Self-accelerating Processing Workflows

Progress Presentation

Supervisors:

Dr Adeesha Wijayasiri, CSE, UOM

Mr Janaka Perera, LSEG.

Group members

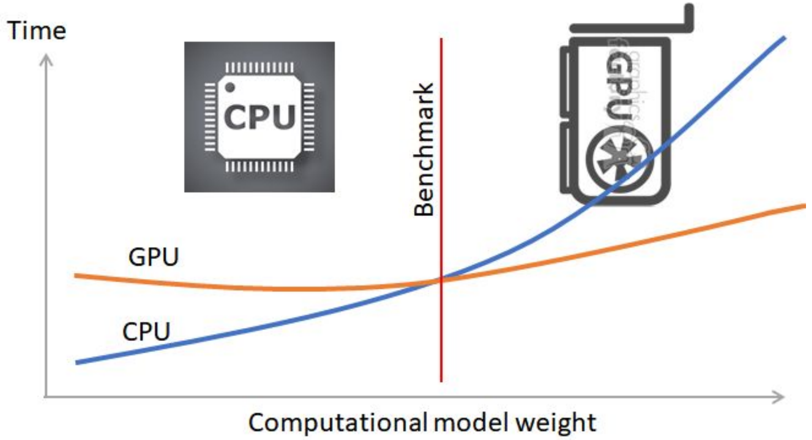
160007J - Balarajah Abinayan

160256U - Jeyakeethan Jeyaganeshan

160442L - Thiyakarasa Nirojan

Introduction

❖ A particular task can be executed in both CPU or GPU but the execution time may vary depending on many factors.



Introduction

- Profit from different processing units for the tasks are varying and depending on various factors such as properties of the accelerators, deployment environment, time of the day, complexity of the tasks and system's current state influence significantly, e.g. contention [14][19].
- Hence, the tasks cannot be pre-classified whether they are efficient to run on GPU or CPU during the programming period.

Introduction

- ❖ Inappropriate scheduling of computations into wrong processors are inefficient and time consuming [17].
- ❖ Many applications utilize GPUs to seek gain but leave CPUs sitting idle [18].
- ❖ At this moment Programmer has to switch the application manually.

Problem Statement

- ❖ Develop a framework that predicts the optimal processor at runtime which has less latency and high throughput for computations at different instances in a heterogeneous environment.

Research Motivation

- ❖ Appropriate selection of optimal processor reduces the overall execution times of the computations.
- ❖ low latency and high throughput can be achieved.
- ❖ It can also prevent starvation of data stream in some instances.
- ❖ The overall performance of the system can be improved like by the branch predictor in CPU hardware.

Objectives and Expected Outcomes

- ❖ A hardware independent framework that predicts optimal processing units using a selection strategy by evaluating some properties given by the programmer related to the task and through obtaining some performance matrices during runtime.
- ❖ It should be able to integrate new computational models defined by the programmer.
- ❖ The solution should adapt to the nature of input computations and avoid them assigned to wrong units.
- ❖ The evaluation process should be asynchronous to avoid latency in the system by its processing.

Literature Review - Selection mechanism

Document	Methods	Key Points
Seamlessly Portable Applications [14]	DLS will look the performance database to analyse the history runs of that particular implementation and compare the time taken to execute with the other implementation for that particular problem size.	Once the platform is selected, it wont switch to other platform during runtime. Only history based runs are selected
Adaptive runtime selection for GPU [17]	Whichever the version that finishes the tasks first will kill the other run	Using resource for redundant calculation.
Toward an Analytical Performance Model [16]	Machine-Learning-based algorithms may achieve high degrees of accuracy, but may also suffer from some drawbacks that limit their applications.	Applicability of such solutions is often limited in production systems. The need for runtime-available parameters in ML solutions leads to overhead for inference at runtime.

Literature Review - Selection mechanism

Document	Methods	Key Points
STARPU A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures [18]	Providing user selectable scheduling mechanism	Once selected, same mechanism is used throughout the run.
Workload Partitioning for Accelerating Applications on Heterogeneous Platforms [19]	Workload partitioning using online and offline profiling	Function nature is not considered. Only data transfer and hardware computational capacity is considered.
Automatic task mapping and heterogeneity-aware fault tolerance [15]	Using performance database to select the accelerator based on previous run.	Won't be effective if there are no previous run.

Literature Review - Experimental Mechanism

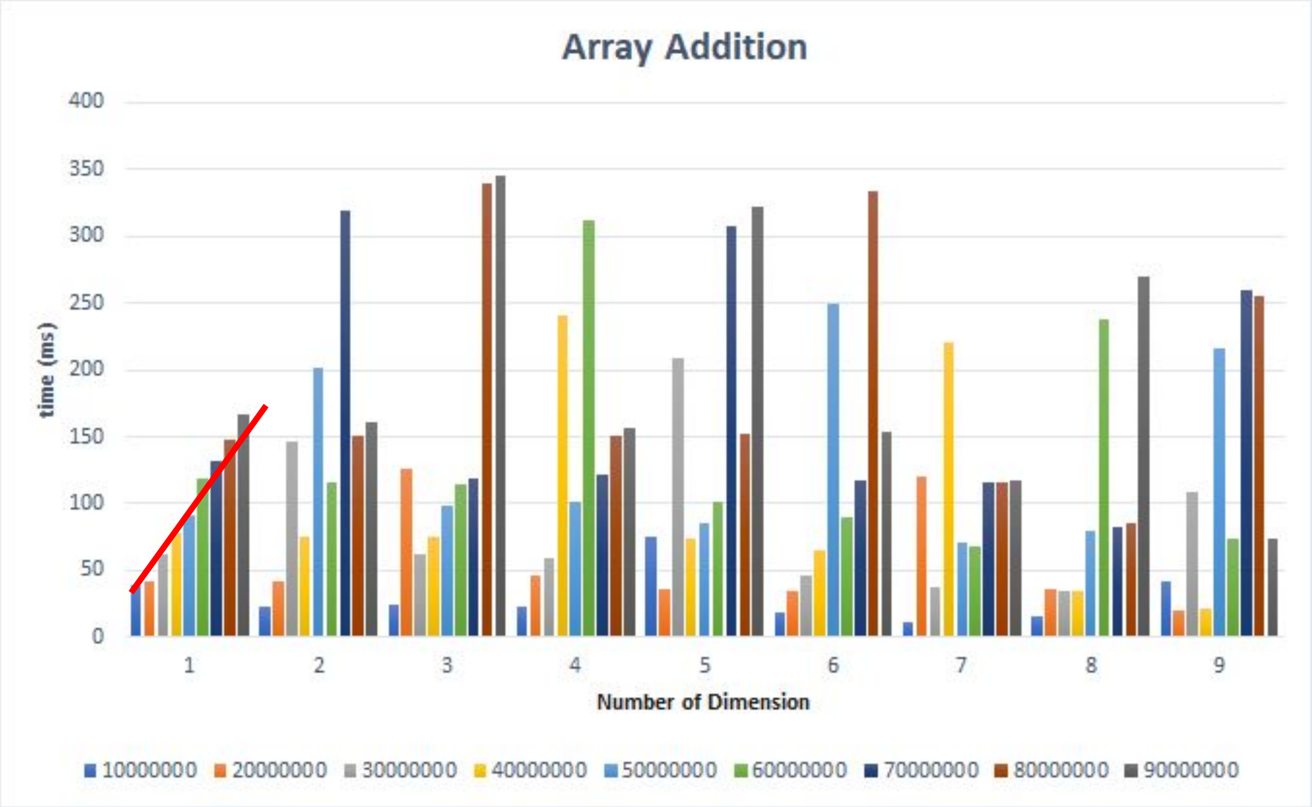
Document	Methods	Key Points
Runtime Coordinated Heterogeneous Tasks in Charm++ [22]	Use Accel framework to select the strategy	Limited to the selection strategies presented in the framework.
Cost-Aware Function Migration in Heterogeneous Systems	The idea is based on online learning of the implementations which assist in guided execution of the best implementation.	This technique is similar to the history based selection.

Methodology

We conducted the experiment in following categories.

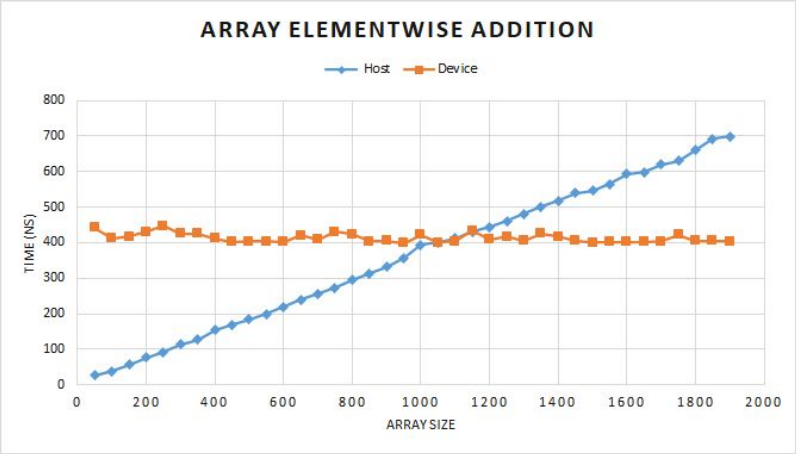
- ❖ Raw GPU Experiment
- ❖ Task Based Experiment
- ❖ Deployment System Experiment
- ❖ Raw Execution Time Based Analysis
- ❖ Nature of input data based analysis

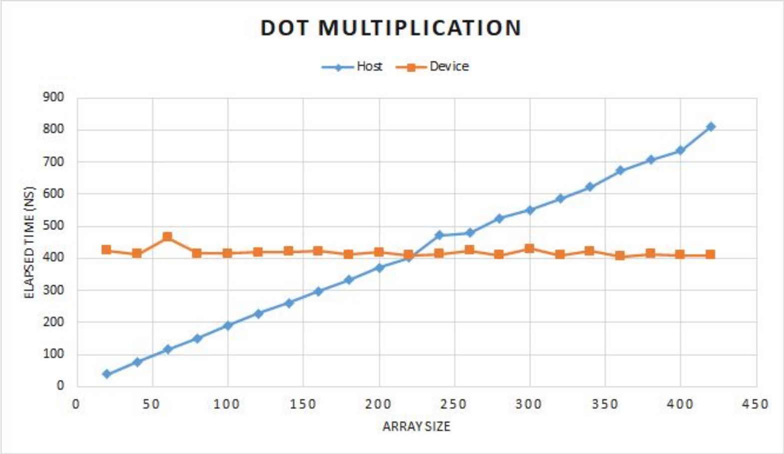
Methodology - Raw GPU Experiment



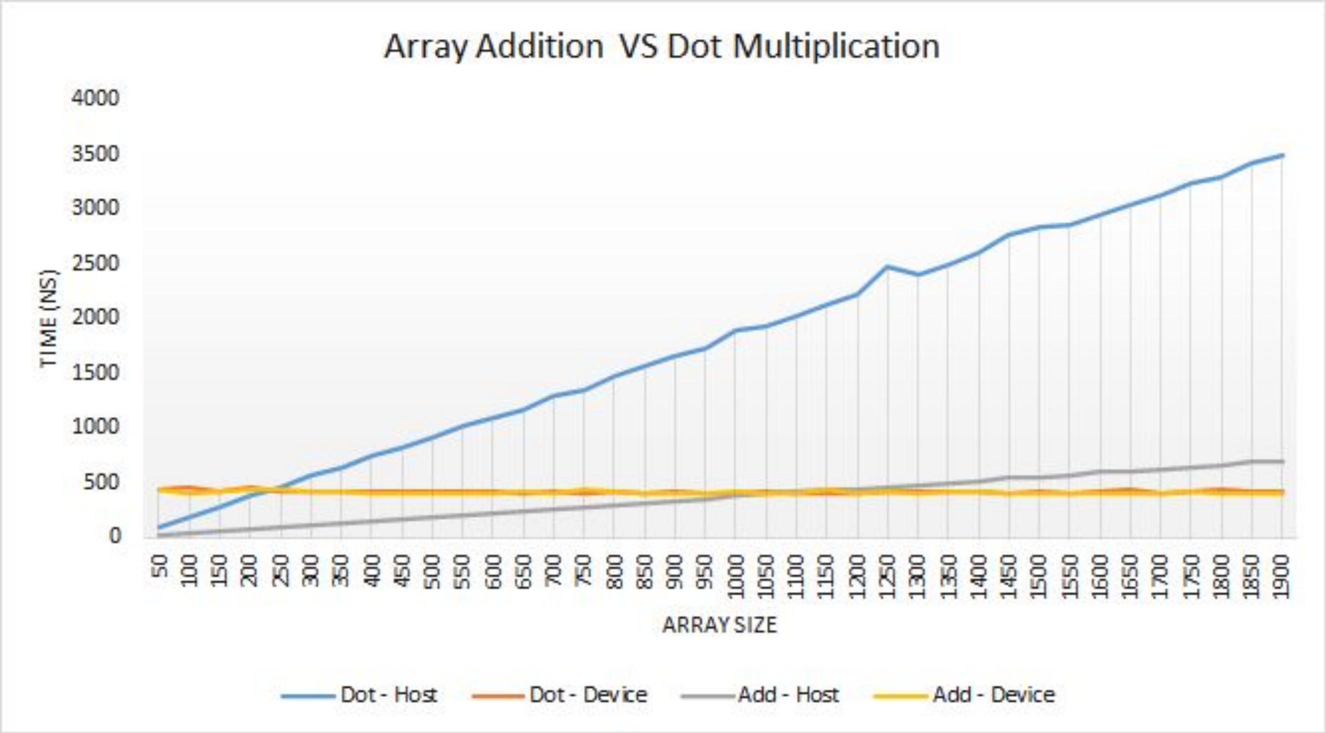
Methodology - Task Based Experiment

We experimented and calculated execution time for array addition and dot multiplication with varying array size on CPU and GPU.





Methodology - Task Based Experiment



❖ CPU to cross the boundary happens soon when the computation become more computationally intensive

Methodology - Task Based Experiment

$$n = \left(\frac{zs}{r\bar{x}}\right)^2$$

$z - standard\ deviation\ for\ required\ confidence\ interval\ from\ normal\ curve$

$\bar{x} - sample\ mean$

$s - sample\ standard\ deviation$

$r - required\ accuracy$

⇨ z = 1.960 (95% confidence interval)

⇨ r = 0.05 (5%)

$$n = \left(\frac{196\ s}{5\ \bar{x}}\right)^2$$

Methodology - Deployment System Experiment

Experimented in multiple machines

Array Size	Host Time	Device Time
100	0.00212042	0.00039633
200	0.00444981	0.00023719
--	--	--
800	0.01605261	0.000225
900	0.0176719	0.00022228
1000	0.01917692	0.0002281
1100	0.02090968	0.00022673
1200	0.02668534	0.00022467
1300	0.02593127	0.00026113

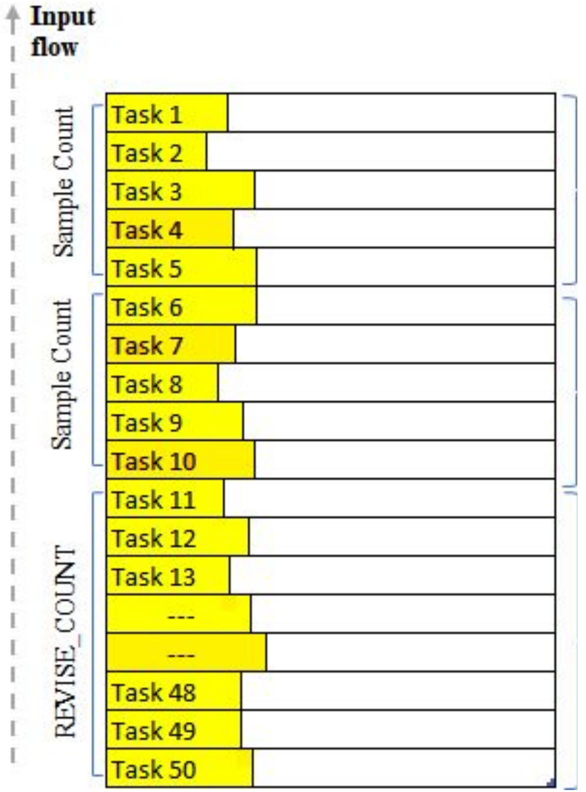
Array Size	Host Time	Device Time
100	0.00032395	0.00247511
200	0.00047535	0.00152941
--	--	--
800	0.00105543	0.00149808
900	0.00123664	0.00152057
1000	0.00199642	0.00166710
1100	0.00270865	0.00209979
1200	0.00233752	0.00320343
1300	0.00300046	0.00151011

Methodology - Raw Execution Time Based Analysis

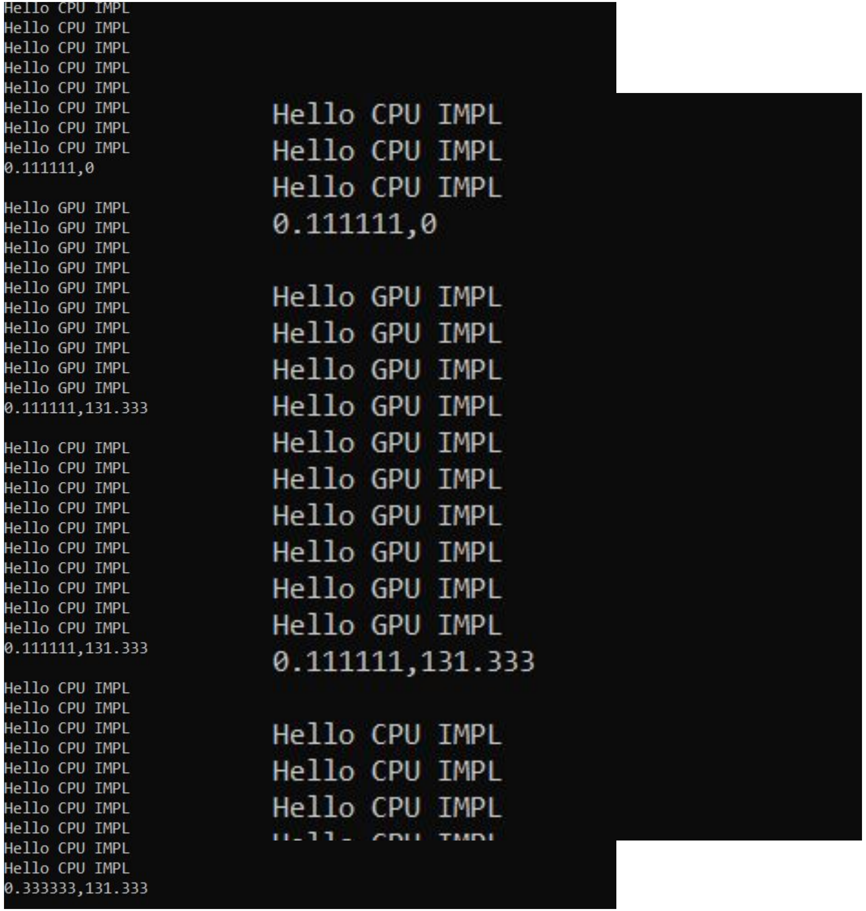
We experimented based on raw execution time after we had implemented a framework.

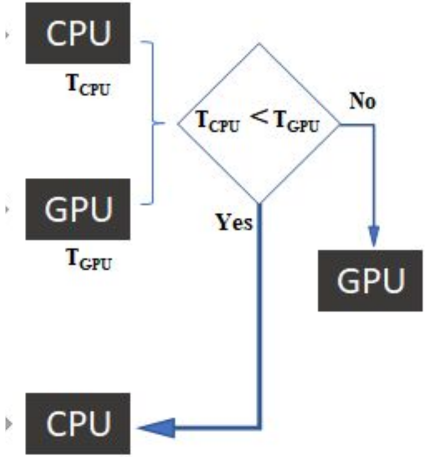
We have followed various approaches.

- ❖ Last N Average Time Comparison
- ❖ Sample Execution Time Algorithm
- ❖ Sample Algorithm with Varying Revise Count



Methodology - Last N Average Time Comparison



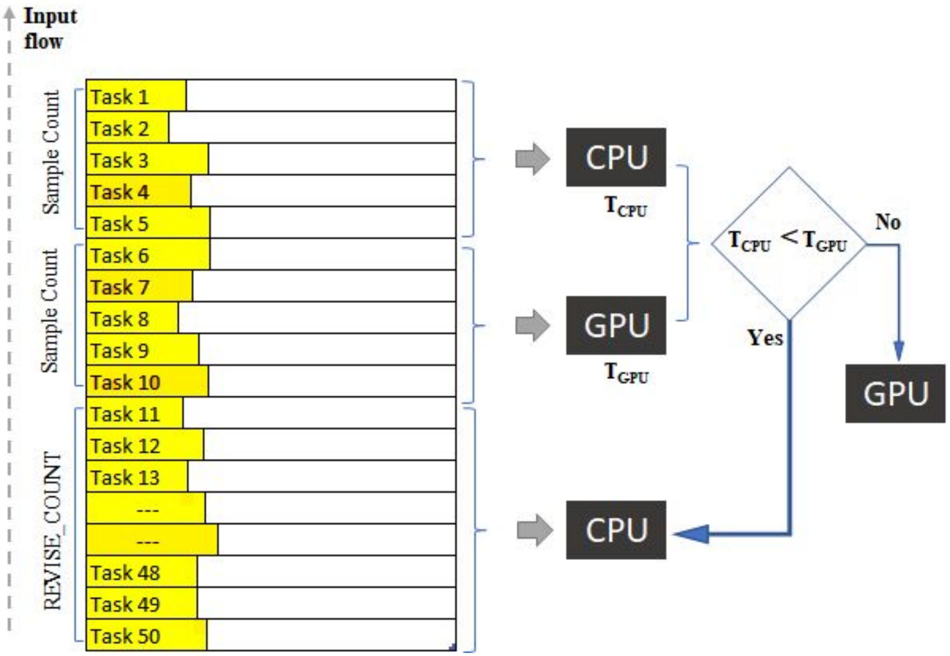


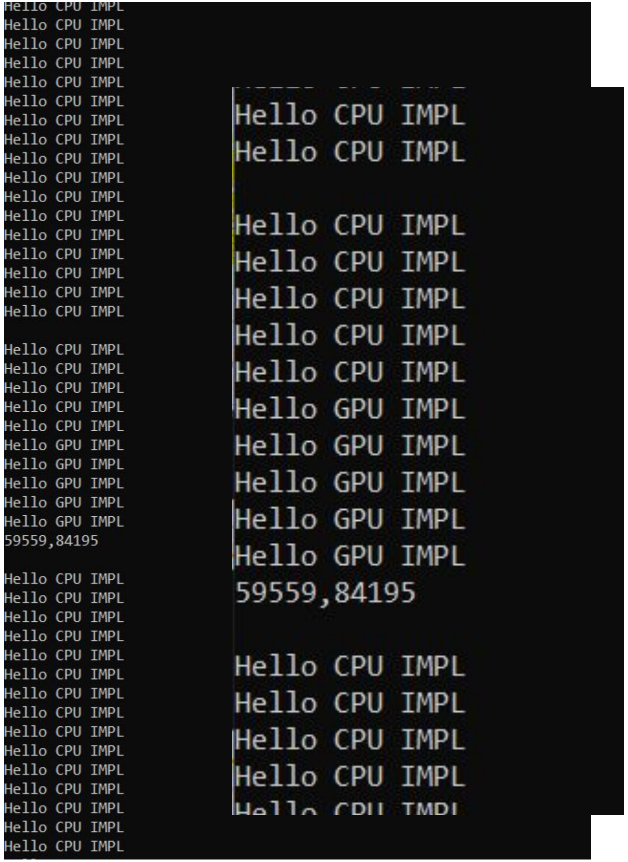
Methodology - Last N Average Time Comparison

Problem :- There may be instances one processor sitting idle if execution time has raised to high value due to some unexpected issues.

- Therefore, the other accelerator continues the execution forever which makes this approach inefficient.

Methodology - Sample Execution Time Algorithm





Methodology - Sample Execution Time Algorithm

Lacks in this approach:

- ❖ It is hard to select the most appropriate REVISE_COUNT value.
- ❖ It is waste to evaluate samples unwantedly if one accelerator specific problems are arriving continuously.
- ❖ Evaluating the samples every fixed count adding overhead to the processing and it recoup the gain over the latency from the algorithm.

Methodology - Sample Algo with Varying Revise Count

```
Hello GPU IMPL
Hello GPU IMPL
Hello GPU IMPL
Hello GPU IMPL
1,3639,29402

Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL

Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello GPU IMPL
Hello GPU IMPL
Hello GPU IMPL
Hello GPU IMPL
Hello GPU IMPL
2,14313,59743

Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
```

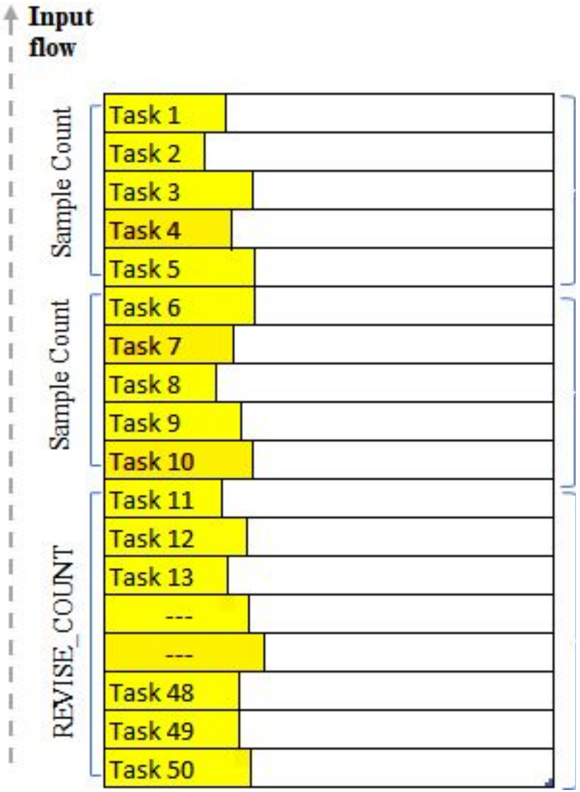

```
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello CPU IMPL
Hello GPU IMPL
Hello GPU IMPL
Hello GPU IMPL
Hello GPU IMPL
Hello GPU IMPL
2,14313,59743

Hello CPU IMPL
Hello CPU IMPL
```

Methodology - Sample Algo with Varying Revise Count

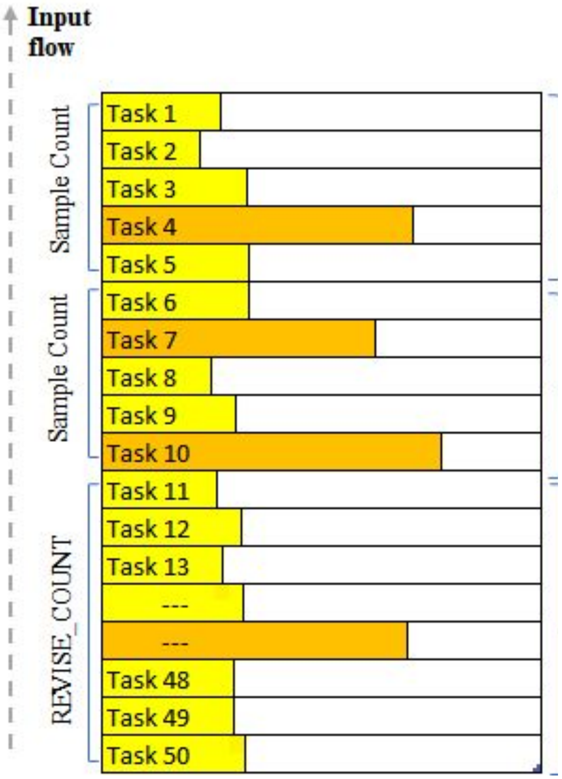
- ❖ Problems in this version of selection algorithm is it cannot tackle with all kind of the nature of the input data.

- ❖ We need to consider the results from the Task Based Experiment with current version of algorithm to determine solution for this issue.

Methodology - Nature of Input Data



Even Input



Odd Input

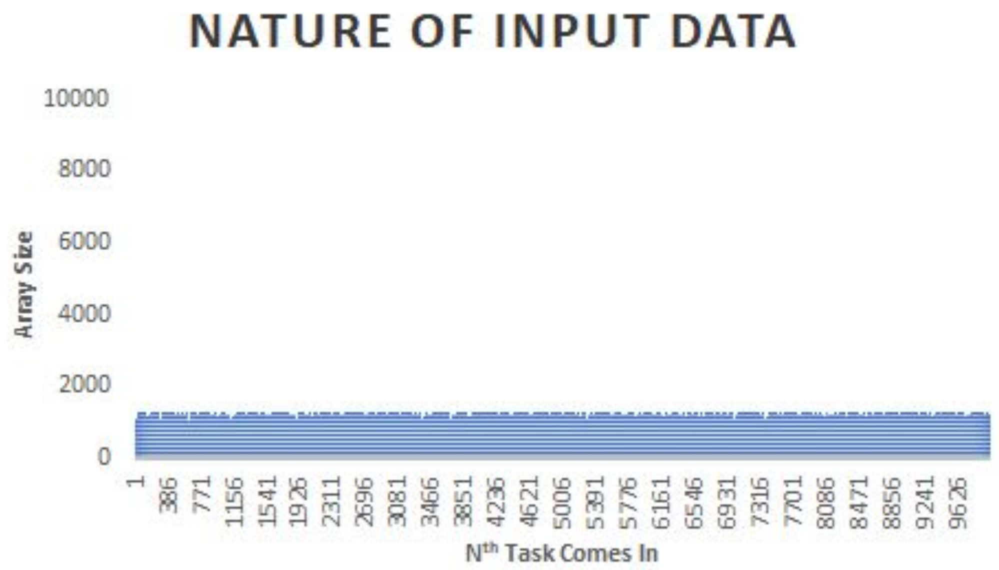
Methodology - Nature of Input Data

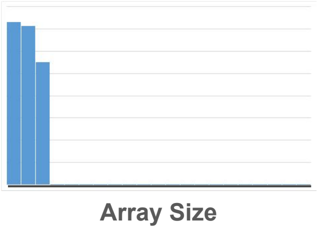
In this analysis, we experimented with 5 nature of input data

1. CPU Specific Input Stream
2. GPU Specific Input Stream
3. Square Aligned Wave Input Stream
4. Binary Aligned Input Stream
5. Odd Input Stream

Methodology

CPU Specific Input Stream



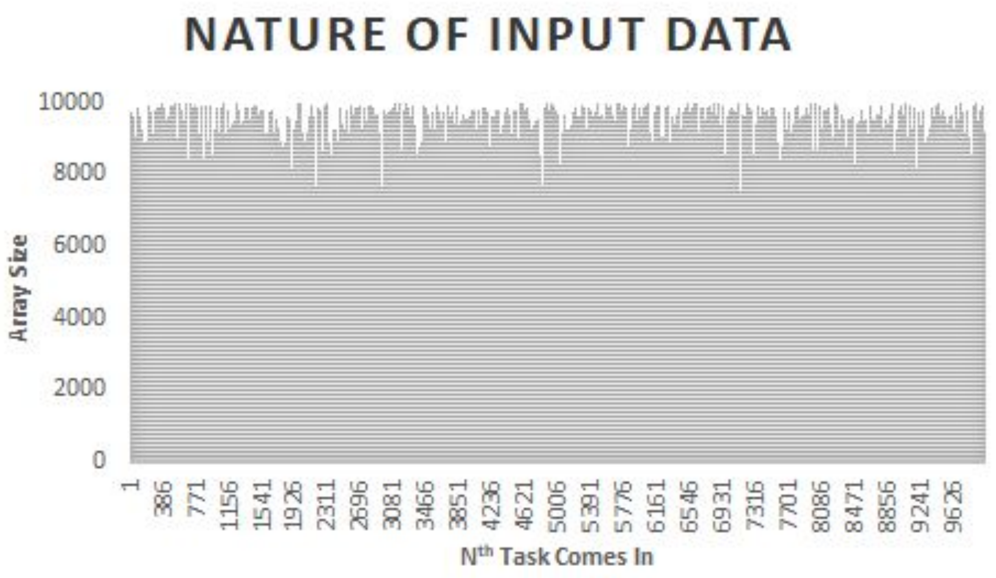


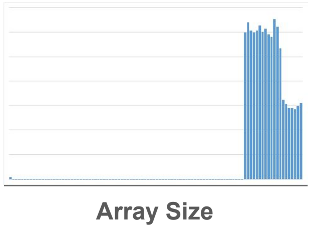
Self Flow Time: 1925 ms

CPU Only Time: 263 ms

GPU Only Time: 34406 ms

GPU Specific Input Stream



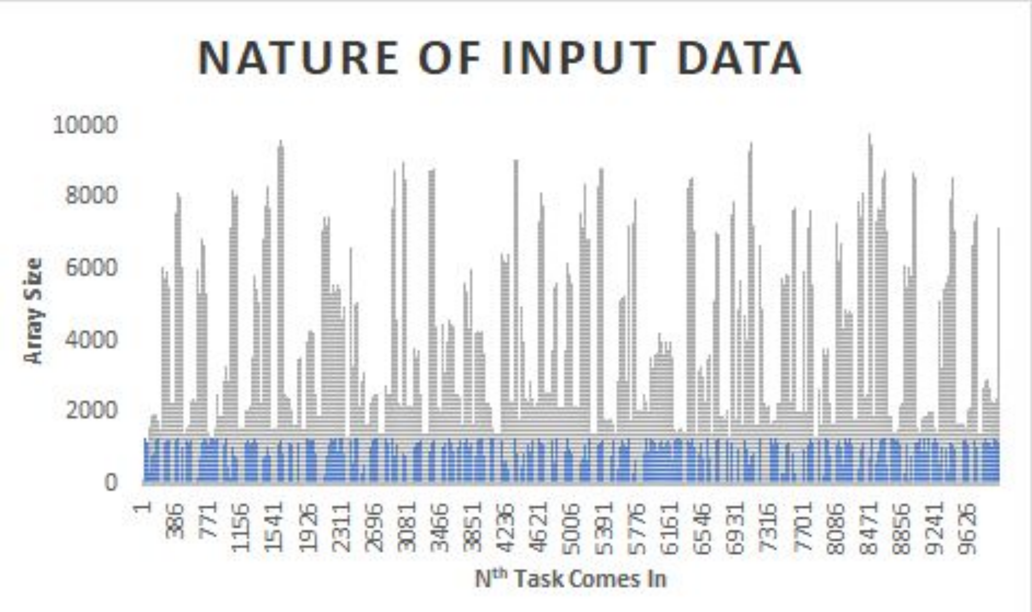


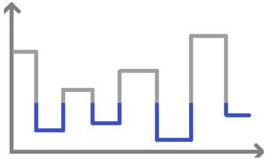
Self Flow Time: 34084 ms

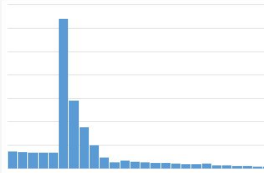
CPU Only Time: 34101 ms

GPU Only Time: 30389 ms

Square Aligned Wave Input Stream





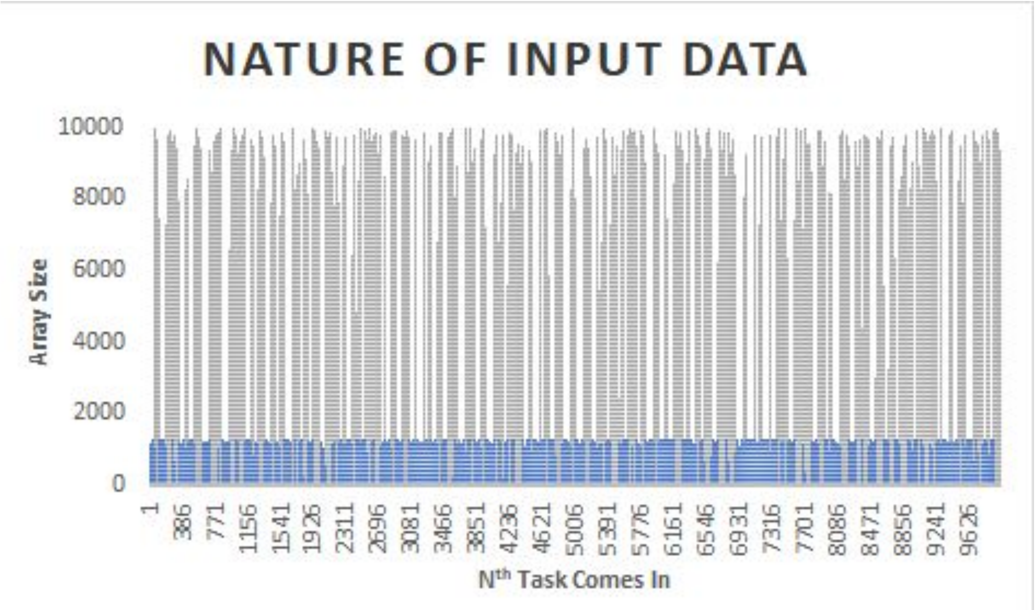


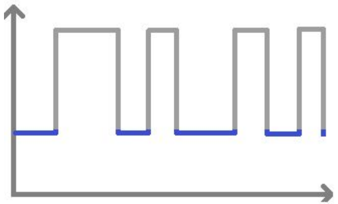
Self Flow Time: 2062 ms

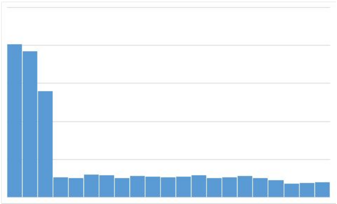
CPU Only Time: 785 ms

GPU Only Time: 35921 ms

Binary Aligned Input Stream





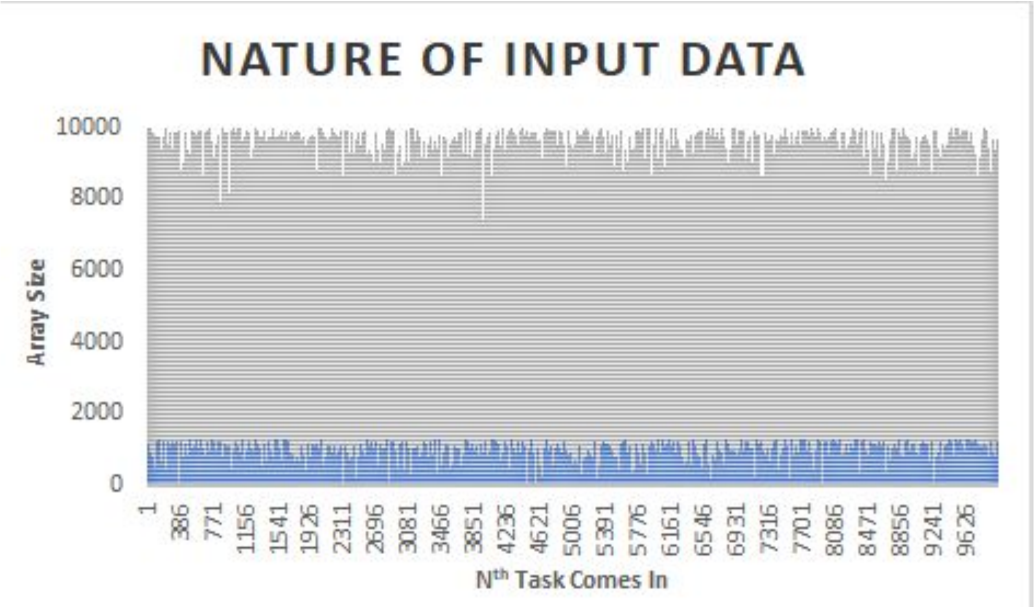


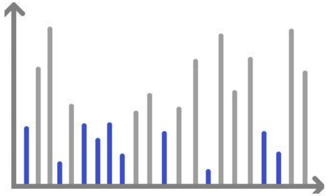
Self Flow Time: 2436 ms

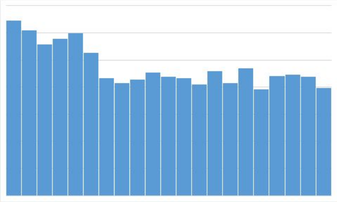
CPU Only Time: 2936 ms

GPU Only Time: 41697 ms

Odd Input Stream







Self Flow Time: 10545 ms

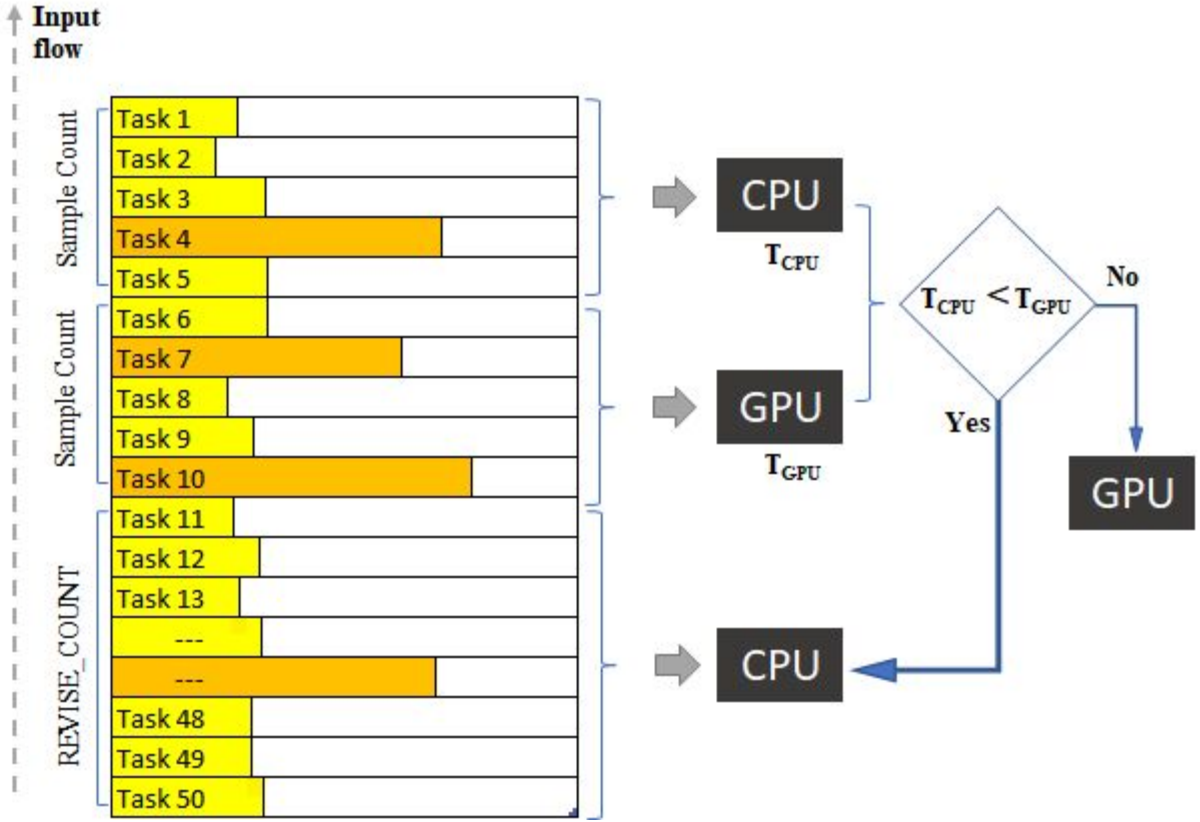
CPU Only Time: 16579 ms

GPU Only Time: 55062 ms

...proposed Solution

- ❖ A hardware independent framework implemented in C++ contains computational models.
- ❖ Programmers will create an object of any of the models with some system specific parameters such as number of cores in CPU, etc.
- ❖ Each time a programmer wants to use the model, he will set the data and call execute() method on the object.
- ❖ The execute() method will invoke the best from two functions implemented for CPU and GPU implicitly.

Selection mechanism



...proposed solution

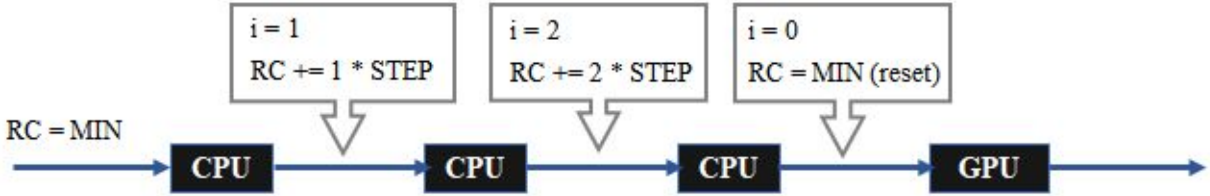
- ❖ Initially, two adjacent disjoint sample sets are executed and evaluated in the processing units and the optimal one set for the rest up to a number, called REVISE_COUNT.

- ❖ After the REVISE_COUNT exceeded, again samples from each evaluated and switch the processing unit if needed.

- ❖ He can manually set to a fixed processing unit for a problem by giving the unit id to the execute method. E.g. execute(1); for CPU.

- ❖ Manual mode will not be evaluated and is there for the policy of implementation reusability.

...proposed solution



- ❖ REVISE_COUNT is incremented if the same unit is selected for the next period but reset to a minimum value if the unit has been switched.
- ❖ A new computational model can be added to the framework with custom implementations.
- ❖ Programmer has to extend a class, "ComputationalModel" and must override two abstract methods, CPUImplementation and GPUImplementation.

Timeline and Future Tasks

Month	Task breakdown	Status
April	● Extracting features ● Prioritizing features ● Set the features into classe	Completed In progress In progress
May	● Group the features for functions ● Implement related algorithm	Completed
June	● Measuring the impacts of the features ● Create models and design of the Library	In progress
July	● Implement and code the Library	In progress
August	● Prepare for mid evaluation	Completed

Timeline and Future Tasks

❖ Optimize the algorithm with varying REVISE_COUNT adapt to the problem stream nature.

❖ Combine the results from the Task Based Experiment with selection algorithm to determine optimal processor.

❖ Make the process asynchronous to reduce the overhead due to the framework calculations.

❖ Complete implementation and evaluate the library.

❖ If time permits, explore more functions, analyze their suitability and REVISE_COUNT relations.

Summary

- Evaluate processing problems in relation to their execution time determining factors.
- A library of functions predicts which processor would offer less execution time.
- The framework switch processor evaluating few samples runtime periodically.
- This solution can be scaled up to other complex computations.
- This research will push the heterogeneous computings into another dimension.

References

[1] 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

[2] 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

[3] S. Brinkmann (2020). "ResearchGate" [Accessed:21 Jan. 2020] [Online].
https://www.researchgate.net/post/Anyone_have_experience_in_programming_CPU_GPU_What_is_the_real_benefit_in_moving_everything_possible_from_CPU_to_GPU_programming

[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].

[5] NVIDIA Corporation, 2020. NVIDIA's Next Generation CUDA Compute Architecture - White Paper. [Accessed 27 March 2020] [online]. https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[6] 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

[7] 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

[8] NVIDIA Organization, 2018. Docs.nvidia.com. [Accessed 27 March 2020] [online]. https://docs.nvidia.com/cuda/archive/9.1/pdf/CUDA_C_Programming_Guide.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] Chikin, A., Amaral, J., Ali, K., Tiotto, E., “Toward an Analytical Performance Model to Select between GPU and CPU Execution”. [online] [Accessed 8 August 2020] https://ieeexplore.ieee.org/document/8778216/

[11] Bakkum, P. and Skadron, K., 2010. Accelerating SQL Database Operations on a GPU with CUDA. [Accessed 27 March 2020] [online]. https://www.cs.virginia.edu/~skadron/Papers/bakkum_sqlite_gpgpu10.pdf

[12] 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

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

[14] 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

[15] 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

[16] 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

[17] 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

[18] 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

[19] 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

[21] 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