DEVELOPING PARALLEL COMPUTING ALGORITHMS USING GPU'S TO DETERMINE OIL AND GAS RESERVES PRESENTED IN THE UPSTREAM (EXPLORATION) SECTOR

Stefan T. Boodoo[1*] and Ajay Joshi[2*]

[1,2]Faculty of Engineering, The University of the West Indies, Trinidad
[1]Email: stefan.boodoo1@my.uwi.edu *(Corresponding author)
[2]Email: ajay.joshi@sta.uwi.edu

Abstract*:* Oil and Gas companies keep exploring every new possible method to increase the likelihood of finding a commercial hydrocarbon bearing prospect. Well logging generates gigabytes of data from various probes and sensors. After processing, a prospective reservoir will indicate areas of oil, gas, water and reservoir rock. Incorporating High Performance Computing (HPC) methodologies will allow for thousands of potential wells to be indicative of its hydrocarbon bearing potential. This study will present the use of the Graphics Processing Unit (GPU) computing as another method of analyzing probable reserves. Raw well log data from the Kansas Geological Society (1999-2018) forms the basis of the data analysis. Parallel algorithms are developed and make use of Nvidia's Compute Unified Device Architecture (CUDA). The results gathered highlight a 5 times speedup using a Nvidia GeForce GT 330M GPU as compared to an Intel Core i7 740QM Central Processing Unit (CPU). The processed results display depth wise areas of shale and rock formations as well as water, oil and/or gas reserves.

Keywords: *GPU, Parallel Processing, CUDA, HPC, Well Logging.*

https://doi.org/10.47412/MRUU5197

1. Introduction

As globalization is furthered, the rate at which data is generated and consumed has engulfed and enveloped the lives of the industrial and commercial sector. People are connected now more than ever and the world is currently generating data faster than it can process and make use of [1]. Because of the sequential nature of the Central Processing Unit (CPU) large amounts of data depending on an application can take days, weeks or even months to process. Industries can have up to petabytes of raw unprocessed data being generated with little means of data analysis.

Hydrocarbon exploration is an extremely resource intensive and expensive process [2] with gigabytes of data being generated daily from various sensors and equipment. This data is often analysed over a lengthy period by traditional CPU computing by which the industry has already new data needing analysis.

High Performance Computing aims to solve the issues of Big Data by leveraging parallel computing methodologies. Parallel computing is used extensively in the field of engineering as an emerging powerhouse for data processing. This makes use of multiple compute units to solve a problem by breaking the problem up into discrete parts to be solved concurrently and in turn further splitting the problem into a series of instructions whereby each instruction is executed simultaneously on a different processor. There are two parallel programming topologies, namely multi-core CPU's and GPU's.

GPU's have multiple cores which when used together can perform a massive amount of computation. A GPU utilizes compute rather than control meaning that GPU's use simple control complexity and more compute power allowing for an embarrassingly amount of parallelization. The fundamental architecture of a GPU is much simpler than a CPU, as CPU's are complex processors and are widely used, GPU's are much simpler in design and as a result much more highly specialized. GPU's excel at heavy arithmetic, processing vast amounts of data, highly parallel data can be processed much faster.

Hydrocarbon exploration is the search for oil and gas reserves beneath the earth's surface. A technique used in exploration is borehole geophysics or well logging. Well logging makes a detailed log of the formation's properties to determine if it is a feasible reservoir for hydrocarbon drilling.

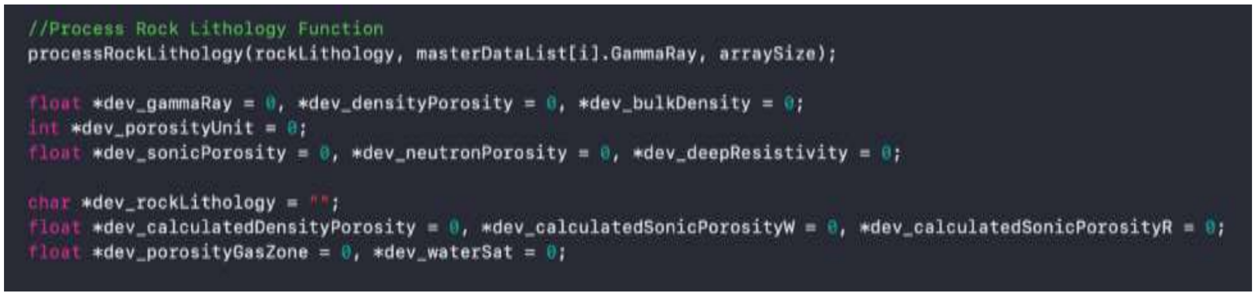
The highly parallel nature of GPU's aims to accelerate this process by utilizing the raw data generated by well logging and employing algorithms developed to predict whether the prospect is an oil producing well, a gas well or neither. GPU's will allow for results to be generated much faster, years of raw data or thousands of prospects with millions of values can be processed concurrently and determine if hydrocarbons are present.

GPU's are many core architectures and using Nvidia's CUDA [3] platform the raw data generated from gigabytes of well logs coupled with parallel algorithms designed for GPU execution aims to reduce the time and complexity needed to determine the hydrocarbon bearing potential of a reservoir. Using the raw data is a viable alternative to using wireline logs or paper-based printouts of the waves generated. The algorithms designed aims to add another layer in the accuracy present and allow for anyone to run raw data and conclude whether an area may contain oil or gas and whether it may just be water or rock. Thousands of well logs or years of data can be analysed in a relatively short space of time.

2. Well Logging

Well Logging or wireline logging used, provides a detailed account of a geologic formation. The oil and gas industry use well log analysis to obtain information on the petrophysical properties of a formation or reservoir and its potential fluid content [1]. Through well logging a reservoir or formation can be indicative of its hydrocarbon bearing potential being an oil or gas zone or its non-hydrocarbon bearing potential being shale sand or water bearing [1].

Wireline Open Hole Logging is a branch of Well Logging used to gather data on a formation for evaluation. Tools are run to obtain measurements of parameters for analysis to determine the condition of the reservoir. The Gamma Ray tool measures the strength of the radiation the formation emits and this determines whether it is a sand or a shale in a siliciclastic environment. Spontaneous Potential measures the potential difference between a mud filtrate of a certain salinity and the area it invades containing water of a different salinity. The formation density is measured by gamma ray Compton scattering by use of a radioactive source and gamma ray detectors. Neutron Porosity is derived from the hydrogen index value that is measured by gamma rays emitted when injected from a source. Sonic is the transit time of compressional sound waves in the formation. Resistivity is the formations resistivity and samples by an induction-type resistivity tool [2]. The petrophysical evaluation of the of the log data with the main properties such as lithology, porosity, permeability and water saturation is essential for the evaluation of the reservoir formation [3]. Figure 1 indicates a code snippet of the petrophysical properties under evaluation.



```
//Process Rock Lithology Function
processRockLithology(rockLithology, masterDataList[i].GammaRay, arraySize);

float *dev_gammaRay = 0, *dev_densityPorosity = 0, *dev_bulkDensity = 0;
int *dev_porosityUnit = 0;
float *dev_sonicPorosity = 0, *dev_neutronPorosity = 0, *dev_deepResistivity = 0;

char *dev_rockLithology = "";
float *dev_calculatedDensityPorosity = 0, *dev_calculatedSonicPorosityW = 0, *dev_calculatedSonicPorosityR = 0;
float *dev_porosityGasZone = 0, *dev_waterSat = 0;
```

Figure 12: Petrophysical Properties

3. CUDA

Single Instruction Stream, Multiple Thread Stream (SIMT) [4] is another type of execution model utilized in parallel computing. This is a variant of the Single Instruction Steam, Multiple Data Stream (SIMD) model whereby single instruction multiple data are combined with multi- threading. The GPU model by NVIDIA uses SIMT as opposed to SIMD in multi-core CPU's [5]. This model fundamentally bears resemblance to SIMD but NVIDIA's CUDA incorporating threads, warps, blocks and grids and the warp uses a thread of SIMD instructions the SIMT model is developed.

CUDA as a parallel programming model and makes use of an instruction set architecture which employs the use of a compute engine from NVIDIA to aid in solving large data heavy computational problems. CUDA is open source and an extension of the standard C programming language [4]. CUDA operates in a two-stage fashion where execution occurs on the CPU known as the host or the GPU known as the device [6]. No data parallelism takes place in the host code and rich data parallelism is implemented in the device code [4]. A CUDA program makes use of a NVIDIA C compiler or NVCC that allows for the both stages to remain independent during the compilation procedure. Both the host and device code make use of the C programming language, the difference with the device lies in the use of extended keywords [7] [6]

CUDA uses a combination of grids, threads and blocks which forms the building block and fundamental architecture of the CUDA programming model. CUDA is built to exhibit massive data parallelism by generating a large number of parallel threads [5]. Generated threads are further grouped into blocks and groups of blocks by threads. Groups of blocks with threads are further grouped into grid which can be executed independently of each other [5] [6]. Blocks are arranged to support a 3D array of threads and each block has a specific block ID (blockIdx). A kernel function is responsible for execution of threads and each thread has a specific thread ID (threadIdx). Blocks are limited to 512 and 1024 threads depending on the architecture or compute capability of the device.

4. Literature

Using an oil reservoir simulator [6], a study was done by Ismail et al [6] to analyse reservoir conditions. A parallel version of the simulator was used utilizing a GPU using CUDA and the SIMT programming model. Oil reservoirs were modelled on 1D, 2D and 3D. The parallel implementation highlights the performance analysis of the GPU versus using the CPU for the same simulation on 1D, 2D and 3D. The experiment was conducted on a Nvidia GeForce GTX 480 GPU which has 15 SMs and 32 cores each versus an Intel Xeon W3520 CPU clocked at 2.67GHz and 8GB RAM. For computation of the speedup achieved by the GPU the implementation on the simulator was conducted in a sequential manner considering 20 iterations within the Conjugate Gradient (CG) model [8] [9]. This process was repeated on a parallel simulator on the GPU using 20 iteration within the CG method. All models were conducted using double precision.

The results indicate that parallel implementation scales well with increasing matrix sizes and the CG has the benefit is fast local memory access of the GPU's built in memory. A speedup of 13 times was achieved for the CG GPU versus CPU for 1D reservoirs and this only increased as the GPU observed 17 times fast for 2D and 26 times faster for 3D. This clearly highlights the performance advantage of a GPU based solver versus a comparable CPU based system for oil reservoir simulation.

A similar study done by Yin et al [15] compared the results performed on both CPU's and GPU's for the computation of formation pressure for multistage hydraulically fractured horizontal wells in tight oil and gas reservoirs. A mathematical model was created and variations in permeability and porosity was considered. By using the CPU-GPU asynchronous computing the solution was broken down into infinite summation and integral forms for parallel computation. Computation was performed on an Intel i5 4590 CPU and in Nvidia GT 730 GPU. The computational speed increase was almost 80 times greater in the GPU platform meeting real time calculations for formation pressure.

5. Methodology

Data sets relevant to the study were sourced from the Kansas Geological Survey (KGS) which a is research and service division of the University of Kansas [12]. The data sets used in the paper were sourced from the KGS because of the wide spread of accessible data for use and the completeness of the data set. The data spanned 1999 to 2018 and soon to be 2019. The data samples were logging document data on multiple wells spanning the Kansas area and were logged and recorded by independent well logging companies. The total number of wells for data analysis was 19,000 wells with the potential for another couple hundred as updated well log data is added every month.

The wireline or well log data is stored in the LAS file format. Over 19,000 wireline logs have its own LAS file for data analysis to determine the contents of that potential reservoir. The LAS files and links from the KGS is downloaded. An algorithm to process the entire data set of 19,000 wells and 20GB of data was created in C++. Since the data was not sorted this algorithm sorted the data into manageable indexable files based on the title of LAS files. Each LAS file name was written to a text file to index the entire data set into an organized dataset. An algorithm in Python was developed due to its ease of use with favourable libraries. This algorithm allowed for the LAS file names in the created text file to be loaded into a python array. The Python array is now used to search through the entire data set to obtain and extract data sets or wells by name. Any individual well in now searchable by name and indexed by its year. When a selected LAS is searched and found it is loaded using lasio [13]. Lasio is a python package to read and write Log ASCII Standard (LAS) files used for petrophysical logs. Lasio was downloaded [14] and the Python module installed allowing for data present in the LAS files such as the curves, units, values, parameters and location to be placed in the dict format.

LAS files are now loaded into the system and each parameter to the study is searched for in the individual files. Once the parameters are found in an individual wireline log file that file is used for data analysis since it is relevant to the study. As a backup to not omit files due to wrongful or incorrect and incomplete naming of parameters a second search is conducted using a description for the parameters under investigation and if any files are found by this method it is used for analysis. If for both the parameter search and description search yields no results it means those parameters are missing and the full data is not present in that log and creation or indexing of the results generated would be ignored for this. Parameters used included gamma ray, spontaneous potential, neutron porosity, density porosity, sonic porosity, spontaneous potential and deep resistivity [15] [16]. Further parameters such as Archie's equation were used for further calculations.

Neutron porosity existed under multiple units so extraction of this parameter from the data involved searching for its base unit and then searching for alternative versions and indicating the different versions extracted. All error reads were checked or NaN values and this was carried out using the python function isnan [17]. This function exists in the math library in python and checks to see if a parameter is true or false.

All error reading is zeroed to allow for computations to be performed easily and accurately. Any value that was not corresponding to the parameters or not present was given and zeroed value so calculations further in the algorithm would be computed correctly.

Parameters that exists are now determined and this data needs to be extracted to be used since all unnecessary data is dealt with. Parameters are extracted from LAS file and loaded into a csv or excel file for easy data manipulation. All the data to process was stored in a master csv file for processing to facilitate faster loading times for data processing.

The parameters to be processed is then loaded and the data which is in the csv file is now extracted is loaded into an algorithm designed in C++ to run on the GPU using Nvidia's CUDA for parallel processing. The CVS is loaded into a struct with arrays for each column of data, the data is the processed in ordered of which data is required for each stage of the processing, so for each stage the required for the stage is loaded from CPU to GPU then the kernel for the first stage of processing is done the data is then sent from GPU to CPU, once completed the memory used is then cleared to prepare for the next stage and the process is repeated until the final stage of processing once the final stage is completed, csv files are also created to display numeric data for each given Las file in their respective years folders. Figure 2 illustrates the process.

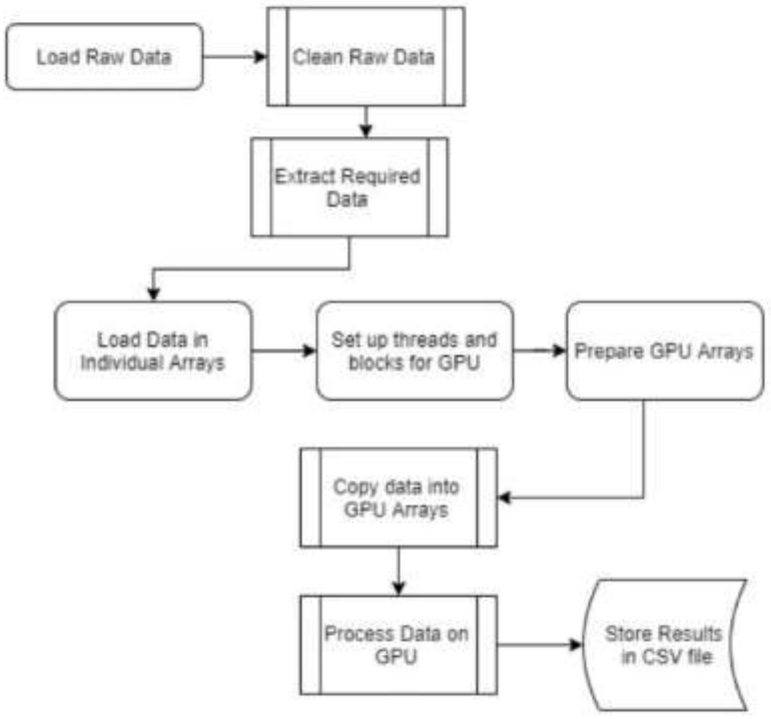

Load Raw Data
Clean Raw Data
Extract Required Data
Load Data in Individual Arrays
Set up threads and blocks for GPU
Prepare GPU Arrays
Copy data into GPU Arrays
Process Data on GPU
Store Results in CSV file

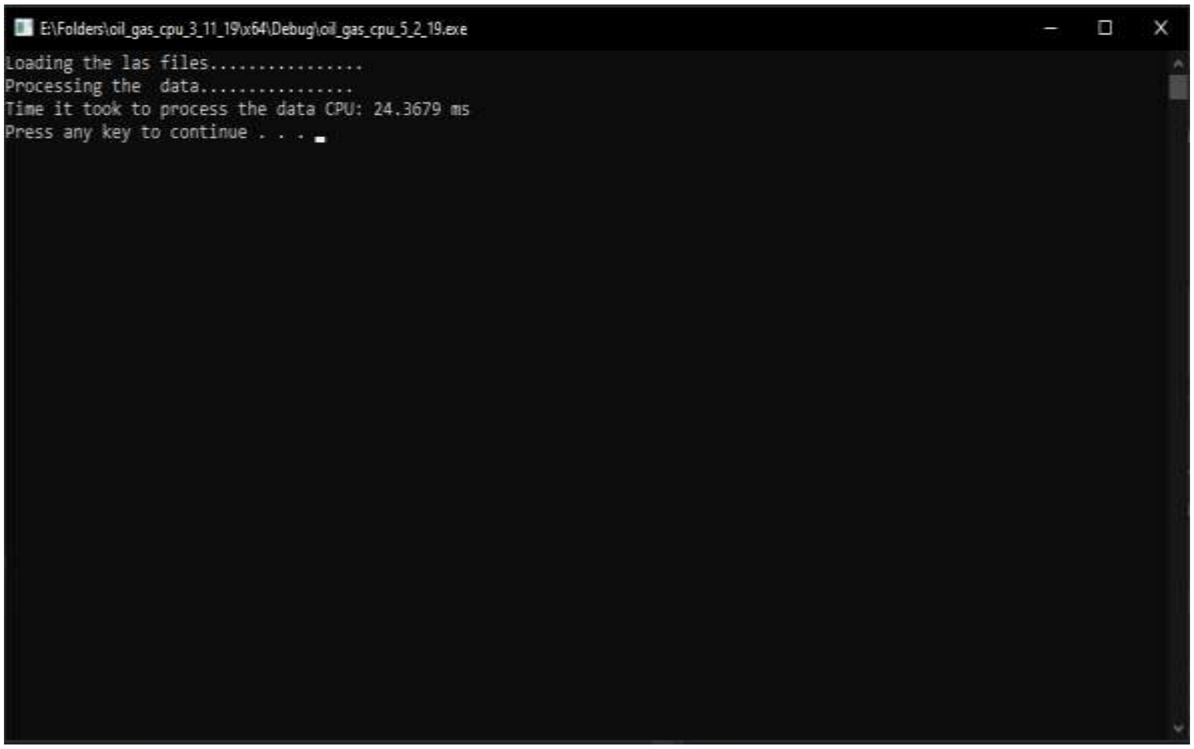
Figure 13: Flowchart of process

6. Results

For the first set of testing and results an algorithm was created to execute sequentially and parallel. Ultimately both algorithms would process the well log data to give results but the way in which it is processed would be different. CPU based execution followed the sequential methodology which was single instruction stream, single data stream (SISD) [18]. Even though a multicore CPU was used it did not utilize multiple cores as in the single instruction stream, multiple data stream (SIMD) architecture. GPU based execution used the single instruction stream, multiple thread stream process.

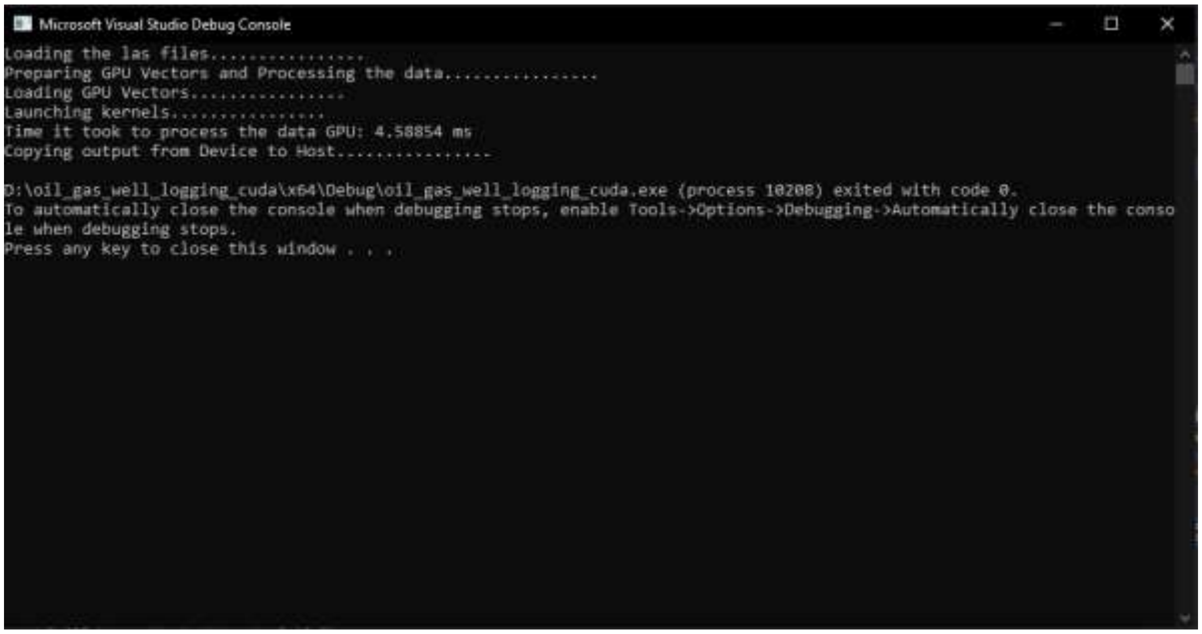
A comparison of CPU vs GPU based execution for the processing of one well log resulted in a speed up of over 500%. This indicated the extreme performance advantage of using GPU's versus CPU's for large data calculations. These results showcased the performance advantage in just one well. Future work will attempt to process the entire data pool of 19,000 wells both sequentially and parallel.

Figure 1 highlights the results of the CPU execution for a single formation, Dolecheck #1 and Fig. 2 presents the same analysis and execution carried out on the GPU with a time of 4.58854ms versus the CPU's execution of 24.3679ms. This highlights a speedup of around 5.3 times faster on the GPU for just one dataset.


E:\Folders\oil_gas_cpu_3_11_19\x64\Debug\oil_gas_cpu_5_2_19.exe

```
Loading the las files...............
Processing the  data...............
Time it took to process the data CPU: 24.3679 ms
Press any key to continue . . .
```

Figure 14: CPU Execution Time



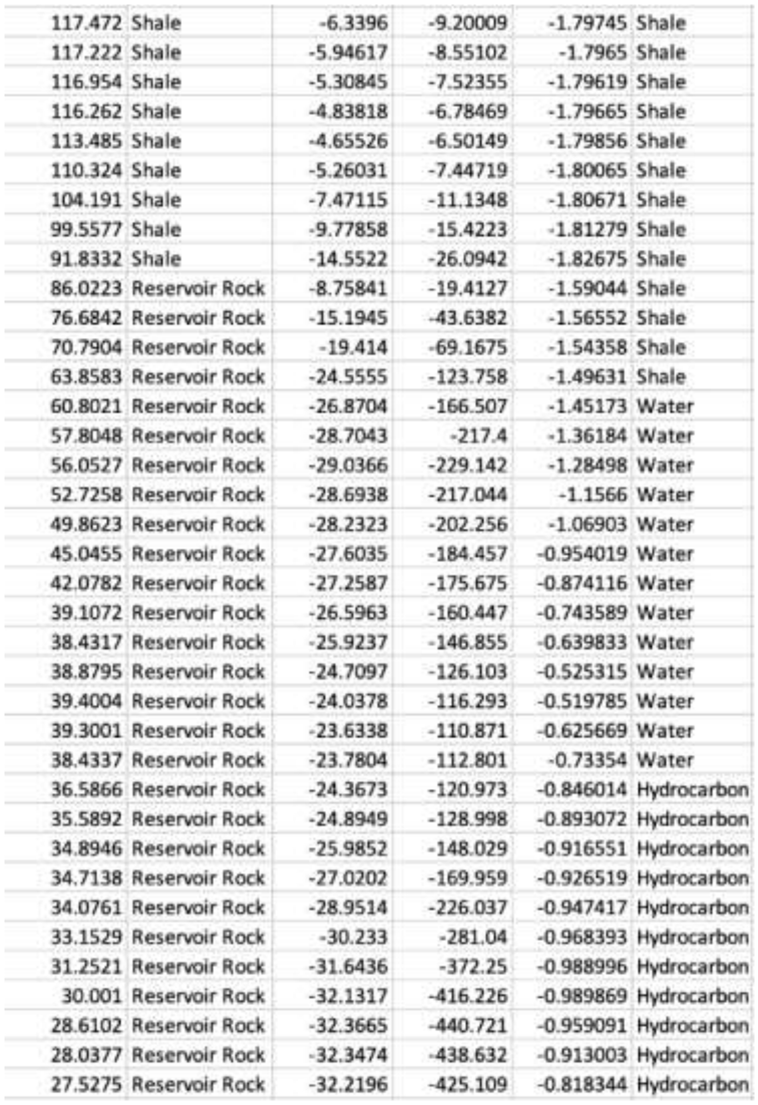
```
Microsoft Visual Studio Debug Console
Loading the las files................
Preparing GPU Vectors and Processing the data................
Loading GPU Vectors................
Launching kernels................
Time it took to process the data GPU: 4.58854 ms
Copying output from Device to Host................

D:\oil_gas_well_logging_cuda\x64\Debug\oil_gas_well_logging_cuda.exe (process 10208) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the conso
le when debugging stops.
Press any key to close this window . . .
```

Figure 15: GPU Execution Time

Figure 3 and Fig. 4 shows the difference in execution times and overall processing times for CPU and GPU based timing favouring parallel implementation. It highlights the advantage of parallel processing for a single well application. Thus, for large data sets where streams of data lie in giga and terabytes parallel processing will provide inherent advantages in oil and gas.

The results were illustrated in an excel based document showing the results for each meter or foot of the reservoir depth wise. Figure 5 displays the processed results in the final stage of formation evaluation. The rightmost column are the results of the well by each foot going downwards. Since reservoirs tend downwards by the thousands of feet only a snippet could have been provided to highlight the results. At the beginning there are areas of shale rock but going downwards there are areas of water saturation detected then hydrocarbon reserves lower. This is how the results are expected to be through the length or depth of the reservoir under study to indicate the contents of the well. This enables the user to see immediately after processing whether this well has a hydrocarbon bearing potential or not.



117.472	Shale	-6.3396	-9.20009	-1.79745	Shale
117.222	Shale	-5.94617	-8.55102	-1.7965	Shale
116.954	Shale	-5.30845	-7.52355	-1.79619	Shale
116.262	Shale	-4.83818	-6.78469	-1.79665	Shale
113.485	Shale	-4.65526	-6.50149	-1.79856	Shale
110.324	Shale	-5.26031	-7.44719	-1.80065	Shale
104.191	Shale	-7.47115	-11.1348	-1.80671	Shale
99.5577	Shale	-9.77858	-15.4223	-1.81279	Shale
91.8332	Shale	-14.5522	-26.0942	-1.82675	Shale
86.0223	Reservoir Rock	-8.75841	-19.4127	-1.59044	Shale
76.6842	Reservoir Rock	-15.1945	-43.6382	-1.56552	Shale
70.7904	Reservoir Rock	-19.414	-69.1675	-1.54358	Shale
63.8583	Reservoir Rock	-24.5555	-123.758	-1.49631	Shale
60.8021	Reservoir Rock	-26.8704	-166.507	-1.45173	Water
57.8048	Reservoir Rock	-28.7043	-217.4	-1.36184	Water
56.0527	Reservoir Rock	-29.0366	-229.142	-1.28498	Water
52.7258	Reservoir Rock	-28.6938	-217.044	-1.1566	Water
49.8623	Reservoir Rock	-28.2323	-202.256	-1.06903	Water
45.0455	Reservoir Rock	-27.6035	-184.457	-0.954019	Water
42.0782	Reservoir Rock	-27.2587	-175.675	-0.874116	Water
39.1072	Reservoir Rock	-26.5963	-160.447	-0.743589	Water
38.4317	Reservoir Rock	-25.9237	-146.855	-0.639833	Water
38.8795	Reservoir Rock	-24.7097	-126.103	-0.525315	Water
39.4004	Reservoir Rock	-24.0378	-116.293	-0.519785	Water
39.3001	Reservoir Rock	-23.6338	-110.871	-0.625669	Water
38.4337	Reservoir Rock	-23.7804	-112.801	-0.73354	Water
36.5866	Reservoir Rock	-24.3673	-120.973	-0.846014	Hydrocarbon
35.5892	Reservoir Rock	-24.8949	-128.998	-0.893072	Hydrocarbon
34.8946	Reservoir Rock	-25.9852	-148.029	-0.916551	Hydrocarbon
34.7138	Reservoir Rock	-27.0202	-169.959	-0.926519	Hydrocarbon
34.0761	Reservoir Rock	-28.9514	-226.037	-0.947417	Hydrocarbon
33.1529	Reservoir Rock	-30.233	-281.04	-0.968393	Hydrocarbon
31.2521	Reservoir Rock	-31.6436	-372.25	-0.988996	Hydrocarbon
30.001	Reservoir Rock	-32.1317	-416.226	-0.989869	Hydrocarbon
28.6102	Reservoir Rock	-32.3665	-440.721	-0.959091	Hydrocarbon
28.0377	Reservoir Rock	-32.3474	-438.632	-0.913003	Hydrocarbon
27.5275	Reservoir Rock	-32.2196	-425.109	-0.818344	Hydrocarbon

Figure 16: Results of Formation Evaluation

7. Discussion

This study attempted to use GPU's using parallel processing methodologies and algorithms modelled after petrophysical parameters to determine the contents of a formation under study. The same parameters geoscientists use for well log evaluations were modelled into an algorithm made for both sequential and parallel processing paradigms. Raw well log data used for wireline logging made up the bulk of the dataset and for this study Dolechek #1 well in the Kraft-Prussia Field was under study. Castle Resources gathered the borehole data and the service was provided by Gemini Wireline LLC in the county of Barton, state of Kansas. The log was taken at Friday 20th June 2014 with a well ID of 15 009 25991. The results presented one well under study showcased how supercomputing methodologies accelerate data processing as compared to traditional serial computing methods. We clearly see the increased calculation speed on one dataset. It is likely that computations performed on the entire well log suite of data would have significantly take faster using the supercomputing approach. From the results at each stage of the data processing the algorithm models the same approaches used in well log analysis and the results of the formation under study are carried our depth wise to indicate areas of potential hydrocarbon bearings. This approach allows for any well under study to use the data generated to determine the contents of a formation in much less time as compared to manual wireline logging. Typically, a petroleum geologist will have to perform calculations and computing over the entire body of the formation to determine its hydrocarbon bearing potential. Through this process analysis can find hotspots in milliseconds so geologists can recheck the areas under study for accuracy. The algorithm is designed to skew towards a few false positives meaning some areas may be flagged as hydrocarbon bearing when in reality it may not be but this is done as a safety precaution for all areas for hydrocarbons can be found.

8. Conclusion

A detailed analysis of the well log data was carried out for reservoir characterization of an oil field from the Kansas Geological Society. Using a suite of well log parameters applied to reservoir data lithological interpretation was carried out on the reservoir's contents and the results assessed to determine the content of the formation under evaluation. The well under study Dolchek #1 was found to consist of primarily shale with pockets of hydrocarbons and water at varying depths. Further testing will attempt to analyse years of data and thousands of wells at the same.

References

[1] M. Steinberger, M. Kenzel, B. Kainz, J. Muller, P. Wonka and D. Schmalstieg, *"Parallel Generation of Architecture on the GPU,"* Computer Graphics Forum, vol. 33, no. 2, pp. 73-82, 2014.

[2] S. Kark, E. Brokovich and N. Levin, *"Emerging conservation challenges and prospects in an era of offshore hydrocarbon exploration and exploitation,"* Conservation Biology, vol. 29, no. 6, 2015.

[3] L. Shi, H. Chen, J. Sun and K. Li, *"vCUDA: GPU-Accelerated High-Performance Computing in Virtual Machines,"* IEEE Transactions on Computers, vol. 61, no. 6, pp. 804-816, 2012.

[4] B. Das and R. Chatterjee, *"Well log data analysis for lithology and fluid identification in Krishna-Godavari Basin, India,"* Arabian Journal of Geosciences, 2018.

[5] T. Darling, *"Gulf Drilling Guides,"* in Well Logging and Formation Evaluation, Gulf Professional Publishing , 2005, pp. 1-27.

[6] M. Rider, The Geological Interpretation of Well Logs, Houston, 1996.

[7] F. Brian, J. Nickolls, H. P. Moreton and B. Coon, *"Implementation of Arrays of Structures on SIMT and SIMD architectures"*. United States of America Patent 8751771, 10 June 2012.

[8] J. Sanders and E. Kandrot, CUDA by Example, Boston: Pearson Education, 2011.

[9] D. Kirk and W.-m. Hwu, Programming Massively Parallel Processors., Morgan Kaufmann, 2012.

[10] Nvidia Corporation, *"Nvidia Cuda,"* April 2018. [Online]. Available: https://docs.nvidia.com/cuda/archive/9.1/pdf/CUDA_C_Programming_Guide.pdf. [Accessed November 2019].

[11] E. Guizzo, *"Geophysics Solving the Oil Equation,"* IEEE Spectrum , no. 45, pp. 32-36, 2008.

[12] L. Ismail, J. Abou-Kassem and B. Qamar, *"Implementation and Performance Analysis of a Parallel Oil Based Reservoir Simulator Tool using a CG Method on a GPU based System,"* International Conference on Computer Modelling and Simulation, pp. 375-380, 2014.

[13] K. Theobald, R. Kumar, G. Agrawal, G. Heber, R. Thulasiram and G. Gao, *"Developing a Communication Intensive Application on the EARTH Multithreaded Architecture,"* in 6th International Euro-Par Conference on Parallel Processing, Heidelberg, 2000.

[14] F. Chen, K. Theobald and G. Gao, *"Implementing Parallel Conjugate Gradient on the EARTH Multithreaded Architecture,"* in IEEE International Conference on Cluster Computing, 2004.

[15] R. Yin, Q. Li, P. Li, Y. Guo, Y. An and D. Lu, *"GPU Based Computation of Formation Pressure for Multistage Hydraulically Fractured Horizontal Wells in Tight Oil and Gas Reservoirs,"* Hindawi Mathematical Problems in Engineering, vol. 2018, no. 2582797, p. 10, 2018.

[16] Kansas Geological Survey, *"Kansas Geological Survey,"* [Online]. Available: http://www.kgs.ku.edu/General/staffIndex.html. [Accessed 05 February 2019].

[17] MIT, *"Log ASCII Standard (LAS) files in Python,"* [Online]. Available: https://lasio.readthedocs.io/en/latest/. [Accessed 24 February 2019].

[18] MIT, "GitHub," [Online]. Available: https://github.com/kinverarity1/lasio. [Accessed 25 February 2019].

[19] Schlumbeger, Log Interpretation Priciples/Applications, Schlumbeger Educational Services, 1991.

[20] R. Hosein, PENG 6028 Lecture Notes, 2019.

[21] LearnAndLearn, *"Python isnan Function,"* [Online]. Available: https://learnandlearn.com/python-programming/python-reference/python-isnan-function. [Accessed 15 March 2019].

[22] J. Hong, *"Verilog HDL,"* in Architectures for Computer Vision, Singapore, John Wiley & Sons, Ltd, 2014, pp. 1-10.