University of Moratuwa

Faculty of Engineering

Final Individual Work Report

Self-accelerating Processing Workflows

Supervisors:

Dr Adeesha Wijayasiri, CSE, UOM.

Mr Janaka Perera, LSEG.

Team member:

160256U - Jeyakeethan Jeyaganeshan

Introduction

This report is regarding my contribution towards the final year project for the module, CS4202 - Research and Development Project. This would show how I have individually contributed to the project, Self-accelerating Processing Workflows. We were three people who have same idea and interests formed a team. We selected the above project that was suggested by the London Stock Exchange Groups (LSEG) and completed successfully.

The keen to explore more computational resources due to parallel computing has set the heterogeneous computing on its trend nowadays. Utilization computational resources appropriately would save investment, maintenance cost and may improve the overall performance also. This was the motivation for all three of us to select the project which was to utilize the CPU over GPU wherever possible and profitable. Initially we had very less knowledge regarding the domain, but we were curious. First, we met supervisor to understand on what LSEG wants us to do the research. We were not crystal after the meeting. We did literature survey on a wrong topic after the first meeting, yet it became the foundation of the research. But we could understand the problem completely after only the third meeting with the supervisor. Since we were assigned with an external supervisor from the LSEG we could not meet often in person, but we often contacted over WhatsApp. The supervisor had very clear idea about the project and guided us whenever we were struggling.

We have started the implementation of the project after the literature survey directly to know the feasibility of the research. We tried several technologies ourselves before finalized it with C++. But we could have a great understanding of GPU programming and its scalability. We could not continue as usual because to the pandemic situation and we had been asked for a repeat for progress evaluation saying contribution was not enough. However, we got together in Jaffna, apart from the University of Moratuwa, compensated the lack in the timeline and attempted for the second chance given after two weeks.

We improved the solution step by step like prototyping and came up with a final solution called, hybrid prediction model. All solutions along the timeline are based on a general class called computational model, an abstract main class responsible for the evaluation and optimal processor selection in the library, expected solution for our research problem. We divided the research into small tasks and completed on time.

Group composition

We were people having the same interest, B.Abinayan (160007J), myself and T.Nirojan (160442L) has formed a group for the final year research module. Each people played different roles during the research for different needs. The research was divided into equality weighted tasks and we chose tasks of our interest and capabilities.

Individual contribution

I have divided this section into three to structure and simplify this report. Literature contribution, Technical contribution, and the Non-technical contribution. Literature contribution is about literature survey, reading, annotation, citation etc. Technical contribution is about experiments, design, implementation, and evaluation details. Finally, the non-technical contribution shows the works related to reports, slides etc.

1. Literature review

We went through lots of papers as supporting and citation is inevitable for a research. Initially, we selected some papers that gives introduction to the GPU programming, read and shared knowledge among us which gave us an into to the context. We selected several papers after a literature survey and divided the papers that were related to our research read them one by one. We have used 19 papers read throughout the research to support our assumptions and statements. Also, we read papers like friend sharing books that everyone read different papers at a time to cover all related knowledge soon.

In paper “Seamlessly Portable Applications” [1], the authors proposed a solution with hardware specific code released by its vender despite our approach where we asked the general implementations of accelerators from the programmer. They keep code for all accelerators in the system and an API is used to access required function. There can be more than one implementation for an accelerator where programmer would choose from them. DLS, the system that chooses the appropriate processing unit will look at the performance database to analyse the history runs of that implementation and compare them to take decision. If the database is empty for the first run it would run on all kind of accelerators and save elapsed times for them. It will also interpolate for execution times of other problems sizes to take the decision. It will not switch to another platform during runtime once the platform is selected. The hard part of the solution was to decouple the accelerator code from the main flow.

We can summarize all the papers read into 3 categories,

1. Execute implementations from existing API, selection based on previous run of the same size. This would not consider present loads in the processors, but the decisions are based on historical data gathered on the system previously. This is close to our research yet need some improvements.
2. Run on all accelerators first finish to kill all others. It would reduce latency very much, but it utilizes all the resources until one find the answer which is costly and waste of resources. We did not go with a similar approach like this.
3. Load balancing: This is a traditional approach where tasks are assigned to the processors which has less contention and try to balance them with equal workload. This kind is not related to our research.

2. Technical contribution

a) Introduction to the context

Heterogeneous computing has an emerging trend nowadays. Different processing units in a heterogeneous system makes the application development more challenging [1]. Effective computational speed can be increased by assigning computations to their optimal processors. we are considering CPU and GPU. The GPU was originally invented for graphical processing where immense parallel but later used for general purpose computing. Though GPUs show higher performance for compute intensive parallel data streams, CPUs can complete some small or complex tasks in less time up to some certain limits. This is because of the overheads due to the high data transfer time between host and device and high context switching time of GPU compared to CPU. The boundary varies with hardware of the system and also depend on the availability of the GPU and CPU resources. Problems that can only be solved either by a GPU or a CPU are eliminated from this research. The main objective is to improve the effective performance of the production systems by utilizing the CPU instead of GPU wherever possible.

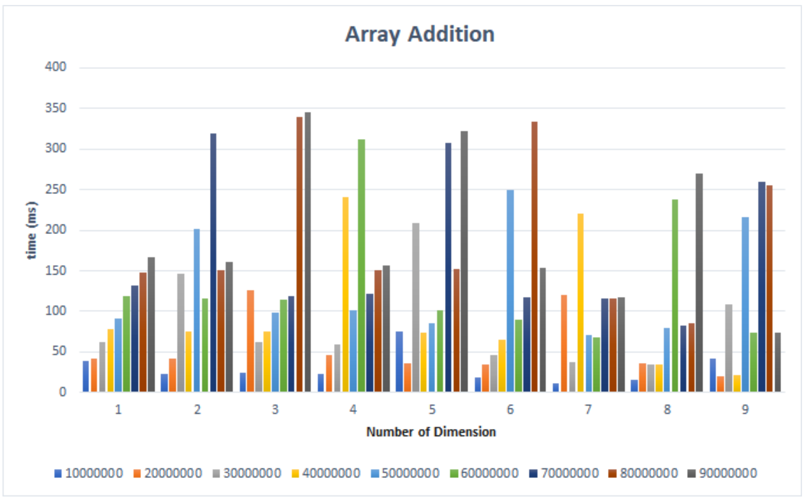
This concept has been adopted from and motivated by the branch predictor hardware implementation where a CPU determines the right branch in a conditional code block within program that is more likely to be taken, in advance. The framework targets the developers and expects to provide a way to do the processor selection automatically based on attributes related to the computations and present utilization of the processors. Developers need to write code for both CPU and the GPU but do not have to worry which is the execution-time optimal processor. It will be very useful in a heterogeneous environment and this concept might be extensible for other processors, FPGA (Field Programmable Gate Array) and TPU (Tensor Processing Unit) as well. A set of attributes that influences the execution of tasks is gathered from developers using a method and used to predict the appropriate processor. We have gone through several research papers related to the topic. None of them has considered present workloads in the system, but some operating system scheduling tasks related research projects do.

b) Problem statement

"Develop a library that evaluate and predict the optimal processor at runtime which has less latency and high throughput for compute intensive tasks at different instances in a real time heterogeneous environment to utilize CPU when it is idle."

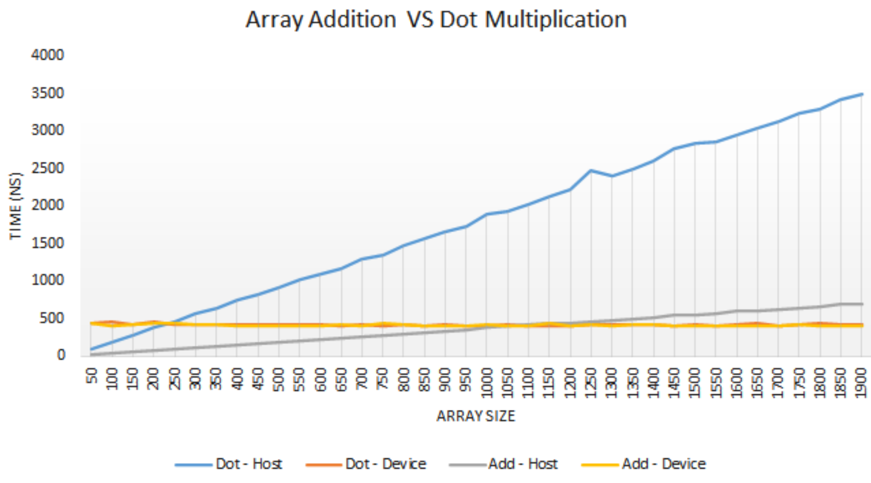
c) Explore GPU in python conda environment.

Before creating a solution for our project, we have experimented and see how GPU programming works and the nature of the GPU. First, we used python which was easy to code and conda framework was used for GPU executions over the python. We used a simple task, element wise array addition for this. We conducted experiments by changing the array size from 10 to 10000 and the dimension from 1 to 9. The execution time measured by wrapping the lines of code on python, executed multiple times and averaged. We plotted graphs in excel using the data which gave us a brief introduction how GPU works with complexity and dimension of tasks. The conda created the object code for the GPU for us and executed it.



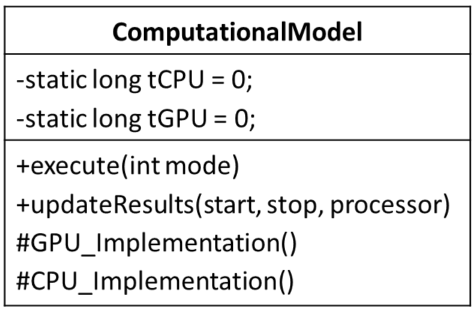
d) Task based analysis in Pycuda environment.

Pycuda is different from conda and it must be provided with the corresponding kernel for GPU execution. We used it to do some task-based analysis over two different tasks. The tasks were element wise array addition and dot matrix multiplication and custom kernels were used for GPU along with "for loop" for CPU execution. The following plotted graphs shows that different tasks have the boundary points at different levels and it also varies for different hardware. Therefore, we concluded that statistical benchmarking would not lead to a general solution for all tasks works on all systems as the same. We moved to the time-based analysis next while shifted to C++ technology.



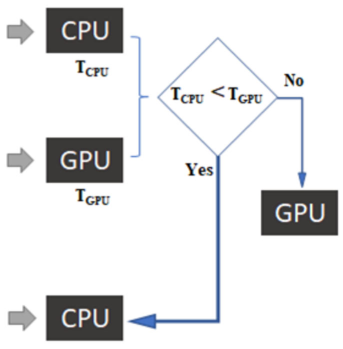
e) Design the solution.

It was not a task that could be accomplished by an individual member. We all came up with different ideas and did idea collision. We designed an initial solution after analyzed all ideas proposed by everyone. That was a library with two classes. "ComputationalModel", an abstract class that controls the execution flow and creates two times stamps before and after the execution of each inputs. "WorkflowController" class that calculate time difference of the stamps asynchronously and set it to the ComputationalModel class back. Implementations for CPU and GPU must be given by the programmer in a class extending the abstract class as it is a general solution. This template was followed from beginning to the end of this research.



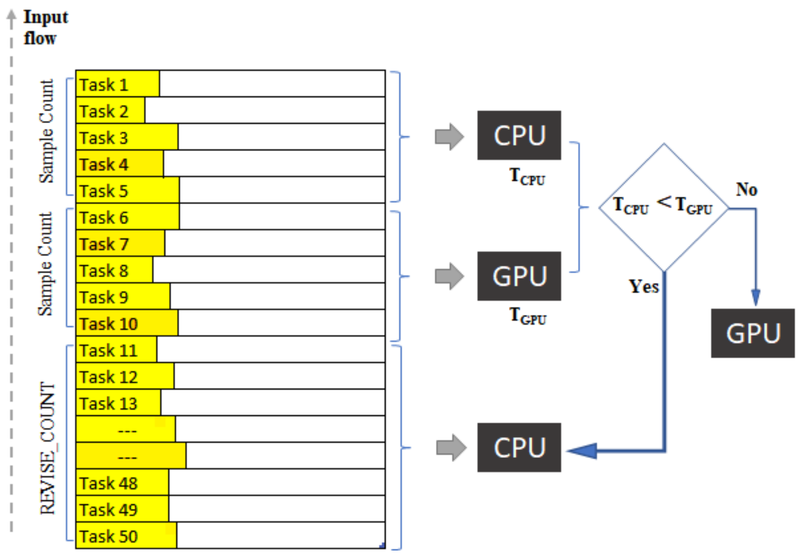
f) Last N Average Time Comparison.

We have moved to “DotMultiplicationModel” when this algorithm has been formed. We also removed the asynchronous updating to reduce thread overheads and to avoid race conditions. Since we found that statistical based solution would not work, we tried to include the execution time for decision making and this solution raised. This is very similar to the last model, but it would maintain last execution times of processor types. The processor which has less execution time would take the next chance. The fall in this algorithm was the processor that had a high execution time unexpectedly may not have a chance again. All execution time-based algorithms explained here, including this works as per flow of following diagram.



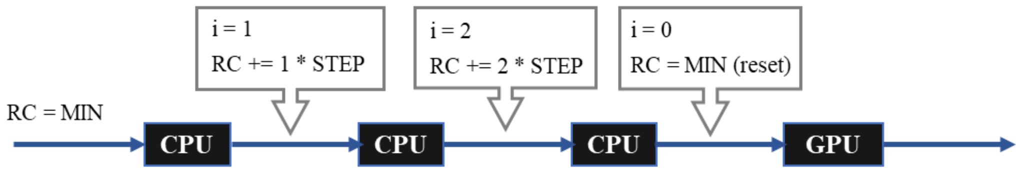
g) Sampling execution time algorithm

This algorithm was developed to overcome the issue, processor which has high time average would not get a chance. This solution was object oriented. It means no static variable paly role here and each object from the classes are evaluated separately. Two subsequent sets of inputs from input flow are executed on both processors every time and the one had less time is set for the next batch of size, REVISE_COUNT. This removed the bottleneck from the algorithm and the evaluation of execution is refreshed whenever sampled. However, the sampling process has an overhead and repeating evaluation unwantedly when a certain kind of inputs are coming continuously is waste. For example, CPU preferred problems coming for long time. It is possible in production environment because their system contains powerful CPUs with 10s and 100s of cores. Execution time of each processor is added to the two variables in the main class separately during sampling and used for optimal processor selection for next batch. It would use some additional resources during sampling. The following image shows how the sampling algorithm works.



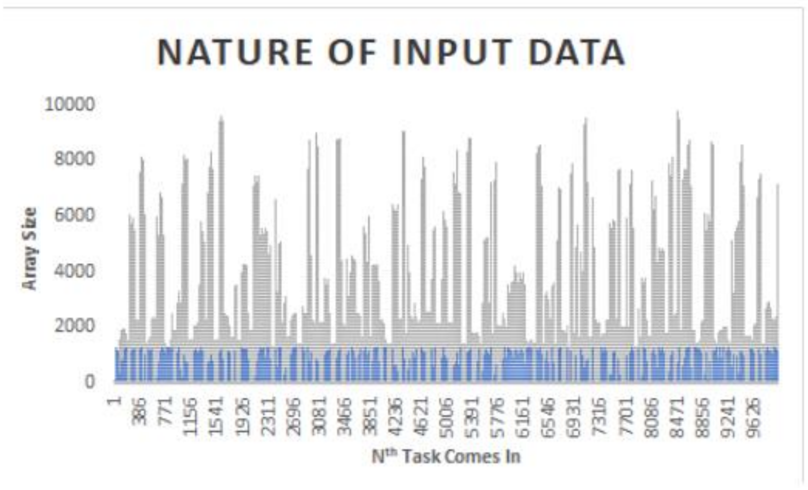
h) Sampling algorithm with varying revise count

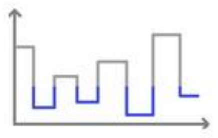
To solve the unwanted sampling issue, we modified the algorithm with incrementing REVISE_COUNT value. If a processor wins in sampling continuously the REVISE_COUNT value is increased by number of successive wins * STEP_VALUE. And the value is reset to REVISE_COUNT_MIN if the other processor wins. A counter was used to count the successive wins and set it to zero if there is a switch in the processor. It improved the performance of our system very much. Yet, this algorithm could not handle outliers. It means larger problems got executed in CPU caused severe delays which was not preferable and recouped the gains earned through other executions.

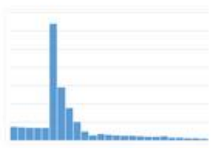


i) Log data and draw data in excels.

We experimented the above algorithm using 5 type of input stream generated by another member. I coded in required places to log the generated input stream to plot graphs, and analyze it based on nature and distribution. After the execution, the logged text data copied into an excel sheet and graphs drawn. An example graph shown below,







j) Matrix multiplication CUDA kernel implementation.

This was very complex kernel compared to other two kernels, array addition and dot multiplication. There was no source code that fitted our requirement, but we have tried some but end up with different answers. Finally, a kernel from concurrent programming module adopted and modified appropriately. One evaluator said that we must have utilized all cores of the CPU. That part was implemented by another member simultaneously. This kernel would work for any dimension of matrices up to 1024x1024.

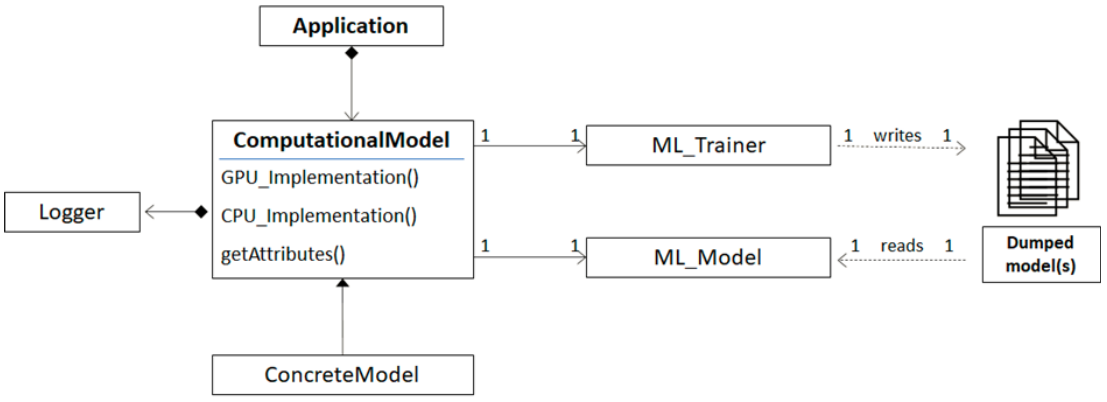
k) XgBoost machine learning model integration.

The XgBoost was integrated to our system from a pre-existing C++ library. However, it was not so easy but required some effort to integrate into the system. Also, the configurations and training part were adopted from the library. But a new C++ class that represented an ML model corresponds to the "MatrixMultiplicationModel" was implemented. This XgBoost integration was a challenging and time consumed task for us.

l) Hybrid of Sampling Algorithm and ML.

We had begun input weights-based analysis as the time-based algorithm could not handle outliers. Though input weights-based ml solution solved the issue, it would not hope with the present loads in the processors since their decisions were static. Therefore, we combined both strategies to create a hybrid solution. The sampling algorithm would do it job except whenever it is CPU turn the input is checked using ML before the execution. A method called catchOutlier was implemented that returns a Boolean and called with the weights of an input before wherever CPU execution takes place. The input is directed to GPU and returns if it is caught as an outlier, but the algorithm continues as usual and otherwise.

```
if (catchOutlier(getAttributes())) {
        GPUImplementation();
        return;
}
```

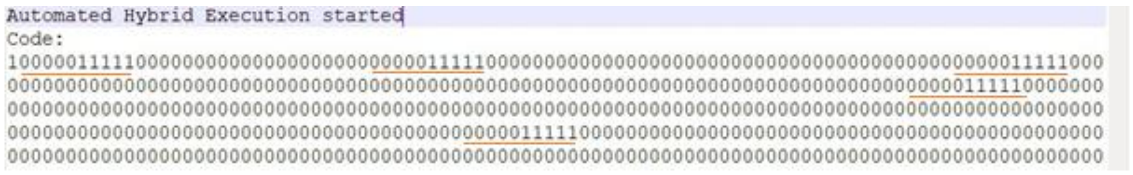


m) Switch due max outlier switches implementation.

The outlier catching strategy worked well and avoid undesirable delays due to larger problems freeze the CPU. But It would continuously check and execute inputs on GPU one by one when GPU preferred inputs come in continuously. It would be unwanted overhead but could be sent to the GPU as a batch. Therefore, we have implemented the above feature using a counter which keeps a count of subsequent catches of outliers and set GPU for next batch of inputs in the count exceeds a limit, say 5 during our experiments. Also, reset it to zero if a non-outlier problem crossed in the middle of the subsequent catches.

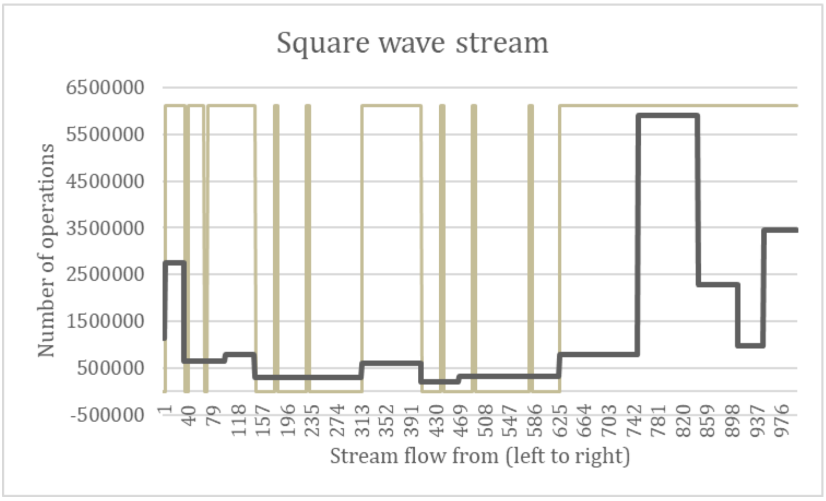
n) Collect data into excel and draw graphs to evaluate the final algorithm.

Evaluation was the important part of a research after it has been completed. We have logged the hybrid model decisions using zeros and ones, zero for CPU and one for GPU. A string stream buffer was used in the implementation to collect the data and written after all executions were finished as direct write to the physical media would cause additional overhead. While the generated input streams for experiment were also logged for analysis. We have also logged for manual executions on CPU only and GPU only, and ML only flow executions to verify them as well. The following segment from the logged data shows the decisions taken by the hybrid solution for a CPU oriented stream.



```
Automated Hybrid Execution started
Code:
100000111110000000000000000000000000011111000000000000000000000000000000000000000000000011111000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000111110000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000111110000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
```

The hybrid model has been experimented using 5 input streams for evaluation. The data collected in the logged file was transferred and converted into an excel sheet and graphs were drawn appropriately to analyze easily. The following figure shows an example graph drawn for square wave input stream.



On the graph, x axis shows input stream flow, and the y axis shows number of operations in the inputs. The thin line shows hybrid model decisions, low and high logged like above. High denotes the GPU execution and low denotes the CPU execution. The surge peaks along the thin line mean to the sampling phase, the 5 samples executed on the GPU. The thick line indicates the flow of the input stream. This graph shows that the algorithm had almost taken the right decisions as thin line edges tightly wraps the thick line edges.

3. Non-technical contribution

Other than the technical activities each of us had to perform some other non-technical activities. E.g., documentation such as report preparation for progress and final report of the research, slides preparation for evaluations, event organization, poster presentation, demo video etc. We prepared all such reports and presentations by sharing it via the Google or MS cloud services, discussed in a conference call and made them together. Therefore, the non-technical activities mentioned above were equally contributed by every one of us.

Challenges

There was pandemic issue during the last half part of the research period. Therefore, we could not meet supervisor or arranged a meeting among us. However, we have used online learning tools such as zoom, Google, MS cloud services to share ideas, to form algorithm and to discuss with our external supervisor. Also, we had the implementation project pushed onto GitHub. Hence, we could divide into small tasks during the discussion, implemented and merged them over the repository.

Other than that, we had not fixed a technology for the experiment at the beginning. We were conducting blind experiments on Conda, Pycuda which were high level programming libraries where we could not achieve the performance gain. But we did get some gains, but we dropped them all and switched to C++ and CUDA framework finally to interact with hardware closer with more flexibilities.

We did not have a method to measure the execution time to evaluate and show gain achieved by the algorithm at the beginning. The time was negligible and could not be measured with the beginning models. We used windows.h header within our C++ implementation for high precision time measurements. Therefore, we needed servers with windows OS to evaluate our final solution on a high-end machine but we did not have the chance at all.

Individual contribution summary

Section	Subsection	Contributors
Dive into the context (statistical analysis)	Explore GPU in python conda environment	Abinayan, Jeyakeethan
	Task based analysis in Pycuda environment	Jeyakeethan, Nirojan
Formation of algorithm development phase - 1	Design the solution	Jeyakeethan, Nirojan, Abinayan
	Computational model class	Jeyakeethan
	Array addition class and kernel	Abinayan, Nirojan
	Dot matrix class and kernel.	Abinayan, Nirojan
	Last N average time comparison	Jeyakeethan, Nirojan, Abinayan
	Sampling execution time algorithm	Jeyakeethan, Nirojan, Abinayan
	Sampling algorithm with varying revise count	Jeyakeethan, Nirojan, Abinayan
Evaluate the latest algorithm	Generate 5 kind of input streams	Nirojan
	Log data and draw data in excels	Jeyakeethan
Logger class integration		Abinayan
Matrix multiplication model integration	Kernel implementation	Jeyakeethan, Nirojan
	OpenMP code for CPU	Abinayan
Formation of algorithm development phase - 2	ML dataset generation	Nirojan
	XgBoost machine learning model integration	Jeyakeethan, Nirojan,
	Caching mechanism	Abinayan
	Hybrid of Sampling Algorithm and ML	Jeyakeethan, Nirojan, Abinayan
	Switch due max outlier switches implementation	Jeyakeethan
Evaluate the final algorithm	Generate 5 kind of pair of matrix input streams	Abinayan
	Logging hybrid decision	Nirojan
	Collect data into excel and draw graphs	Jeyakeethan

References

[1] M. Kicherer, F. Nowak, R. Buchty and W. Karl, "Seamlessly portable applications: ACM Transactions on Architecture and Code Optimization", vol. 8, no. 4, pp. 1-20, 2012. Available: 10.1145/2086696.2086721 [Accessed 27 August 2020]. https://dl.acm.org/doi/pdf/10.1145/2086696.2086721

[2] Z. Memon, F. Samad, Z. Awan, A. Aziz and S. Siddiqi, "CPU-GPU Processing", IJCSNS International Journal of Computer Science and Network Security, Vol.17, No.9, September 2017. [Accessed on: 30- Dec- 2019] [Online]. http://paper.ijcsns.org/07_book/201709/20170924.pdf

[3] A. Syberfeldt and T. Ekblom, "A comparative evaluation of the GPU vs. The CPU for parallelization of evolutionary algorithms through multiple independent runs", International Journal of Computer Science & Information Technology (IJCSIT) Vol 9, No 3, June 2017. [Accessed: 31- Dec- 2019] [Online]. http://aircconline.com/ijcsit/V9N3/9317ijcsit01.pdf

[4] Vella, F., Neri, I., Gervasi, O. and Tasso, S. (2012). A Simulation Framework for Scheduling Performance Evaluation on CPU-GPU Heterogeneous System. Computational Science and Its Applications – ICCSA 2012, pp.457-469. [Accessed:21 Jan. 2020] [Online]. https://link.springer.com/chapter/10.1007/978-3-642-31128-4_34

[5] Cullinan, C., Wyant, C. and Frattesi, T., 2020. Computing Performance Benchmarks among CPU, GPU, and FPGA. [Accessed 27 March 2020] [online]. http://Computing Performance Benchmarks among CPU, GPU, and FPGA

[6] S. Goyat, A. Sahoo, "Scheduling Algorithm for CPU-GPU Based Heterogeneous Clustered Environment Using Map-Reduce Data Processing", ARPN Journal of Engineering and Applied Sciences, Vol. 14, No. 1, January 2019. [Accessed on: 31- Dec- 2019] [Online]. http://www.arpnjournals.org/jeas/research_papers/rp_2019/jeas_0119_7546.pdf

[7] P. C. Pratt-Szeliga, J. W. Fawcett and R. D. Welch, "Rootbeer: Seamlessly. Using GPUs from Java," 2012 IEEE 14th International Conference on High-Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems, Liverpool, 2012, pp. 375-380. https://ieeexplore.ieee.org/document/6332196

[8] Cullinan, C., Wyant, C. and Frattesi, T., 2020. Computing Performance Benchmarks among CPU, GPU, and FPGA. p.15. [online] [Accessed 29 March 2020] https://web.wpi.edu/Pubs/E-project/Available/E-project-030212-123508/unrestricted/Benchmarking_Final.pdf

[9] Lee, V., Kim, C., Chhugani, J., Deisher, M., Kim, D., Nguyen, A., Satish, N., Smelyanskiy, M., Chennupaty, S., Hammarlund, P., Singhal, R. and Dubey P., “Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU”. [online] [Accessed 29 March 2020] https://dl.acm.org/doi/pdf/10.1145/1815961.1816021?casa_token=WGkb9giVyI8AAAAA:n3DRHmgyO46x6w3e-F12nW-pGJ9P9Pjzvntbs6DdXp4Eg2fYzQxo43Akqde585XkHFjEaDDi0oOd

[10] M. Kicherer and W. Karl, “Automatic task mapping and heterogeneity-aware fault tolerance: The benefits for runtime optimization and application development”, Journal of Systems Architecture - Embedded Systems Design, 2015. [online] [Accessed 29 March 2020]. https://dl.acm.org/doi/10.1016/j.sysarc.2015.10.001

[11] A. Chikin, J. Amaral, K. Ali and E. Tiotto, Toward an Analytical Performance Model to Select between GPU and CPU Execution, 2019. Available: https://ieeexplore.ieee.org/document/8778216. [Accessed 30 August 2020]. https://doi.org/10.1016/j.sysarc.2015.10.001

[12] J. Dollinger and V. Loechner, "Adaptive Runtime Selection for GPU," 2013 42nd International Conference on Parallel Processing, Lyon, 2013, pp. 70-79, doi: 10.1109/ICPP.2013.16. [online] [Accessed 29 March 2020]. https://ieeexplore.ieee.org/document/6687340

[13] C. Augonnet, S. Thibault, R. Namyst, PA., Wacrenier, “StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures”, 2009. Lecture Notes in Computer Science, vol 5704. Springer, Berlin, Heidelberg. [online] [Accessed 29 March 2020].
https://doi.org/10.1007/978-3-642-03869-3_80

[14] J. Shen, A. L. Varbanescu, Y. Lu, P. Zou and H. Sips, "Workload Partitioning for Accelerating Applications on Heterogeneous Platforms," in IEEE Transactions on Parallel and Distributed Systems, vol. 27, no. 9, pp. 2766-2780, 1 Sept. 2016, doi: 10.1109/TPDS.2015.2509972. [online] [Accessed 29 March 2020].
https://ieeexplore.ieee.org/document/7360199

[15] M. P. Robson, R. Buch and L. V. Kale, "Runtime Coordinated Heterogeneous Tasks in Charm++," 2016 Second International Workshop on Extreme Scale Programming Models and Middlewar (ESPM2), Salt Lake City, UT, 2016, pp. 40-43, doi: 10.1109/ESPM2.2016.011. [online] [Accessed 29 March 2020].
https://ieeexplore.ieee.org/document/7831559