,0,4,3,0,2,0,0,1,0,0,0,0,10,0,1,0,1,0,1,4,2,2,3,0,2……

10001,6,1,1,0,0,7,5,2,2,0,0,3,0,1,1,0,1,0,0,0,0,1,0,0,4,0,2,0,0,0,1,4,1,2,2,0,2……..

10002,3,1,2,0,0,7,4,2,2,0,0,1,0,4,4,0,1,0,1,0,0,0,0,0,1,0,2,0,4,0,10,4,1,2,4,0,2……

10003,4,1,2,0,0,1,3,2,2,0,0,3,0,3,3,0,1,0,0,0,0,0,0,1,4,0,2,0,2,0,1,4,1,2,2,0,2,0…...

10004,7,1,1,0,0,0,0,2,2,0,0,3,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,2,2,0,0,0,4,1,2,0,0,2,0…...

……..

3.3 Data Input and Readability

To conduct our experimental study, the dataset was reduced by 40.68%, to exactly 1 million data points for advanced analytics insight and industry standard readability via .CSV; which can be regenerated to open in Microsoft Excel from the Microsoft Office suite of software.

3.4 Data Density Heatmap

After selecting the required dataset, we conducted a “scaling of data” due to the source dataset spanning in different ranges within an unstructured manner. A graphical representation of dataset representing the density of dots in a map was demonstrated.

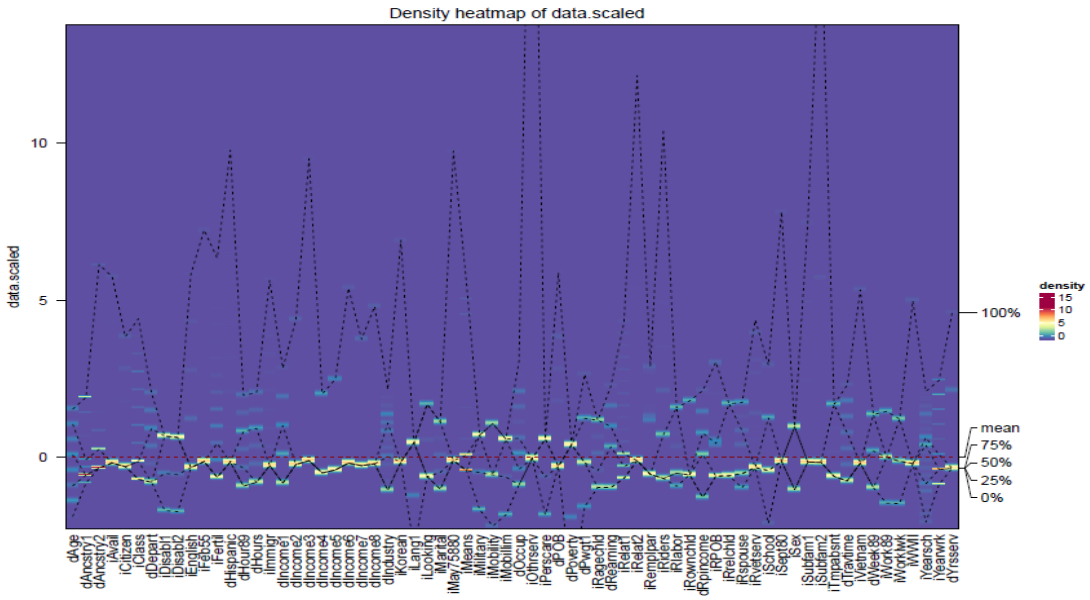

Figure 3: Density Heatmap of the US Census Data (1990) Data Set Normalized

Figure 3 shows the perceive density of points using a scale function to normalize the data points.

4. COMPUTER ARCHITECTURE AND RESOURCE MANAGEMENT

All executions were conducted in a Computer Architecture Lab (via heterogeneous cluster for executing batch jobs) and Queue System (via execution queues (or partitions) configured using SLURM - Simple Linux Utility for Resource Management) as a jobs manager.

Let's now define the accessible queues and nodes used:

test.q: contains only the aolin-login node

- This node provides up to 4 threads (4 cores, each core with 1 H/W threads) and no GPU. This partition was used for testing of all submission scripts.

cuda.q: contains nodes aolin[11-15], aolin17, aolin19, aolin21, aolin23, aolin24, and aomaster

- These nodes contain GPUs and some nodes contain the higher number of CPU cores.

The characteristics of the computation queues nodes in the cluster used:

aolin24:

- Processor: Intel Core i7-3770 @ 3.40GHz (4 cores, 2 threads x core: total of 8 H/W threads)
- L3 cache: 8MB; Memory: 8 GB at 1067 MHz
- NVIDIA GeForceGTX 1080 Ti (3584 cores)

The submission of the parallel GPU jobs will be executed on the ***aolin24*** node, with the "***NVIDIA GeForceGTX 1080 Ti***" GPU device, which is autonomous of the aolin-login node via SLURM.

Also, for all parallel GPU jobs, the submission must be sent to the ***cuda.q*** queue with a special option to specify the number of GPUs reserved for the execution job via ***Gres=gpu:name_GPU:number*** or ***Gres=gpu:number*** (i.e. not select a particular GPU device).

In this case, the following commands had to be typed before each session:

```
#!/bin/bash -l
#
#SBATCH --job-name=ML_KMs
#SBATCH -N 1 # number of nodes
#SBATCH -n 2 # number of cores TOTALES
#SBATCH --partition=cuda.q
#SBATCH --nodelist=aolin24
#SBATCH --gres=gpu:1
```

GNU Compiler toolset was installed: ***module add gcc/8.2.0***
The PGI toolset was installed: ***module load pgi/18.4***
The CUDA toolset was installed: ***module load pgi/18.4***
Show info about available GPUs: ***nvidia-smi***

5. APPLICATION PERFORMANCE ANALYSIS AND RESULTS

In this research, we want to measure the application performance and study the results of the GPU accelerated implementations (via OpenACC directives). We will conduct two (2) distinctive experiments,

1) "K" Clusters, ("K" number of groups with all other variables remain constant),
2) "D" Multidimensions ("D" dimention of points with all other variables remain constant),

5.1 Compiling, Building and Running the Application

All codes, for the GPU-accelerated implementations (via OpenACC directives) were compiled with the command: ***pgcc -fast -acc -ta=tesla -Minfo=all kmeansACC.c -o codename.*** Each execution had ***200*** iterations with the use of ***perf stat*** and ***pgprof/ nvprof*** in order to measure basic performance metrics. Every code is resulted from the previous one, with some alterations, and it has been checked that provides correct results.

5.2 Application Execution

For advanced analytics insight and final results industry standard readability, we now display how the GPU-accelerated application executes, using the following input variables:

N: number of points = **1,000,000** (1 million)

D: dimension of point = **68**

K: number of groups (k clusters) = **2**

As discussed before, we present the usual output .CSV file of the k-means clustering problem from our sample of the USCensus1990raw data set that will fit on the GPU memory:

Number of clusters: **2**
Cluster-**1**: **750007** elements
Cluster-**2**: **249993** elements

Cluster - **1**

caseid,dAge,dAncstry1,dAncstry2,iAvail,iCitizen,iClass,dDepart,iDisabl1,iDisabl2,……

10000,5.0,0.0,1.0,0.0,0.0,5.0,3.0,2.0,2.0,1.0,0.0,1.0,0.0,4.0,3.0,0.0,0.0,2.0,0.0,0.0……

10001,6.0,1.0,1.0,0.0,0.0,7.0,5.0,2.0,2.0,0.0,0.0,3.0,0.0,1.0,1.0,0.0,0.0,1.0,0.0,0.0……

10002,3.0,1.0,2.0,0.0,0.0,7.0,4.0,2.0,2.0,0.0,0.0,1.0,0.0,4.0,4.0,0.0,0.0,1.0,0.0,1.0……

10003,4.0,1.0,2.0,0.0,0.0,1.0,3.0,2.0,2.0,0.0,0.0,3.0,0.0,3.0,3.0,0.0,0.0,1.0,0.0,0.0……

10004,7.0,1.0,1.0,0.0,0.0,0.0,0.0,2.0,2.0,0.0,0.0,3.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0……

10007,4.0,1.0,2.0,0.0,0.0,6.0,0.0,2.0,2.0,0.0,0.0,4.0,0.0,5.0,5.0,0.0,2.0,1.0,0.0……

……

Cluster - **2**

caseid,dAge,dAncstry1,dAncstry2,iAvail,iCitizen,iClass,dDepart,iDisabl1,iDisabl2,……

10005,1.0,1.0,2.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0……

10006,1.0,1.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0……

10012,1.0,1.0,2.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0……

10020,1.0,2.0,2.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0……

10024,1.0,1.0,2.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0……

10026,1.0,3.0,2.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,1.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0……

……

The most relevant performance metrics (via **perf stat**) for the "**kmeansACC.c**" code are the elapsed execution time (10.09 seconds time elapsed), the total number of executed machine instructions (57.104 Giga), the IPC rate (1.55 insns per cycle) and total utilization rate (0.969 CPUs utilized).

5.3 GPU Activity Display

Continuing with the k-means clustering problem from our sample of the USCensus1990raw data set, where N = 1M, D = 68, K = 2 and I = 200, the command "**pgprof**" was used to measure the execution time of each kernel, displaying the GPU activity as following:

Time (%)	Time	Calls	Avg	Min	Max	Name
99.12%	3.62860s	200	18.143ms	17.782ms	22.388ms	PointsToCentroides_121_gpu
0.85%	31.028ms	18	1.7238ms	896ns	1.9869ms	[CUDA memcpy HtoD]
0.01%	471.25us	203	2.3210us	736ns	315.05us	[CUDA memcpy DtoH]
0.01%	426.10us	200	2.1300us	1.9520us	3.2000us	PointsToCentroides_215_gpu
0.01%	225.57us	200	1.1270us	1.0880us	2.0810us	PointsToCentroides_111_gpu
0.00%	112.33us	200	561ns	513ns	832ns	[CUDA memset]

Table 2: **pgprof** results of the k-means clustering problem where N = 1M, D = 68, K = 2 and I = 200

Table 2 shows the kernel assignment each point to the nearest centroid and updating the center for all the centroids takes 99.12% of the execution time on GPU and the kernel reassignment each points to nearest centroids memory allocation and copying (in/out) of Punts, Centroides and Labels takes 0.01% with the kernel final memory allocation, data creation of New_Centr and copies data out of Sep to the host takes only 0.01% of the execution time. Total execution time of GPU device was 4.03 seconds.

5.4 Application Scalability

The scalability of the algorithm using combine computation between CPU and GPU in a heterogeneous system with performance analysis process of multiple 'K' clusters and

'D' multidimensional experimentations will now be presented. Tableau Software will be used for data visualization results of the GPU-accelerated computer system.

5.5 'K' Cluster Experimentations

5.5.1 GPU Data Visualizations

– Execution Time (seconds)

Figure 4 shows the data visualization of execution time(s) of selected "K" Clusters.

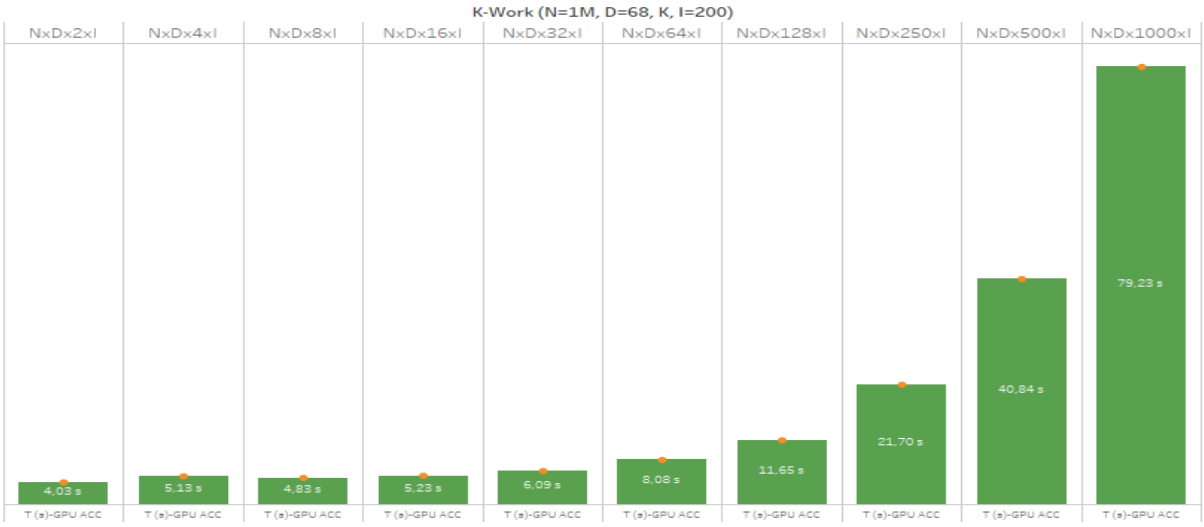

Figure 4: GPU execution time(s) of selected 'K' clusters (with N = 1M, D = 68, K and I = 200)

– Performance (Work/Time)

Figure 5 shows the data visualization of performance of selected "K" Clusters:

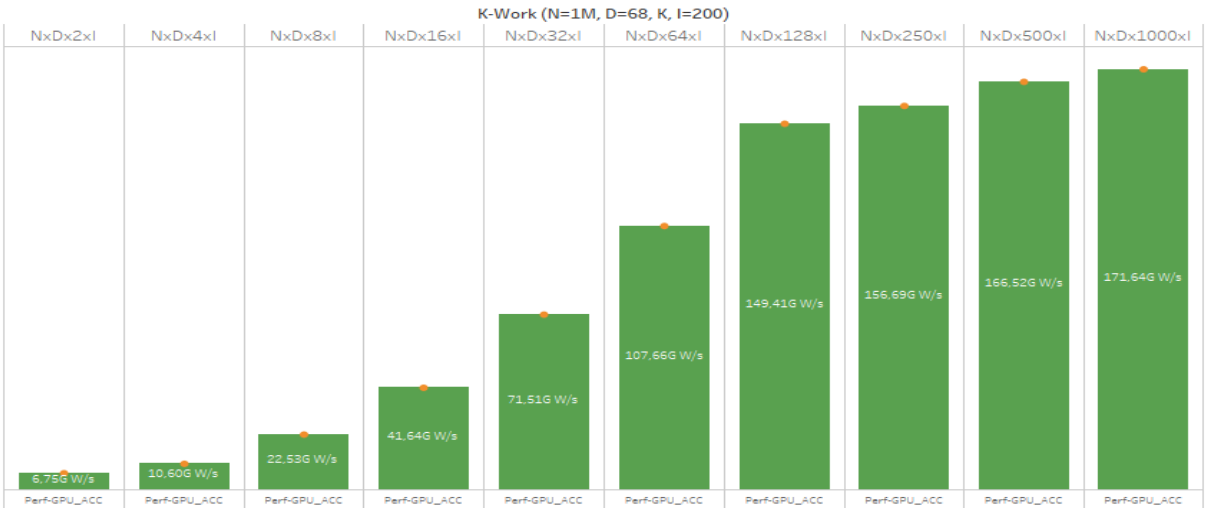

Figure 5: GPU performance of selected 'K' clusters (with N = 1M, D = 68, K and I = 200)

5.6 "D" Multidimensional Experimentations

5.6.1 GPU Data Visualizations

– Execution Time (seconds)

Figure 6 shows the data visualization of execution time(s) of selected 'D' multidimensional.

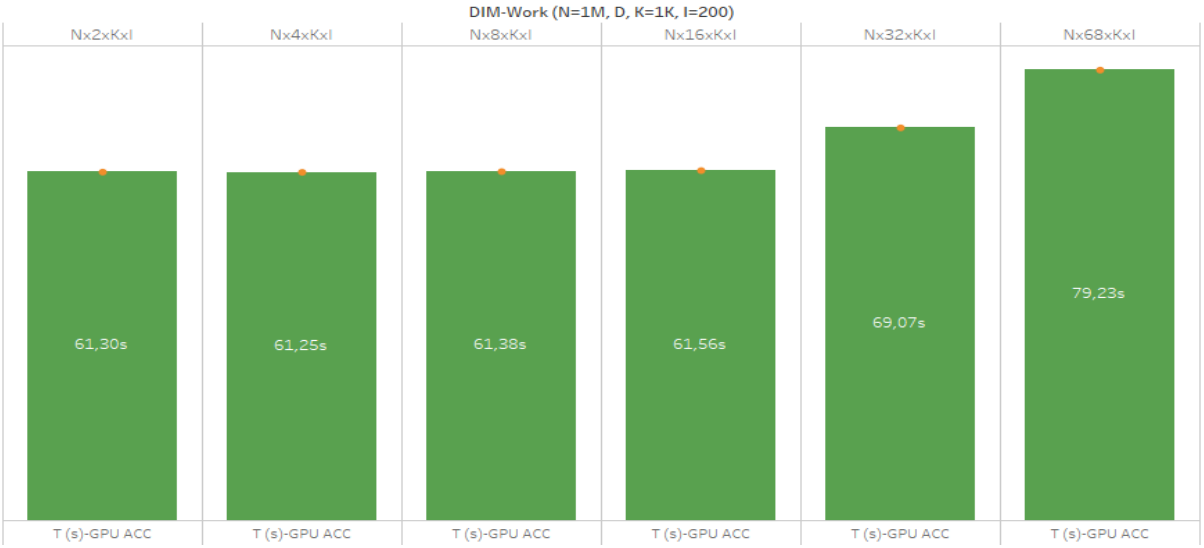

Figure 6: GPU execution time(s) of selected 'D' multidimensional (with N = 1M, D, K = 1000 and I = 200)

– Performance (Work/Time)

Figure 7 shows the data visualization of performance of selected 'D' multidimensional:

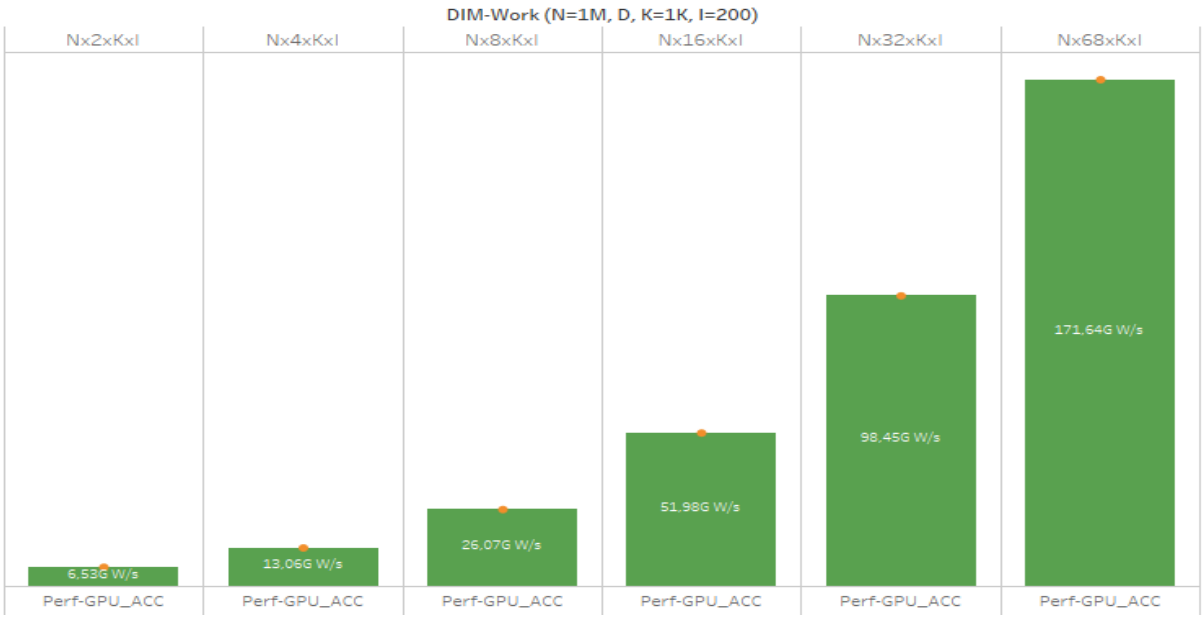

Figure 7: GPU performance of selected 'D' multidimensional (with N = 1M, D, K = 1000 and I = 200)

6. CONCLUSION

We have investigated and developed the principles of GPU Computing within the scientific arena. These principles led to approaches that solve the assumptions of our GPU-accelerated implementation.

The critical step in this research, was to develop an application(s), select a computationally expensive algorithm applied to a dataset(s), utilize standard(s) for parallel computing, simplify the parallel computing process and investigate different approaches of a heterogeneous system.

Nevertheless, we have used a different type of programming standard for parallel computing that helped us to provide a fast application performance and significantly reduced our programming efforts. We have seen this process in our three (3) application design approaches, particularly in our GPU – Accelerated Version, where we inserted a simple set of OpenACC compiler directives that helped to generate performance improvements and accelerate computation in our application.

From our GPU-implementation, the application execution provided results based on our selection variable criteria that focused on an investigation to the outcome of an unsupervised learning - data mining algorithm process. It was discovered that due to the size of the dataset, selection variable criteria and small workload size, instruction and memory latency were indicated from the GPU device; with instruction execution stalling excessively. A significant reason for instruction and memory latency issues were the size of the input data and/or problems investigated.

To utilize the parallel computing process complexity of a GPU device with better performance and higher memory usage, it is recommended to have a good balance between the selection variable criteria (in the program default settings) and a very large dataset (i.e. BIG DATA).

BIBLIOGRAPHY

[1] Janki Bhimani, Miriam Leeser and Ningfang Mi, "Accelerating K- Means clustering with parallel implementations and GPU computing", Proceeding of the High Performance Extreme Computing Conference (HPEC) IEEE, pp. 1-6, 2015.

[2] Mahmoud Al-Ayyoub, Qussai Yaseen, Moahmmed A. Shehab, Yaser Jararweh, Firas Albalas, Elhadj Benkhelifa, "Exploiting GPUs to accelerate clustering algorithms", Computer Systems and Applications (AICCSA) 2016 IEEE/ACS 13th International Conference of, pp. 1-6, 2016.

[3] M. Zechner, M. Granitzer. "Accelerating k-means on the graphics processor via CUDA." Proc. IEEE INTENSIVE, pp. 7-15, 2009.

[4] R. Wu, B. Zhang, M. Hsu, "Clustering billions of data points using GPUs", Proceedings of the combined workshops on UnConventional high performance computing workshop plus memory access workshop, pp. 1-6, 2009.

[5] NVIDIA Corporation, High Performance Computing, https://developer.nvidia.com/computeworks

[6] Wikipedia contributors. "Machine learning." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 29 May. 2019. Retrieved Web. 2 Jun. 2019.

[7] Crew, Henry, "The Principles of Mechanics". BiblioBazaar, LLC. p. 43. ISBN 978-0-559-36871-4. (2008).

[8] Sebastian Schaetz, Dirk Voit, Jens Frahm, and Martin Uecker, "Accelerated Computing in Magnetic Resonance Imaging: Real-Time Imaging Using Nonlinear Inverse Reconstruction," Computational and Mathematical Methods in Medicine, vol. 2017, Article ID 3527269, 11 pages, 2017. https://doi.org/10.1155/2017/3527269.

[9] Jiawei Han, Micheline Kamber, Data Mining: Concepts and Techniques, London: Academic Press, 5, 2001.

[10] Han, Jiawei, and Micheline Kamber. Data Mining: Concepts and Techniques. San Francisco: Morgan Kaufmann Publishers, 2001. Print.

[11] Fayyad, Usama, Gregory Piatetsky-Shapiro, Padhraic Smyth, From Data Mining to Knowledge Discovery in Databases, 1996.

[12] Nelofar Rehman, "Data Mining Techniques Methods Algorithms and Tools", International Journal of Computer Science and Mobile Computing, Vol.6 Issue.7, July- 2017, pg. 227-231

[13] F.Robert A. Hopgood, Roger J. Hubbold, David A. Duce, eds. Advances in Computer Graphics II. Springer. p. 169. ISBN 9783540169109. "Perhaps the best known one is the NEC 7220", 1986.

[14] "NVIDIA Launches the World's First Graphics Processing Unit: GeForce 256". Nvidia. 31 August 1999. Archived from the original on 12 April 2016. Retrieved 28 March 2016, from https://www.nvidia.com/object/IO_20020111_5424.html

[15] Wikipedia contributors. "Graphics processing unit." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 2 Jun. 2019. Retrieved Web. 2 Jun. 2019.

[16] Benson Tao, Vivante Corp. "Understand the mobile graphics processing unit". OpenSystems Media**.** Archived on September 16, 2014. Retrieved 16:11, March 27, 2019 from http://www.embedded-computing.com/embedded-computing-design/understand-the-mobile-graphics-processing-unit

[17] Ma, Zhihua & Wang, Hong & Pu, S.H., "GPU computing of compressible flow problems by a meshless method with space-filling curves". Journal of Computational Physics. 263. 113-135. 10.1016/j.jcp.2014.01.023. 2014.

[18] Jewell D. Performance Engineering and Management Method — A Holistic Approach to Performance Engineering. In: Liu Z., Xia C.H. (eds) Performance Modeling and Engineering. Springer, Boston, MA. 2008.

[19] C.U. Smith and L.G. Williams. Software Performance Engineering for Object-Oriented Systems: A Use Case Approach. Technical Report, Performance Engineering Services, 1998.

[20] Han, Jiawei, and Micheline Kamber. Data Mining: Concepts and Techniques, Third Edition. 3rd ed. Waltham, Mass.: Morgan Kaufmann Publishers, 2012.

[21] Singh, Archana K., Avantika Yadav and Ajay Rana. "K-means with Three different Distance Metrics." 2013.

[22] Komal D. Nistane1 and Shailendra W. Shende. GPU Accelerated Clustering Techniques. International Journal of Science and Research. Volume 4 Issue 4, April 2015.

[23] Tutorials Point. Performance Testing.
https://www.tutorialspoint.com/software_testing_dictionary/performance_testing.htm

[24] Wikipedia contributors. "Performance engineering." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 17 Mar. 2019. Retrieved Web. 6 Apr. 2019.

[25] cplusplus.com. Reference. http://www.cplusplus.com/reference/

[26] Wikipedia contributors. "Optimizing compiler." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 19 Mar. 2019. Retrieved Web. 19 Apr. 2019.

[27] Wikipedia contributors. "OpenACC." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 7 Jan. 2019. Retrieved Web. 19 Apr. 2019.

[28] OpenACC Directives, "OpenACC: More Science Less Programming." https://developer.nvidia.com/openacc

[29] Dua, D. and Graff, C. "UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]". Irvine, CA: University of California, School of Information and Computer Science. 2019.

[30] UCI Machine Learning Repository. US Census Data (1990) Data Set.
https://archive.ics.uci.edu/ml/datasets/US+Census+Data+(1990)

[31] Wikipedia contributors. "Slurm Workload Manager". Wikipedia, The Free Encyclopedia. March 5, 2019, 00:53 UTC. Retrieved April 19, 2019.

[32] SchedMD®. "Frequently Asked Questions". Slurm Workload Manager. https://slurm.schedmd.com/faq.html

[33] DataFlair ©. "Tableau Tutorial For Beginners". https://data-flair.training/blogs/tableau-tutorial/

[34] Wikipedia contributors. "Tableau Software". Wikipedia, The Free Encyclopedia. March 7, 2019, 08:20 UTC. Retrieved April 21, 2019.

[35] Wikipedia contributors. "Speedup." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 11 Apr. 2019. Retrieved Web. 22 Apr. 2019.

[36] Wikipedia contributors. "Heat map." Wikipedia, The Free Encyclopedia. Wikipedia, The Free Encyclopedia, 26 Apr. 2019. Retrieved Web. 27 Apr. 2019.