CAPITAL UNIVERSITY OF SCIENCE AND TECHNOLOGY, ISLAMABAD

A Load-Balanced Task Scheduler for Heterogeneous Systems based on Machine Learning

by

Asad Hayat

A thesis submitted in partial fulfillment for the degree of Master of Science

in the

Faculty of Computing

Department of Computer Science

2021

I dedicate my dissertation work to my family, teachers, and friends. The special feeling of gratitude to my loving parents for their love, endless support, and encouragement.

CERTIFICATE OF APPROVAL

A Load-Balanced Task Scheduler for Heterogeneous Systems based on Machine Learning

by

Asad Hayat

(MCS183021)

THESIS EXAMINING COMMITTEE

S. No.	Examiner	Name	Organization
(a)	External Examiner	Dr. Abdul Rehman Buzdar	HITEC, Taxila
(b)	Internal Examiner	Dr. M. Shahid Iqbal Malik	CUST, Islamabad
(c)	Supervisor	Dr. Saima Nazir	CUST, Islamabad

Dr. Saima Nazir
Thesis Supervisor
April, 2021

Dr. Nayyer Masood
Head
Dept. of Computer Science
April, 2021

Dr. Muhammad Abdul Qadir
Dean
Faculty of Computing
April, 2021

Author's Declaration

I, **Asad Hayat** hereby state that my MS thesis titled "**A Load-Balanced Task Scheduler for Heterogeneous Systems based on Machine Learning**" is my own work and has not been submitted previously by me for taking any degree from Capital University of Science and Technology, Islamabad or anywhere else in the country/abroad.

At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw my MS Degree.

(Asad Hayat)

Registration No: MCS183021

Plagiarism Undertaking

(Asad Hayat)

Registration No: MCS183021

Acknowledgement

All praise is to ALLAH (S.W.T). The creator and Sustainer of all seen and unseen worlds. First and foremost, I would like to express my gratitude to ALLAH Almighty for his countless blessings upon me to complete this work. Secondly, I would like to express my sincerest thanks to my supervisor **Dr. Saima Nazir** for her valuable guidance and assistance. I pray for her well being and success in the future.

I am highly indebted to my parents and my family, for their support, assistance and encouragement throughout the completion of my degree. They form the most important part of my life after ALLAH (S.W.T). They are the sole source of my being in this world. No words can ever be sufficient for the gratitude I have for my parents and for my family. A special thanks to my sister for their support and encouragement during my studies. I express gratitude to **Dr. Yasir Noman Khalid**, Assistant Professor HITEC University Taxila for his assistance and cooperation throughout the Thesis. I would also like to extend my gratitude to **Mr. Muhammad Asif**, a Ph.D. Scholar at CUST and **Mr. Muhammad Usman**, Lecturer Fast University and a generous friend for his valuable support and guidance.

Finally, I am very thankfull to **Mr. Muhammad Nadeem Nadir**, and **Mr. Baber Naeem** MS students at CUST, who are my research members and with whom I completed this thesis.

I pray to ALLAH (S.W.T) that may He bestow me with true success in all fields in both worlds and shower His blessings upon me for the betterment of all Muslims and the whole Mankind.

Ameen

Asad Hayat

Abstract

The homogeneous system consists of the similar type of multiple processors, whereas the heterogeneous system is equipped with a different type of multiple processes, i.e. Central Processing Units (CPUs), accelerators, and Graphics Processing Units (GPUs). Heterogeneous systems based on CPUs and GPUs are becoming mainstream due to disparate processing and performance capabilities of these multi-core architectures. Mostly CPU is better suited to perform latency-sensitive tasks and incorporate architectural advances such as branch-prediction, out-of-order execution, and super-scalar capabilities. Whereas, many-core GPUs are more suited to perform data-parallel and throughput-sensitive tasks due to the inherent massive multi-threading capabilities. Despite much interest in heterogeneous systems, key scheduling challenges associated with them have not received much attention. Particularly, with highly shared resources having heterogeneous CPU GPU devices, new programs scheduling problems are arising. In such environments, high utilization of resources and overall system execution time are important considerations in addition to the need for scaling a single application. In the heterogeneous computing environment, programmers map the applications only on CPU device or GPU device. However, the default process for device mapping is not able to produce best results. If one resource in the heterogeneous environment is powerful in terms of more computing capability, the scheduling schemes favor the powerful resource. In this scenario, the powerful resources are overloaded while all other resources are under-utilized. This load imbalance problem results in more energy consumption and increased execution time. In this research, a novel load-balanced task scheduler for heterogeneous systems based on machine learning is proposed that distributes workload based on the execution time of the application. The proposed scheduling scheme determine which data parallel applications are like to best utilize a core. We show that predicted execution time is a good scheduling priority function so a parallel data execution time predictor application is developed. The scheduler uses this prediction to schedule tasks. The proposed scheme consist of two phases: 1) A machine learning based execution time prediction. 2) load-balanced scheduling of jobs to achieve the utilization of processing cores and reduction in the overall execution time of jobs. The experimental results on dataset generated from two benchmarks Polybench and AMD shows that the proposed model reduces execution time by 65.63%, increased resource utilization ratio by 93.3%, and throughput by 65.5% in comparison to baseline scheduling schemes.

Contents

Author's Declaration **iv**

Plagiarism Undertaking **v**

Acknowledgement **vi**

Abstract **vii**

List of Figures **xi**

List of Tables **xii**

Abbreviations **xiii**

Symbols **xiv**

1 Introduction **1**

1.1 OpenCL: Heterogeneous Programming Framework 2

1.2 Program Scheduling on Heterogeneous Systems 4

1.3 Heterogeneous Scheduling Issues 5

1.4 Problem Statement 6

1.5 Research Questions 7

1.6 Purpose 7

1.7 Scope 7

1.8 Application 8

1.9 Definition/Explanation 8

1.9.1 CPU 8

1.9.2 GPU 8

1.9.3 OpenCL 9

1.9.4 Machine Learning 9

1.9.5 CUDA 10

1.9.6 Homogeneous Computing 10

1.9.7 Heterogeneous Computing 10

1.9.8 HSA 10
1.9.9 PCA 11
1.9.10 CAP 11
1.9.11 SIMD 11
1.9.12 OpenMP 11
1.10 Dissertation Organization 11

2 Literature Review **13**
2.1 Critical Analysis 21

3 Methodology **22**
3.1 Introduction 22
3.2 Work Stealing Scheduling Strategy 24
3.3 Evaluation Benchmark 27
3.3.1 Mean Absolute Error (MAE) 27
3.3.2 Mean Square Error (MSE) 28
3.3.3 Root Mean Square Error (RMSE) 28
3.3.4 R^2 28
3.3.5 CPU Only 29
3.3.6 GPU Only 29
3.3.7 First Come First Serve 29
3.3.8 Device Suitability 29
3.3.9 Alternate Assignment 29
3.3.10 Input Size 30

4 Results and Discussion **31**
4.1 Introduction 31
4.2 Experimental Setup 32
4.3 Application Data-set 33
4.4 Feature Extraction 36
4.5 Feature Selection 37
4.6 Machine Learning Model Selection 41
4.6.1 Model Training and Testing 44
4.7 Work Stealing Scheduler 49
4.7.1 Work Stealing Performance and Comparison 53
4.8 Performance Discussion 57
4.9 Comparison with State-of-the-art Techniques 58

5 Conclusion and Future Work **60**
5.1 Future Work 61

Bibliography **62**

List of Figures

3.1 Methodology 23
3.2 Work Stealing Strategy 25
3.3 Scheduler Job Pool 27

4.1 Experimental Result of Benchmark Application Execution 34
4.2 Code Feature Extraction 36
4.3 Correlation Analysis of OpenCL Features 39
4.4 Tree-Based Feature Values 40
4.5 Machine Learning Model 42
4.6 Training and Testing Fitness 45
4.7 Actual and Predicted Values 46
4.8 Actual and Predicted Values Fitness 47
4.9 R-Squared Comparison 48
4.10 Mean Absolute Error Comparison 49
4.11 Mean Square Error Comparison 50
4.12 Root Mean Square Error Comparison 50
4.13 Work Stealing Scheduler 53
4.14 Execution Time of Scheduling Heuristics 54
4.15 Resource Utilization of Scheduling Heuristics 55
4.16 Time Improvement of Scheduling Heuristics w-r-t Proposed Scheduler 56
4.17 Throughput of Scheduling Heuristics 57

List of Tables

2.1 Critical Analysis of Literature Review 20

4.1 Device Specification of Prediction Model 32
4.2 Device Specification of Scheduler 33
4.3 Detail of Benchmark Suite and Input Size 35
4.4 Static Code Features 37
4.5 Important Features from the Feature Set 41
4.6 Multiple Linear Regression Model Specification 43
4.7 Random Forest Model Specification 43
4.8 Gradient Boosting Model Specification 44
4.9 Detail of Data-set used for Prediction 45
4.10 Machine Learning Models Specification 48
4.11 OpenCL Jobs with Input Size and Predicted Values 51
4.12 OpenCL jobs with Sorted Difference Time 52
4.13 Comparison of Work Stealing Performance with other Models . . . 59

Abbreviations

AMD	Advanced micro devices
CAP	Coscheduling asymptotic profiling
CPU	Central processing unit
CUDA	Computer unified device architecture
EM	Evaluation metrics
FCFS	First come first serve
FPGA	Field programmable gate array
GB	Gradient boosting
GPU	Graphic processing unit
HDSS	Heterogeneous dynamic self-scheduler
HPC	High performance computing
HSA	Heterogeneous system architecture
IDE	Integrated development environment
ISP	Internet service provided
KNN	K-nearest neighbour
LIFO	Last in first out
LLVM	Low level virtual machine
MAE	Mean absolute error
MLR	Multiple linear regression
MSE	Mean square error
PCA	Principal component analysis

Symbols

C_time	Predicted CPU time
G_time	Predicted GPU time
D_time	Difference of CPU time and GPU time
yi	Actual value of y
$y*$	Predicted value of y
$y-$	Mean value of y
N	Number of jobs

Chapter 1

Introduction

Modern computing systems have mainly two types of processing units, one is the Central Processing Unit (CPU) and the other is Graphics Processing Unit (GPU). Both Central Processing Unit and Graphics Processing Units have significantly different structures because of their application domain and perform different type of task execution. Normally a CPU device executes sequential programs with latency optimization and GPU device executes parallel programs like image processing with throughput optimization [1]. In multi-core architecture similar processing cores or units (CPU's) combine in an integrated circuit to execute a program or multiple programs. The system that has the same type of processor is known to be a homogeneous system. While a system that uses more than one kind of cores or processors is referred to as a heterogeneous system. The heterogeneous system consists of multi-core CPUs and many-core GPUs. The hybrid system of CPU+GPU is mostly used in mobile platforms, desktop environments, supercomputers, and data centers. In heterogeneous systems, the multi-core CPU works as a host program and controls the distribution of parallel workload, memory copying and execution of the programs in Multiple Instruction Multiple Data (MIMD) style while GPU consists of many cores and performs the parallel execution of programs in Single Instruction Multiple Data (SIMD) way. The multi-core architecture is developed as a solution to the issue of power usage, heat dissipation, and transistor density.[2] Heterogeneous System Architecture (HSA) is capable of using multiple processor types like CPUs, GPUs, etc. [3]. HSA is a multi-core system that not only gains efficiency by integrated cores but also by implementing advanced computing capabilities to cope with complex tasks while maintaining energy efficiency. The GPUs are of two type 1) Discrete type 2) Integrated type. Integrated type GPU is built in by the vendors while the Discrete is defined by user according to their need.

The latest applications create a workload of complex specifications due to enormous data collection and massive computing capacity. The Central Processing Unit (CPU) only is not capable of handling these diverse requirements. However, heterogeneous computing is designed to support and enable different processors such as CPU and GPU to be used effectively to address these fast-developing workloads effectively. The utilization of these processors helps and enables new experiences and also maximizes the throughput and reduces the turnaround time. Using the multiple processors gives different possibilities to find at least the finest of them that could succeed at completing a specific job. Some devices are rather not sufficient for some programs while excelling in others. Once we understand that each device has its power, we can effectively select the appropriate device for a specific task. Heterogeneous computing helps in designing multiple processors for working together and enabling unique user experiences.

1.1 OpenCL: Heterogeneous Programming Framework

OpenCL, CUDA, OpenMP, are programming models widely used as a toolkit for mapping jobs on heterogeneous systems. Among them, OpenCL is developed as an industry standard for data-parallel applications for heterogeneous multi-core and many-core architectures. OpenCL applications can be executed on multiple processors like CPU's GPU's FPGA etc. and multiple hardware architectures like AMD, Intel, and Nvidia, etc. because of its compact nature [4]. The OpenCL program consists of two parts. 1) Host program: It is the outer control logic that performs the configuration for a GPU-based application. It executes the serial part of the OpenCL program and is normally executed on the CPU. Only one host program can run on the host. The host program is the only program that does everything, such as memory control and synchronization, while the computing work is done by the kernels. 2) Kernel function: A kernel function is a Program Written in the OpenCL language that allows it to be Compile on Any platform allowing OpenCL for execution. Kernel function executes a parallel part of OpenCL program on CPU, GPU, or any other supported device. The only way the host can call a Program that will run on a computer is through the kernel function. The kernels are executed on the processing devices. The kernels are mapped either one to one, or one to many, it mean kernel program can run on one device and can run on many devices.

The heterogeneous system performs very well if the programs are mapped in an optimized way to their computing devices. Even though OpenCL provides convenient values, its performance would probably vary over distinctive portions of the heterogeneous system. The cause would be that one program will have a low execution period on the GPU rather than on the CPU, if allocated to the GPU while achieving low execution time on CPU, the efficiency of another program could decrease significantly. Programmers usually allocate jobs either to a CPU device or GPU device, so other processing device stays idle. For instance, if all tasks are allocated to a GPU, it causes the CPU to idle and waits for the GPU to complete the assigned tasks. In this research, we are using OpenCL due to its portability and having wide range of supported computing devices. OpenCL is an open platform specifically for parallel programming that produces highly portable code, this means that different platforms use the same code. Used by OpenCL, the model consists of a host and many devices. The host maps to the main program control core, For example, if the program is running on a processor, it maps to the core that begins and merges the threads, and it maps to the system CPU if it is running on a GPU. As a consequence, the modules will either map to the rest of the cores on the CPU or to the cores of the GPU. The memory model of OpenCL consists of two primary types of memory: host memory and device memory. The host memory is the memory accessed by the host, and the device memory is the memory accessed by the processing devices. In contrast, four memory regions of the device memory is based on what type of data it is, the kernels will distribute data, and which kernels need access to it. In all work groups, two global memory regions can be accessed by anyone. The first one is global memory. In this memory area provides access to all work-items in all work-groups to read/write and the second one is constant memory in this type of memory data remains stable throughout execution time. Device can read data from it but the host can allocate data in constant memory. There is a memory area which is independent region of each kernel that is a private memory. Finally, there is a local memory that acts as a shared memory for devices in the same work-group, ensuring it can be accessed by all devices in the same work-group.

1.2 Program Scheduling on Heterogeneous Systems

There are multiple scheduling techniques used and applications designed for the mapping of jobs to CPU and GPU in the heterogeneous system [5]. The scheduler determines a specific data-parallel job should be allocated to which executing device i.e. CPU or GPU. The proposed applications are suitable if the level of tasks carried out or the data to be processed is identified before [6]. Few researchers map jobs on devices at run-time [7, 8]. The advantage of scheduling tasks on run-time is that the decision of job mapping is more efficient and can be adjusted at run-time but it increases the complexity and higher the scheduling overhead. Researchers [9, 10] Splits the code and maps each part on a suitable computing device. Splitting the code requires high programming skills. Static and dynamic code features are widely used to characterize a program application [2, 3, 5, 11, 12]. Static code features like number of multiplication, number of float data types, number of functions, etc. are extracted at compilation time. Dynamic features include input workload. The code features consist of several instructions.

A data structure tree of the program is created at compilation time using clang and LLVM compiler [13]. The tree provides information about the behavior of

applications i.e. number of operations in the application, type of operations in the application, number of code blocks in an application [3]. The feature vector is formed which consists of all feature values.
The feature vector is provided as input to a machine learning-based predictive model. The predictive model is trained on the provided feature vector. Important features from the feature vector are selected based on their contribution to the output. Using the reduced features set for predictive model training causes a reduction in the issue of over fitting, improves the accuracy, and also causes decrease in the training time of the model.

1.3 Heterogeneous Scheduling Issues

Task scheduling of a processor is a tough assignment. It becomes more difficult when heterogeneous systems are involved in task execution. Programmers must write the code in such a manner that automatically maps the jobs to their executable device. Programmers use the default scheduling strategy i.e. assigning the serial portion of the program (host program) to CPU device and parallel portion of the program i.e. kernel function to GPU device for execution [10]. This causes the wastage of resources as most of the computation is done through GPU and the CPU waits for the completion of the execution of the application on GPU.
Although OpenCL can map jobs on both CPU and GPU. This is the wastage of the main resource (CPU) of the system by not performing any task. However, some researchers addressed the problem and proposed some scheduling mechanisms. Most of the scheduling mechanisms schedule a single data-parallel program [14] that depends on code splitting and requires application profiling.
The longer execution time of applications is also the main concern while executing data-parallel applications. When only GPU executes jobs from the job pool, kernels take a long execution time as the CPU is not taking part in application execution. The overall execution time of the job pool can be reduced by assigning jobs to both CPU and GPU simultaneously.
As CPU only performs the management of assignment of jobs to GPU and does not take part in the actual execution of programs, this causes load imbalance between

the computing devices [15]. A scheduler is required that can schedule programs to both CPU and GPU device in a load-balanced manner so that both the computing devices (CPU and GPU) can complete the processing at the same time. It will reduce the energy consumption and heat dissipation as well as the overall execution time of the job pool. Researchers are trying to find new scheduling techniques and applications and optimize existing ones to reduce the overall execution time of jobs, maximize throughput and device utilization. In this research thesis, we are designing a task scheduler that maps a pool of jobs on a heterogeneous system in a load-balanced manner through a machine learning model, reducing the overall execution time of jobs by considering the throughput and device utilization. In the literature, most scheduling applications in heterogeneous systems are evaluated. However, the following are some issues:

- Most of the scheduling techniques require code splitting. They split the program to execute on CPU and GPU. This code-splitting results in additional time overhead.
- Some researchers map a single job in a heterogeneous environment which causes time overhead.
- To the best of our knowledge, no prior work attempts to do a load-balanced task scheduling of the job pool of the heterogeneous system using a machine learning approach.

1.4 Problem Statement

In a heterogeneous environment, the vast majority of the literature evaluated scheduling tasks by using the data-parallel application code approach. Researchers are trying to map jobs in a heterogeneous environment where they consider the processing capability of the device, suitability of application, speedup of a job on CPU GPU, etc. They are trying to map a suitable job on a suitable core means

CPU-suitable job on CPU and GPU-suitable job on GPU and does not consider the execution time of jobs.

1.5 Research Questions

The critical analysis of the literature survey has led to the following research gaps, which have been focused on in this thesis:

1. How to analyze optimization techniques for execution time predictors through machine learning?
    (a) Which set of features play an important role to predict data-parallel application execution time?
2. How to design and develop a load-balanced scheduler to achieve minimal execution time, maximal throughput, and improved resource utilization?

1.6 Purpose

This research aims to propose a load-balanced task scheduler through machine learning to map a pool of jobs to CPU and GPU in a heterogeneous system reducing the overall execution time of job pool, considering the utilization of devices.

1.7 Scope

In this research work, a load-balanced task scheduler through machine learning for heterogeneous systems is proposed. The scope of this thesis is to elaborate on existing scheduling techniques of heterogeneous systems which are based on code features, device suitability, Speedup of devices, machine learning, etc. This research will design a load-balanced task scheduler for a heterogeneous system reducing the overall execution time of the job pool.

1.8 Application

The research will assist the user to improve the overall performance in terms of execution time, throughput, and resource utilization.

1.9 Definition/Explanation

1.9.1 CPU

Central Processing Unit is the main component of the computing system which processes the instructions. The computer takes data as input the CPU process the input data and the computer system give information from the data processed by the CPU as information. The CPU runs the Operating system and other computer applications. Each CPU in the computer system consists of at least one processor that performs the actual calculations.
The early CPUs have only one processor, today CPU consist of many core and multi core processing cores. A CPU having two cores is called dual core and CPU having four cores is called quad core. The modern computer systems have more than one CPU and number of processing cores. In the CPU cores, each core have their own ALU registers, and cache. CPU is suitable for executing serial instructions.

1.9.2 GPU

Graphic Processing Unit is the processing device made to execute and process the graphics operations. The GPU can performs the calculations of both 2 dimension array, 3 dimension array, graphics processing, video games, and parallel programs execution. The first desktop GPU was introduced by Nvidia in 1999 named GeForce 256. The GeForce 256 can process 10 million polygons per second. The GPUs are friendly in use and programmers uses the GPUs in their programs. Modern Operating systems and programs have the support of general purpose graphic processing units and many programmers uses GPU for non graphics calculations as well that increases the overall performance of the computing systems. In GPUs, instead of CPUs, more simple cores are used, but GPU cores are a little complicated. The only feature that these small cores compromise is the probability of execution out of sequence and branch estimation.

1.9.3 OpenCL

Open Computing Language is developed as an industry standard for data-parallel applications for heterogeneous multi-core and many-core architectures. OpenCL applications can be executed on multiple processors like CPU's GPU's FPGA etc. and multiple hardware architectures like AMD, Intel, and Nvidia. OpenCL distribute the load of computing on different processing units that causes an increase in the efficiency and performance of the programs. At first the GPUs were only used for graphic processing but after the OpenCL came to market, the GPUs now also perform the non graphics processing.

1.9.4 Machine Learning

Machine learning is the process to use statistics for finding the patterns in the data. Machine learning (ML) is widely used for prediction purposes. In the sense of machine learning, the learning refer to automatic method of looking for appropriate representations of data.
There are two types of machine learning. One is supervised machine learning and other is unsupervised machine learning. Supervised learning is type of machine learning in which the output class/class labels are provided to a classifier for training and testing the model. While unsupervised learning does not have output class. Clustering is an example of unsupervised machine learning. Supervised learning have further two types: classification and regression. Classification is type of supervised learning in which the output class label is categorical (discrete) while the regression have numerical (continuous) output class label.

Machine learning refer to finding patterns present in data not to create patterns in the input data

1.9.5 CUDA

Compute Unified Device Architecture is a programming model that is used for mapping program codes on GPUs. It is built in C/C++ programming language for only Nvidia architecture. CUDA is not as compatible as OpenCL.

1.9.6 Homogeneous Computing

Homogeneous computing is multi-core architecture in which similar processing cores (CPU's) are combined in an integrated circuit and execute a program or multiple programs.

1.9.7 Heterogeneous Computing

Heterogeneous computing is that which uses more than one kind of cores or processors. The heterogeneous system consists of multi-core CPUs and many-core GPUs. In heterogeneous systems, CPU works as a host program and controls the distribution of parallel workload and memory copying while GPU and performs the parallel execution of programs

1.9.8 HSA

Heterogeneous System Architecture (HSA) is a cross-vendor series of requirements that permit the convergence of CPU and GPU on the same bus, with shared memory and tasks. It is capable of using multiple processor types like CPUs, GPUs. HSA is a multi-core system that not only gains efficiency by integrated cores but also by implementing advanced computing capabilities to cope with complex tasks.

1.9.9 PCA

Principal Component Analysis is an unsupervised machine learning technique, used for the reduction of dimensions in a data-set.

1.9.10 CAP

Co-scheduling Asymptotic Profiling split dynamically workload of a task to CPU and GPU, and predict the workload for next partition through profiling method. CAP makes a balanced workload distribution between CPU and GPU with few iterations.

1.9.11 SIMD

Single Instruction Multiple Data is system that perform or execute a single instruction for multiple data. SIMD perform simultaneously the same task operation on multiple data. while MIMD, Multiple Instruction Multiple Data is system that execute different multiple instructions with multiple data. In MIMD manner multiple tasks are performed with different data simultaneously.

1.9.12 OpenMP

Open Multi-Processing is an Application Program Interface (API), used by developers for parallel applications with shared memory. OpenMP supports C/C++ and FORTRAN on different architectures.

1.10 Dissertation Organization

The rest of the thesis document is divided into the following sections. In Chapter 2, we present comprehensive literature review related to job scheduling in heterogeneous environment and also a critical analysis of these research techniques.

Chapter 3, presents the proposed methodology to answer the research questions, identified in this study. Chapter 3 also presents the details of the proposed system architecture and performance evaluation metrics. Chapter 4 encompasses the details regarding implementation of the proposed load balanced job scheduler with the detail of experimentation performed on a large set of application. Each experiment is explained in detail and evaluation is performed comprehensively. The Chapter 5 concludes the research work and suggested the future directions.

Chapter 2

Literature Review

Task scheduling is a non-trivial problem that involves efficient mapping of jobs to the processors such that the total execution time of applications is minimized. When there is a heterogeneous system in which each processing unit has a sustainable range of features, the scheduling becomes more complicated. Heterogeneous systems focus on Central Processing Units (CPU) and Graphic Processing Units (GPU) and are becoming effective due to the varying computing and performance capabilities of these multi-core architectures. In most instances, CPUs are best suited to execute latency-sensitive tasks and implement architectural innovations such as branch prediction, out-of-order execution, and super-scalar capabilities [13]. However many-core GPUs are more suitable for data-parallel and throughput sensitive tasks, due to the vast multi-threading capabilities that are intrinsic [5]. There is a small range of strong and complex cores in the CPU so the effective implementation of numerous types of applications is widespread, whereas the GPU comprises a significant number of simpler cores specialized primarily in the execution of data-parallel portions of the program. Therefore, it can also be known to efficiently map computation to processors when designing the heterogeneity of computational systems. Programmers usually associate roles at either a CPU device or GPU device due to which other devices stay idle e.g. if tasks are allocated to a GPU, it causes the CPU to remain idle for just waiting to complete the assigned tasks.

Various scholars have suggested techniques for scheduling heterogeneous applications [5, 9, 16–18]. Kaleem et al., [9] Becchi et al., [16] and Lee et al., [17] split program code between CPU device and GPU device. Boyer et al., [15] and Munshi [19] map a single program to either CPU or GPU device while Tsog et al., [1] Wen et al., [2] and Khalid et al., [13] schedules a pool of jobs to CPU device and GPU device. Predictive modeling based on machine learning is also an effective technique for optimizing parallel programs. Several machine learning classifiers are trained from training data and have an adaptive behavior for varying platforms [12, 20, 21]. Some scheduling techniques [4, 22–24] create load unbalancing between CPU and GPU while scheduling because CPU only manages the kernel execution and does not take part in actual computation. The CPU waits until the GPU completes their execution and that wait is not desirable.
Ideally, a scheduler is required that can schedule data-parallel programs to CPU and GPU in such a way that all processing devices complete their processing at the same time. In this way, the overall execution time of the job pool will be reduced, and also the energy intake and heat dissipation are decreased because of reducing the idling state of processors. In a heterogeneous environment, the optimum choice of devices is a key for all scheduler schemes.

According to Lee et al., [17] programmers mostly consider CPU for sequential tasks and GPU for parallel tasks which causes load imbalances and wastage of computing power. Lee et al., [17] introduced Maat, a library by which programmers build a parallel version of a kernel program that selects a load balancing method and run the program on all available resources. This method utilizes the same kernel code and needs no effort for extra programming.

Huchant et al., [25] automatically compiles and executes only single OpenCL programs on CPU-GPU considering issues of communications, load-balancing, and load-variations. The technique consists of 2 approaches, Static approach and dynamic approach. In the static phase kernel code is partitioned and mapped to different processing devices and the execution time of each kernel is noted. In the dynamic phase, Partitioned kernel queuing is adapted to achieve optimized.

This technique only maps a single kernel program while our approach schedules multiple programs from a job pool.

According to Albayrak et al., [26] different kernels have different characteristics in a multi-application environment. Some kernels execute faster on GPU while some refer to run on CPU, so there is a need to map kernels to their proper devices to improve overall performance. The proposed method is profiling based and profile execution time and data dependencies. A greedy algorithm is used for scheduling kernels to CPU and GPU.

In a study Belviranli et al., [7] Proposed Heterogeneous Dynamic Self-Scheduler HDSS a scheduling mechanism that divides the workload among processing devices. HDSS improves the execution time of the kernel. It has two phases, the profiling phase, and the adaptive phase. In the profiling phase, each processing unit has assigned some loop operations to evaluate the computation power while in the adaptive phase the remaining loop operation is assigned on the base of processing speed. Both profiling-phase and adaptive-phase help in load balancing at CPU-GPU devices. The proposed scheduler does not split the jobs.

According to Choi et al., [8] Selection of devices for executing a data-parallel program is a critical factor in determining the application performance. The researcher estimated the execution time to determine application scheduling on processing devices. The model is trained through the execution history of an application. It also maps the new jobs to devices that have finished their job earlier. The finish time is found through the total time of execution of the application and execution time of the currently executing application. In contrast, the proposed model predicts the execution time and also reduces the overall execution time of jobs.

Grewe and O'Boyle [14] proposed a static feature and predictive modeling based program which is designed to partition OpenCL programs on CPU-GPU systems. Machine learning techniques like SVM are used for partitioning the code features automatically. The values of code features are normalized and computed as the number of work-items divided by data transfer size multiplied by total operations in program code. Principal Component Analysis (PCA) is applied for dimensionality reduction. The model is evaluated on GPU-friendly, CPU-friendly, and other benchmarks while the proposed model predicts the execution time of jobs and schedules them to reduce the overall execution time of the job pool.

In the study of Smart Multi-Task Scheduling for OpenCL Programs on CPU/GPU Heterogeneous Platforms Wen et al., [2] Schedules multiple kernels from multiple programs on CPU and GPU. It determines which device best utilizes the kernel at run-time. The kernel speedup is predicted on the base of static code features and also considers the input data size for scheduling of jobs on CPU GPU. It also Extracts features through clang and LLVM and uses the SVM machine learning classifier for the prediction of suitable devices In contrast the proposed model predicts the execution time of data-parallel applications.

Wen and O'Boyle [20] presents a run-time model that schedules the tasks on a suitable CPU GPU system. The model predicts at run-time whether to merge or schedule separately the OpenCL programs in a job pool using a decision tree machine learning model. The model is based on both static code feature and dynamic code features. The model first separates the kernel to CPU device and GPU device based on estimated device affinity and then determines whether to merge or not the kernels. The merging of programs is done through the JIT compiler and scheduling by a thin run-time layer.

Khalid et al., [13] schedules the given task in a heterogeneous system in a load-balanced manner considering job requirements, device suitability, and also performance predicted on a processor. CPU suitable and GPU suitable jobs are combined in a pool of jobs based on the suitability of the device and sorted on the base of predicted speedup. Load balancing mechanism is gained on the basis of job processing requirements and device computation capabilities. Lower execution time maximum throughput, higher device utilization is achieved through device suitability and mapping of jobs in a load-balancing manner. Users allocate OpenCL programs, and the computational assessment module checks the computational requirement through computational complexity and data size. The Kernel code features extractor extracts the Code features from the OpenCL job and provides a device suitability classifier to classify and label according to device suitability. The OpenCL programs along with the code feature extracted and input data size are provided to the speedup predictor component to predict speedup concerning other devices. The application then sorts the CPU-GPU job pool on the basis of device suitability where CPU suitable jobs pool is arranged in descending order on the basis of speedup and GPU suitable jobs pool is arranged in ascending order. After sorting, both job pools are combined for scheduling. E-OSched maps the jobs to CPU-GPU jobs. The top jobs from the job pool are mapped to CPU while jobs at the bottom are mapped to GPU.

Heterogeneous architecture has multi-core CPUs as well as many-core GPUs and performs parallel execution [3]. Task scheduling in heterogeneous architecture is a challenging job. An OpenCL framework is designed to perform task execution on heterogeneous architecture. Many features affect the scheduling of a task. Tasks like out-of-order execution, branch prediction, etc. are suitable for CPU while parallel execution of tasks is more suitable on GPU. The main theme of this research is to map OpenCL applications based on process capability and application/device suitability and is achieved through a machine learning classifier that predicts the computational compatibility of processors. LLVM based analyzer is used for feature extraction and tree-based method for classifier selection.

Resource Aware Load Balancer for Heterogeneous Cluster [5] RALB-HC is a supervised machine learning-based approach that distributes the workload in multi-node heterogeneous computing environments based on the computing capabilities of resources and needs of applications computing. The model considers the device suitability, the expected speedup, and the load balancing for job mapping in heterogeneous environments. The RALB-HC technique works in 2 phases: 1) Mapping of jobs is based on available resources. 2) Load balancing for a higher resource utilization ratio. It automates decision about jobs for a specific computing device. Synthetic and Google-like workloads are produced using AMD and Polybench benchmark, For testing performance. RALB-HC reduces the execution time, increases the utilization of resources, and improves the throughput.

The work of Belviranli et al., [7] is based on distributing OpenCL kernels among CPU and GPU in heterogeneous architecture. Supervised machine learning algorithm split a single task into multiple kernels and analyze the static program feature and predict a ration for distribution of kernels. The architecture also provides state information about the system. The algorithm assigns 0 or 1 to devices and maps them in the heterogeneous platform. In this architecture partition can be done through machine learning as well a user can also do it manually. The architecture takes as input the static partitioned which is the relationship of static program feature and a specific split ratio and gives information about split ration as output. For evaluation, Weka toolkit is used.

OpenCL can execute a program on multiple devices like CPU, GPU, FPGA, etc. [11]. Moren and G¨ohringer estimates the best application speed-up for each device using a machine learning classifier. It is based on dynamic code-feature extraction through the LLVM framework. Prediction is performed on multiple parallel applications. The accuracy of the model depends on prediction mechanisms and scenarios. The limitations of this approach are it is platform-dependent and does not work properly on complex code. It only supports simple arithmetic operations like addition, subtraction, multiplication, etc., and is not valid for complex operations. It also takes more time as compared to static code features.

Machine learning classifiers do not perform well in high-density computing requirements because of the execution of instructions in a sequential way [27]. A machine learning model is designed for which the GPUs are used for training purposes and FPGAs are used as inference models. A model converter is also defined between the training model and inference model. The approach is evaluated through two use cases. First through constitutional neural networks and second through deep neural network regression.

Programmers write the code in a way to distribute the workload on processing devices and also distribute the data among different devices [28]. Mostly programmers maps suitable application to suitable device that causes load imbalance and under-utilization of the processing devices. They try to map CPU suitable job on CPU device and GPU suitable job on GPU device. To remove the burden of data management and device connectivity from programmers, EngineCL is extended with the help of OpenCL and also supports the FPGA devices. The proposed model performs management of data as well overlapping of commands and performance is improved by 96%. It also gains 36% of energy-delay. Six different benchmarks are used for experimentation purposes and executed for each benchmark application with different data input size. Results are obtained with five methods i.e. static, dynamic, HGuided, Performance, and Energy Consumption. Performance up to 96% increase.

Simhash [29] is a commonly used tool capable of attributing a bit string existence to a word so that identical texts have identities that are similar. A real-time solution for a simhash measurement in OpenCL is proposed in this research and also illustrates how multi-core CPUs, GPUs, and FPGAs can make use of it. Simhash is implemented on three HPC platforms: Multi-core CPU, GPU, and FPGA. Random text generation methodology is used for evaluation. The experiments are performed on Multi-core CPU, GPU, and FPGA.

Wang et al., [23] designed Co-scheduling Asymptotic Profiling that is a two-phase scheduling method designed for heterogeneous computing. A static partitioning approach is used in the first step to assign a small amount of workload fairly to both CPU and GPU. The execution time is measured for the allocated workload. Based on the previous execution time of workload, the amount of work is doubled on a faster device. The scheduling is carried out with the elevated workload before the difference between the present and former executions are fewer than the predefined threshold. In the second step according to computing units, the remaining workload is split as per the first step of the sampling.

Table 2.1: Critical Analysis of Literature Review

Ref	App	Feature	Kernel	Benchmark	Performance	ML
[14] Grew et al. 2011	47	13	Single	Parboil N-vidia	1.57% speedup	Support Vector Machine
[2] Wen et al. 2014	35	16	Multi	AMD Nvidia Parboil Polybench	1.5 time reduction in TAT	Support Vector Machine
[10] Ghose et al. 2017	34	15	Multi	Nvidia Polybench	-	Random Forest
[20] Wen et al. 2017	20	-	Multi	Parboil Polybench	36% Performance increase	Decision Tree
[4] Khalid et al. 2018	18	-	Multi	AMD parboil Polybench Rodinia	8.1% reduction in exec time. 7.07% throughput	-
[11] Moren et al. 2018	-	11	Single	AMD Nvidia Polybench	61% increase in accuracy	Random Forest
[5] Ahmed et al. 2019	930	30	Multi	AMD Polybench Own	31% reduction in exec time. 67.8% utilization. 147.35% throughput	Gradient Boosting
[13] Khalid et al. 2019	199	23	Single	AMD Parboil Polybench Rodinia	38% reduction in exec time	Random Forest, Gradient Boosting
[3] Ahmed et al. 2021	930	23	Multi	AMD Polybench	R^2:0.76 F-measure: 0.91	Decision Tree, Naive Bayes, Random Forest, KNN

2.1 Critical Analysis

After the comprehensive analysis of different approaches, there are multiple techniques for scheduling in heterogeneous environment. Ghose et al., [11] does not consider the load balancing for mapping jobs to the scheduler. Wen et al., [2], Khalid et al., [4], Ahmad et al., [5], and Khalid et al., [13] have reduced the execution time of jobs but some of them do not consider load balancing while some have not used the machine learning classifiers. The proposed model predict the execution time of jobs using machine learning classifiers, and creates a job scheduler to schedule jobs from the job pool in heterogeneous system considering the reduction in the overall execution time of jobs, increase in the throughput, and utilization of the processing device.

Chapter 3

Methodology

3.1 Introduction

Developers maps applications on CPU and GPU in heterogeneous computing environment. However, it is difficult to make a decision about mapping a job to a specific computing device (CPU GPU). There are a number of scheduling techniques and models for mapping jobs on CPU GPU processors. The scheduler have a number of jobs and makes the decision of mapping the jobs to the available computing devices. The decision should be balanced to achieve maximum throughput and utilization of devices. It is difficult for a scheduler to map applications on CPU and GPU in heterogeneous system and the decision become more critical while mapping jobs from a pool of jobs [4]. The mapping of applications on multiple heterogeneous devices is crucial for researchers therefore, for the optimal distribution of jobs, an automated approach should be developed. Most programmers uses the default scheduling strategy for mapping jobs on heterogeneous system. In this strategy, the CPU executes the serial portion (host part) of the program while GPU runs the parallel portion (kernel part) of the program. This strategy causes CPU idleness as all computation is performed by GPU. The waiting time leads in the wastage of valuable power-consuming CPU energy. It is challenging for the developer to grasp the essence of each task and choose the

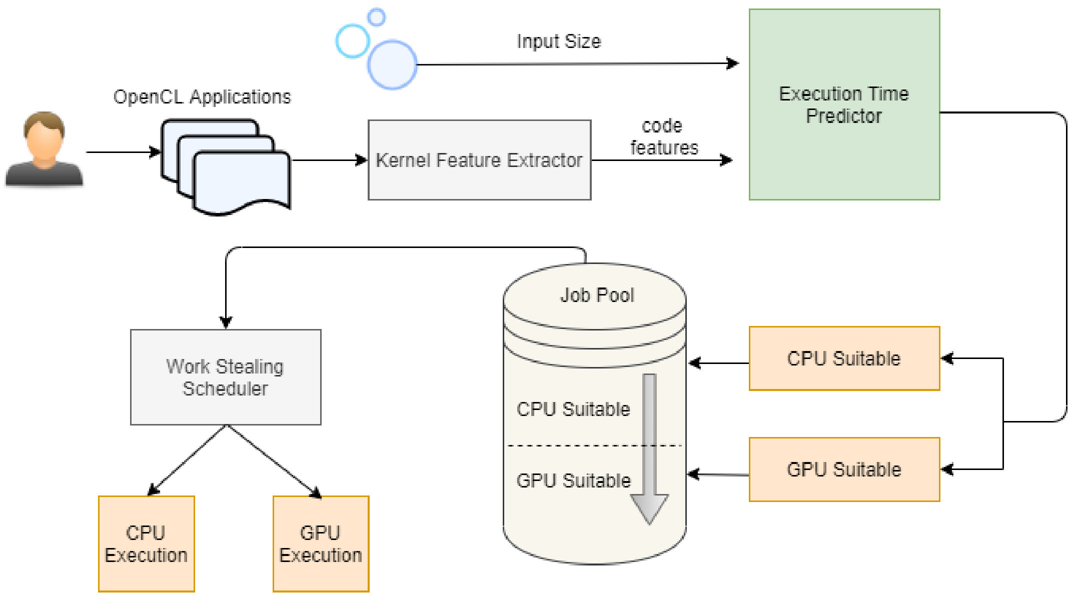


FIGURE 3.1: Methodology

device according to the nature of job and hardware specifications. It is a critical task to map applications according to their suited devices. To overcome this problem, we come up with a predictive model based on machine learning that predicts the execution time of OpenCL applications on CPU device and GPU device. The predicted execution time is used by the job allocation scheduler for the optimum workload mapping. The load-balanced task scheduler for heterogeneous systems based on machine learning is proposed that utilizes the predicted execution time of programs and then balances the load among the available processing devices. To measure the efficiency of the proposed technique, the detailed methodology is presented in figure 3.1.

The user submit data parallel OpenCL programs and the proposed scheduling system assess the submitted jobs by extracting the static code features. Correlation between the features is find out through correlation analysis. Important and influential features are selected for training of the machine learning classifier. The static code features and the dynamic workload of the jobs is used for predicting the execution time of each job for both CPU and GPU. The machine learning model uses the static code features and dynamic code features and predict the execution time of each job on CPU device and GPU device. For scheduling jobs on CPU and GPU a scheduler is designed. The scheduler uses the predicted execution time of jobs and maps each job to their suitable executing device and balancing the load between both CPU and GPU. The scheduler leads to reduce the overall execution time of the submitted jobs and gain utilization of the processing devices.

3.2 Work Stealing Scheduling Strategy

Work stealing is a scheduling algorithm that is widely used for the mapping of programs in heterogeneous environment having multi-core processing devices [31, 32]. The basic theme of the work stealing scheduler is distributing the workload on the idle processing devices which ultimately causes the reduction in overall execution time of jobs and utilization of the processing devices. In work stealing scheduler, each processing device have a double ended queue (deque) to hold the tasks assign to it. The processing device gets the processes from the deque in a Last in First out (LIFO) manner. The jobs in the deque are sorted in descending order on base of their execution time. All the jobs are put in the deque and the processing devices take one job at a time from the deque. When a processing device executes all their assigned jobs and the deque of a processing unit become empty, the device become thief and steal a job from the bottom of another processing device deque which become victim. When the thief processor execute the job and more jobs are present in other processor deque, the thief processor steal another job from the bottom of the deque. This process remains until all the available processing devices executes all the available jobs. The work stealing algorithm reduces the process migration as compare to other scheduling algorithms. It is because the migration of a process from one deque to another cannot take place when all the processing devices have jobs for execution. The stealing of jobs starts when a processing device executes all their assigned jobs. The complete hierarchy of the work stealing scheduler is depicted in figure 3.2. When the CPU processing unit completes the execution of all the assigned jobs, it becomes thief and steal the jobs from the deque of the GPU. The strategy of work stealing is widely

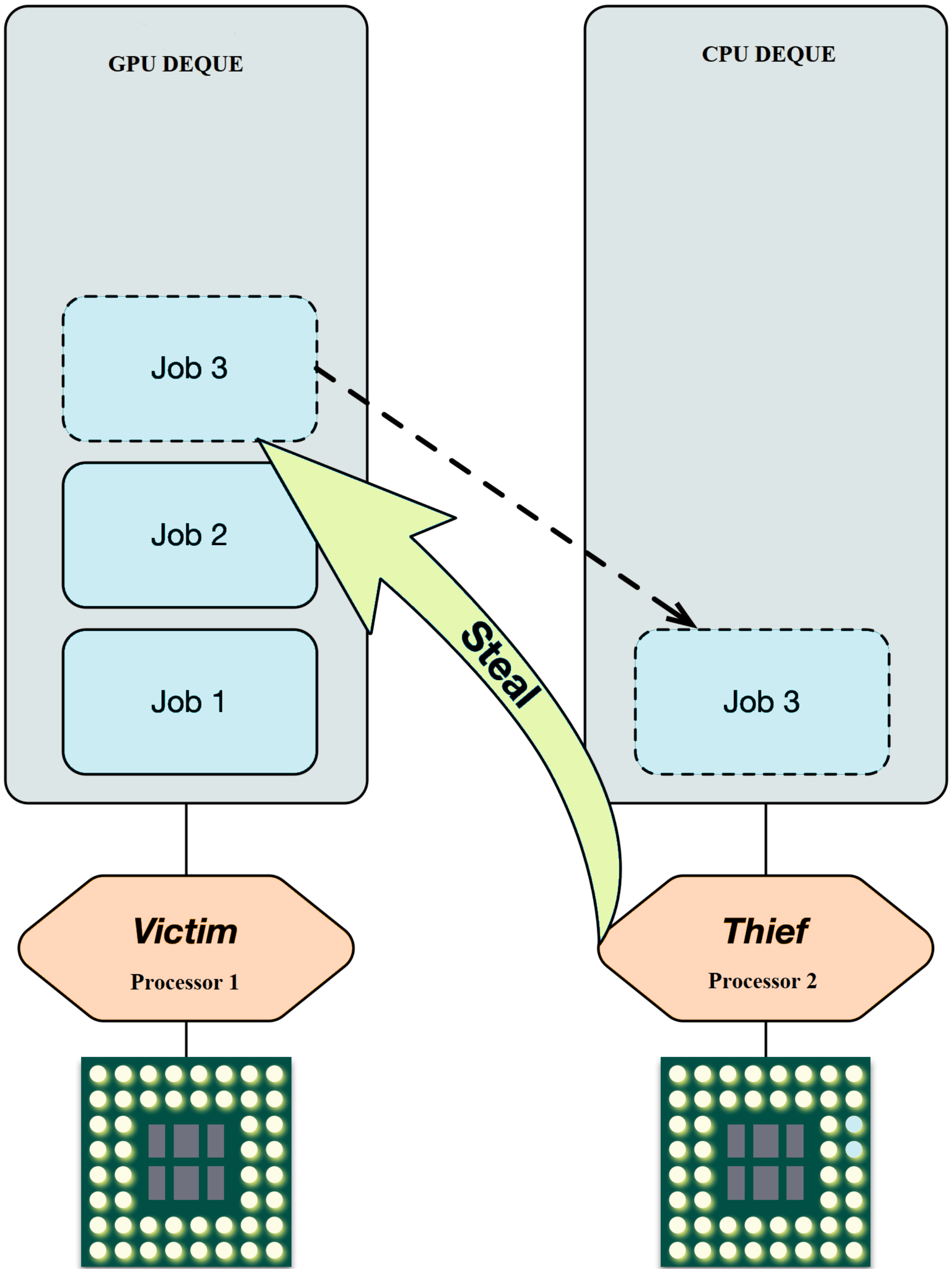


FIGURE 3.2: Work Stealing Strategy

used in scheduling jobs to processing devices. The benefit of the work stealing scheduling scheme is that when a processing device completes their execution, and there are programs that are to be executed on another processing device, it takes the programs from that processing device having jobs, by self. It means a processing device having more jobs have to only execute the programs. In this research the concept of work-stealing scheduling algorithm is used. Two deque are created for the assignment of the job to their executing device. One deque is created for the jobs to be executed on CPU and other deque for the jobs to be executed on GPU. First of all the execution time of jobs on CPU as well as on GPU is predicted through machine learning. For finding the suitable device, for each job the predicted execution time of GPU (G_time) is subtracted from predicted execution time of CPU (C_time). The scenario is depicted in figure 3.3. The difference (D_time) is sorted in ascending order. The D_time consists of both positive and negative values. A job having a negative D_time value is considered as CPU suitable as it have less predicted execution time on CPU. A job having a positive D_time value is considered to be GPU suitable as it have more execution time on CPU. A job having D_time value zero is considered as CPU suitable as in most of the cases the CPU remains idle so if a neutral job is assigned to GPU it causes increase in overall execution time of job pool and idleness of CPU processor.

$D\_time = C\_time - G\_time$

The jobs sorted on the base of D_time are moved to the job pool where two deque are maintained. Deque is open ended queue that hold the jobs for the devices. One deque have the jobs having negative values and are suitable for CPU while the other deque have jobs having positive values and are suitable for GPU. The work stealing technique is applied for assigning jobs to processing devices from the both deque. When a processing device completes their execution of all assigned jobs, it steals the jobs from another processing device. Let say CPU completed the execution of the jobs available in the deque, and GPU have jobs remaining in the deque, the CPU will become thief and steal a job from the end of the deque of GPU. CPU will take the job from end of the deque because jobs in deque are sorted in descending order and at the end of the GPU deque, the jobs have less difference in between the predicted execution time on CPU and GPU. This small difference causes small amount of penalty. The penalty is because of executing a non-suitable job on a non-suitable device. It is for sure that executing a non-suitable job with maximum D_time on a non-suitable device causes maximum penalty.

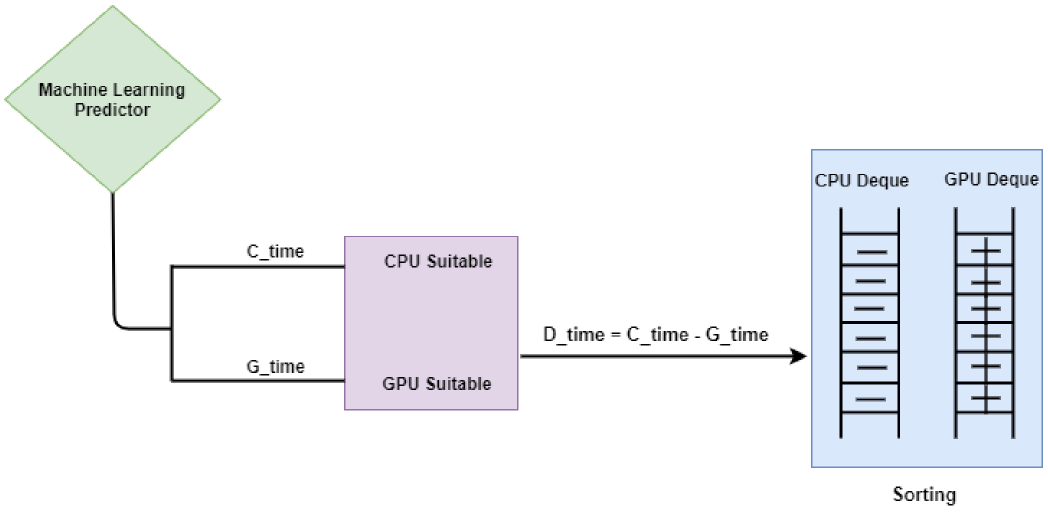


FIGURE 3.3: Scheduler Job Pool

3.3 Evaluation Benchmark

Two types of evaluation metrics are evaluated in this work. For evaluation of machine learning classifiers four evaluating metrics R-Squared (R^2) , Root Mean Square Error(RMSE), Mean Square Error (MSE), and Mean Absolute Error (MAE) are evaluated. For evaluating the efficiency of work stealing scheduler the execution time, the utilization of the devices and throughput of the processors of the proposed work stealing scheduler is compared with 10 different metrics. The metrics are CPU_Only, GPU_Only, FCFS, device suitability, Input size, and Alternate assignment. All the evaluating metrics of machine learning and scheduler are:

3.3.1 Mean Absolute Error (MAE)

Mean Absolute Error (MAE) shows the difference of the actual values and predicted values. It is the average absolute difference on the data-set. The formula of Mean Absolute Error is:

$$\text{MAE} = \tfrac{1}{N}\textstyle\sum_{i=1}^{N} |yi - y*|$$

3.3.2 Mean Square Error (MSE)

Mean Square Error also shows the difference between the actual values and predicted values. It is the average squared difference on the data-set.

$$\text{MSE} = \tfrac{1}{N}\textstyle\sum_{i=1}^{N}(yi - y*)^2$$

3.3.3 Root Mean Square Error (RMSE)

Root Mean Square Error is the square root of the Mean Square Error.

$$\text{RMSE} = \sqrt{MSE} = \sqrt{\textstyle\sum_{i=1}^{N}(yi - y*)^2}$$

3.3.4 R^2

It is co-efficient of determination. It shows the co-efficient that how accurate the predicted values fit as compared to the actual values. R-Squared is normally in the range of 0 and 1. The higher the R-Squared value the better the model is and smaller the R-Squared value inferior the model is. The formula of the R-Squared is

$$\text{R}^2 = 1 - \tfrac{\sum(yi-y*)^2}{\sum(yi-y-)^2}$$

yi - Actual value

$y*$ - Predicted value of y

$y-$ - Mean value of y

The following Six performance metrics are used for the evaluation of performance of the proposed work-stealing scheduler. These evaluation metrics are:

3.3.5 CPU Only

In CPU Only all the jobs are executed only on CPU device. Whether the jobs are CPU suitable or GPU suitable, all are executed only on CPU device. It is considered as a baseline and is used as standard for the evaluation of the proposed work-stealing scheduler.

3.3.6 GPU Only

In GPU Only all the jobs are assigned to only execute on GPU device whether the jobs are CPU suitable or GPU suitable, all are assigned to GPU device only. The GPU Only is also considered as baseline and standard for evaluation of the work-stealing scheduler.

3.3.7 First Come First Serve

In First Come First Serve (FCFS) evaluating metric the jobs are assigned to the processing devices on first come first serve base. It is to be noted that in FCFS based the concept of device suitability becomes invalid.

3.3.8 Device Suitability

In device suitability evaluation the jobs are assigned to their suitable devices. Suitable device mean the job have less execution time on that device.

3.3.9 Alternate Assignment

In alternate assignment the jobs are shuffled and assign alternate job to the processing device. In this research the jobs are randomly shuffled 5 times i.e. Alternate Assignment 1, Alternate Assignment 2, Alternate Assignment 3, Alternate Assignment 4, and Alternate Assignment 5 and their execution time is noted.

3.3.10 Input Size

All the jobs are sorted in descending order on the basis of their input size. Half of the jobs from the top of the queue are assigned to GPU device while the remaining jobs are assigned to CPU device. The jobs having largest values of input size are executed on GPU device while jobs having smallest input size are executed on CPU device.

Chapter 4

Results and Discussion

4.1 Introduction

This chapter cover the details of the machine learning prediction model and the job scheduler. In this research, two benchmark suite Polybench benchmark and AMD benchmark suite are executed for experimentation purposes. Polybench is a set of computation kernels used to test the performance of compilers and related applications, such as matrix multiplication, 2D or 3D convolution, or linear equation solver. Mentioned application is considered as a core of many applications for high-performance like image processing [33]. These benchmark suits are executed with different input size on CPU device as well on GPU device with same input size. The static code analyzer is used for extracting the code features from the programs of the OpenCL framework. Two data-set are generated from the static code features and dynamic workload along with the execution time of the program on processing devices. One data-set contain the feature set and the program execution time on CPU device and the other data-set contain the feature set with program execution time on GPU device. The important features are selected from the data-sets through correlation analysis and tree-based feature importance induction. The machine learning classifiers are applied on both the data-sets for the prediction of the execution time of the OpenCL programs. The predicted execution time is further used by the work-stealing scheduler. The work stealing scheduler assigns the jobs to the processing devices in such a manner that reduces the overall execution time of the job pool and also utilizes the processing devices. The proposed scheduler is evaluated on the base of execution time and utilization of devices metrics. Each experiment is explained in detail and the evaluation is comprehensively performed.

4.2 Experimental Setup

In this research, the experimentation are performed on multiple systems. The experimentation of the machine learning classifier are performed on laptop having windows operating system installed. The specifications of the employed system is mentioned in table 4.1

TABLE 4.1: Device Specification of Prediction Model

Device	**Specification**
Operating System	Windows 10 pro
CPU	Intel® Core™ M-7Y30
System type	64 bit
Processor	1.00 GHz, 1.6 GHz
Primary Memory	8.00 GB
Secondary Memory	1 TB

For the implementation of the machine learning model, the Python programming language is used. Python is widely used for the predictive analysis and data science related tasks of both qualitative and quantitative type of data. For the implementation of the model, the PyCharm version 2019.2.3 professional edition toolkit and Jupyter notebook IDE along with Python version 3.6 are used. The pandas, numpy, seaborn, matplotlib and sklearn libraries of python are used for the implementation of the machine learning predictor. Three different regression models multiple linear regression, Random forest, and gradient boosting are implemented on two type of data-sets. One data-set having all the extracted

features, while other data-set having only important features. Linear regression model is implemented when data is in linear form. The data used in this thesis is not in linear form that's why the result of the multiple linear regression model is not so good.
The experimentation of scheduler and the generation of the data-set, a CPU-GPU system having Intel Core i5-4460 CPU and an NVIDIA GeForce GTX 760 GPU. The experimentation are performed on Linux Ubuntu 16.04 Operating system. The specifications of the setup are mentions in table 4.2

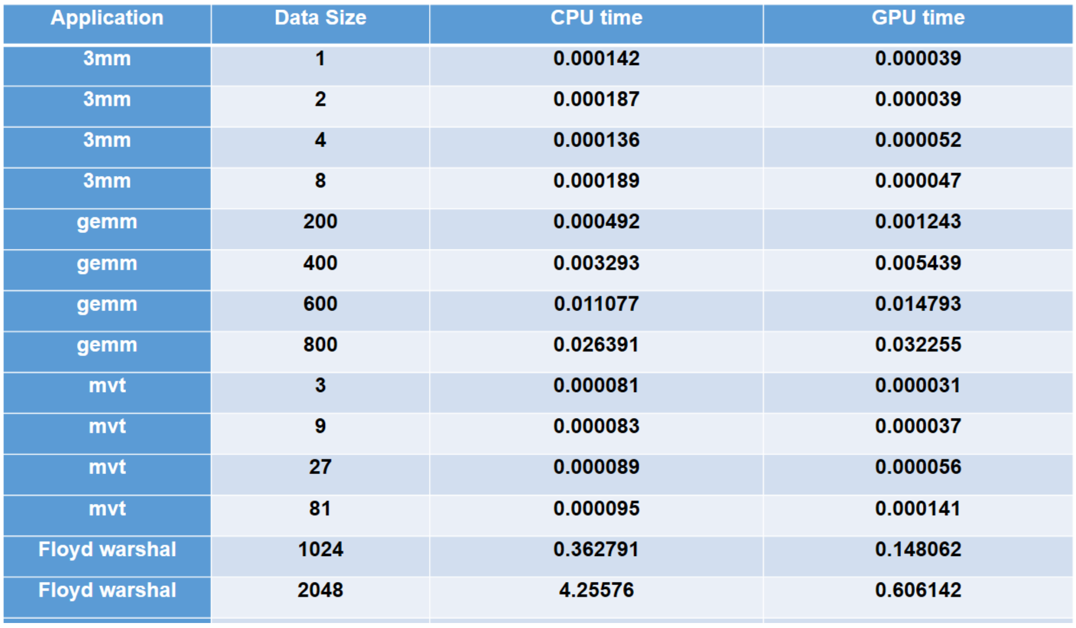
TABLE 4.2: Device Specification of Scheduler

Device	**CPU**	**GPU**
Architecture	Haswell	Kepler
Processor Number	i5-4460	GeForce GTX 760
Number of Cores	4	1152
Number of Thread	4	-
Memory	8.00 GB	2 GB
Memory Bandwidth	25.6 GB/s	192.2 GB/s
Performance	409.6 GFLOPS	2257.9 GFLOPS
ISP	32	2
Memory Speed	1600Mhz	6.0 Gbps
OpenCL SDK	Intel SDK for OpenCL 2016	CUDA 8.0
Compiler	GCC 5.4.0	Nvcc

4.3 Application Data-set

The application data-set was collected from two benchmark suits i.e. Polybench benchmark and AMD benchmark. Both the polybench and AMD benchmarks are widely used for research purposes in heterogeneous system [4, 5, 13]. A total 20 applications were executed with different input data size and a total 1165 jobs were created. The details of the benchmark suits are given in table 4.3.
The sample of the data-set generated is mentioned in figure 4.1. The Application is from the two benchmark suite polybench and AMD. 3mm, gemm, and mvt

belongs to polybench suite while the floyd, warshal application belong to AMD suite.



Application	Data Size	CPU time	GPU time
3mm	1	0.000142	0.000039
3mm	2	0.000187	0.000039
3mm	4	0.000136	0.000052
3mm	8	0.000189	0.000047
gemm	200	0.000492	0.001243
gemm	400	0.003293	0.005439
gemm	600	0.011077	0.014793
gemm	800	0.026391	0.032255
mvt	3	0.000081	0.000031
mvt	9	0.000083	0.000037
mvt	27	0.000089	0.000056
mvt	81	0.000095	0.000141
Floyd warshal	1024	0.362791	0.148062
Floyd warshal	2048	4.25576	0.606142

FIGURE 4.1: Experimental Result of Benchmark Application Execution

The data size is the input size at which the application is executed. The CPU time is the executed time of the application on CPU device and GPU time is the executed time of Application on GPU device. A total of 1165 tuples are executed on both CPU device and GPU device with different input size. first each application is executed with the increasing power of 2 i.e. 2^0, 2^1, 2^2, 2^3, 2^4 and so on. When the application program is no more executing with large input size, then the execution is started with the power of 3 i.e. 3^0, 3^1, 3^2, 3^3, 3^4. When the application program is not able to execute at large 3 power value, the programs are executed with a random number. A total 13 applications from Polybench and 7 applications from AMD suite are executed with different input size.
The details of the benchmark suite, different benchmark applications that are executed and the maximum range of the input data size of each application is mentioned in table 4.3. The AMD benchmark applications are executed with large input sizes while Polybench benchmark applications are comparatively executed with small input size The fdtd-2d application have less execution time on GPU device at every input size. The Atax application have less execution time on CPU device at every input size. The other application programs i.e. gemm, mvt,

bicg, floyd warshal have less execution time on CPU at some input size and less execution time on GPU device at some input size.

TABLE 4.3: Detail of Benchmark Suite and Input Size

Suite	**Benchmark Applications**	**Input Data Size**
Polybench	GEMM	1 – 4600
Polybench	3MM	1 – 4600
Polybench	GESUMMV	1 – 14500
Polybench	ATAX	1 – 22000
Polybench	2MM	1 – 22000
Polybench	MVT	1 – 15500
Polybench	BICG	1 – 22000
Polybench	CORRELATION	1 – 2800
Polybench	COVARIANCE	1 – 2600
Polybench	FDTD-2D	1 –12000
Polybench	GRAMSCHMIDT	1 – 8800
Polybench	SYR2K	1 – 2200
Polybench	SYRK	1 – 3000
AMD	Binomial Option	1 – 20000000
AMD	Bitonic Sort	512 – 20000000
AMD	Discrete Cosine Transformation	1 – 8192
AMD	Fast Walsh Transform	1 – 100000000
AMD	Floyd Warshall	1 – 15500
AMD	Matrix Multiplication	128 – 12500
AMD	Matrix Transpose	1024 – 10000000000

The data-set generated from these applications were used in machine learning prediction analysis. The applications from the two benchmarks i.e. Polybench and AMD covers the domain of image processing, pattern recognition, data mining, Stencils, and linear algebra. For training the machine learning model the data-set is shuffled randomly to achieve maximum accuracy. The details of the data-set is mentioned in table 4.9.

4.4 Feature Extraction

In this section the code feature extraction mechanism is explained. The static code analyzer is designed for the extraction of features from the kernel code of OpenCL programs of two benchmark suite i.e. Polybench and AMD. The extracted features set shows the behavior of the programs. The aim of the extraction of features from the code is to collect the attributes of the OpenCL programs from both benchmarks. The Clang (front end) compiles the code of the OpenCL programs to ensure the code is error free. The LLVM passes extracts the code features. Figure 4.2 shows the complete methodology of extraction of the features from the OpenCL programs.

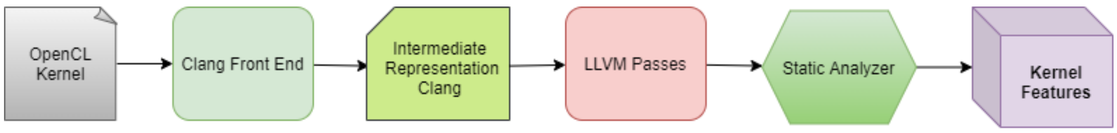


FIGURE 4.2: Code Feature Extraction

In this research, a total of 22 features are extracted from the OpenCL application programs. The extracted features and the input data size are provided to the machine learning predicting classifiers. The input data size is type of feature an OpenCL program is executed with multiple input size for generating of the data-set. For example, Atax, an OpenCL application is executed with different input Size starting from 1 and maximum of 22000 on both CPU device and GPU device. These features consists of number of loops, number of int data type, number of float data type, total number of functions, total number of blocks, total number of instructions, total number of loops etc. Table 4.4 consists of all the features extracted from OpenCL programs through clang and LLVM. The extracted code features and the workload of the jobs are provided to the execution time prediction classifier and classify the submitted jobs, according to execution time. The predicted execution time is further utilized by the load balancing job scheduler.

Table 4.4: Static Code Features

S.No	Feature Name
1	Total number of Return Statements
2	Total number of Control Statements
3	Total number of Allocation Instructions
4	Total number of Load Instructions
5	Total number of Store Instructions
6	Total number of Multiplication (Float Data type) Operations
7	Total number of Multiplication (Integer Data type) Instructions
8	Total number of Division (Float Data type) Instructions
9	Total number of Division (Integer Data type) Instructions
10	Total number of Condition Check Instructions
11	Total number of Addition (Float Data type) Instructions
12	Total number of Addition (Integer Data type) Instructions
13	Total number of Subtraction (Float Data type) Instructions
14	Total number of Subtraction (Integer Data type) Instructions
15	Total number of Function Call Instructions
16	Total number of Functions
17	Total number of Blocks
18	Total number of Instructions
19	Total number of Float Operations
20	Total number of Integer Operations
21	Total number of Loop Operations
22	Total number of Loops

4.5 Feature Selection

Feature selection find out the importance of the features with respect to the class label of the data-set (CPU time or GPU time) and correlation of each feature with another feature. The selection of important feature is very crucial because it causes reduction in over fitting, improves the accuracy, and reduces the training time of the classifier. The Two feature selection methods, feature correlation analysis and the tree based importance are adopted in this research. The Pearson correlation is a filter based correlation analysis method that is widely used for finding correlation between features. We used the Pearson correlation method for feature correlation. The two features are correlated if change in value of one feature causes change in another feature value. If increase in one feature value causes increase in other feature or decrease in one feature value causes a decrease in another feature value, the two features are correlated positively, and if decrease in value of a feature causes increase in other feature value or increase in one feature value causes decrease in another feature value, the two features are correlated negatively. Using highly correlated features for training purposes causes lower prediction accuracy. In this research Heatmap is used for finding the correlation between features. Figure 4.3 shows the matrix of the correlation of features employed in table 4.4. The two features having value 1 or tends to 1 are highly correlated and features having values negative or tends to a negative value are not as much correlated. The following features have positively or negatively correlation with other features.

- Data size
- Total number of Return Statements
- Total number of Allocation Instructions
- Total number of Load Instructions
- Total number of Multiplication (Float Data type) Operations
- Total number of Subtraction (Integer Data type) Instructions
- Total number of Function Call Instructions
- Total number of Float Operations

These features are also marked as important by Tree based feature selection as shown in figure 4.4. The following features have neutral relation with all other features. These type of features can influence the result.

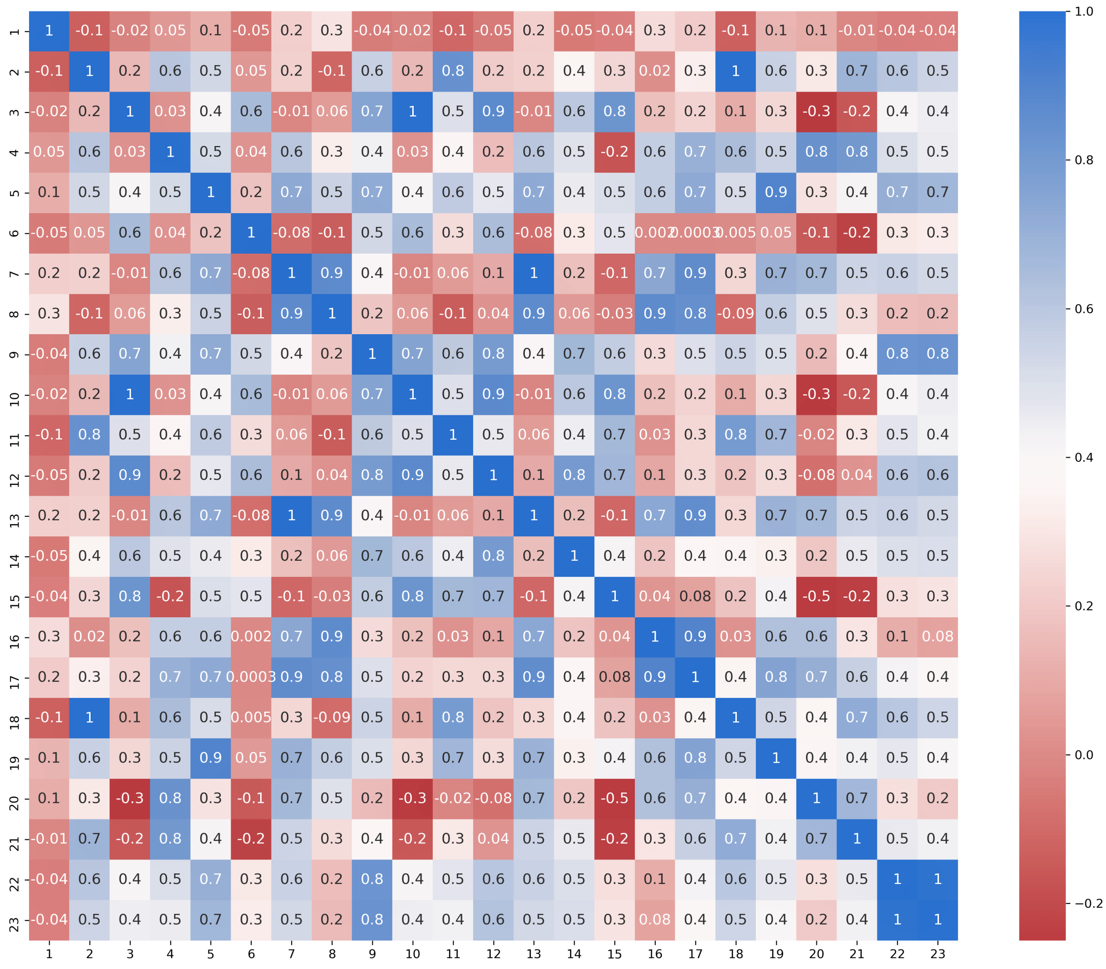


FIGURE 4.3: Correlation Analysis of OpenCL Features

- Total number of Integer Operations
- Total number of Loop Operations
- Total number of Blocks

The following features are low or not influencing the class label so therefore all these features are excluded from the data-set.

- Total number of Control Statements
- Total number of Multiplication (Integer Data type) Instructions
- Total number of Condition Check Instructions
- Total number of Addition (Float Data type) Instructions

- Total number of Addition (Integer Data type) Instructions
- Total number of Store Instructions
- Total number of Division (Float Data type) Instructions
- Total number of Division (Integer Data type) Instructions
- Total number of Subtraction (Float Data type) Instructions
- Total number of Instructions

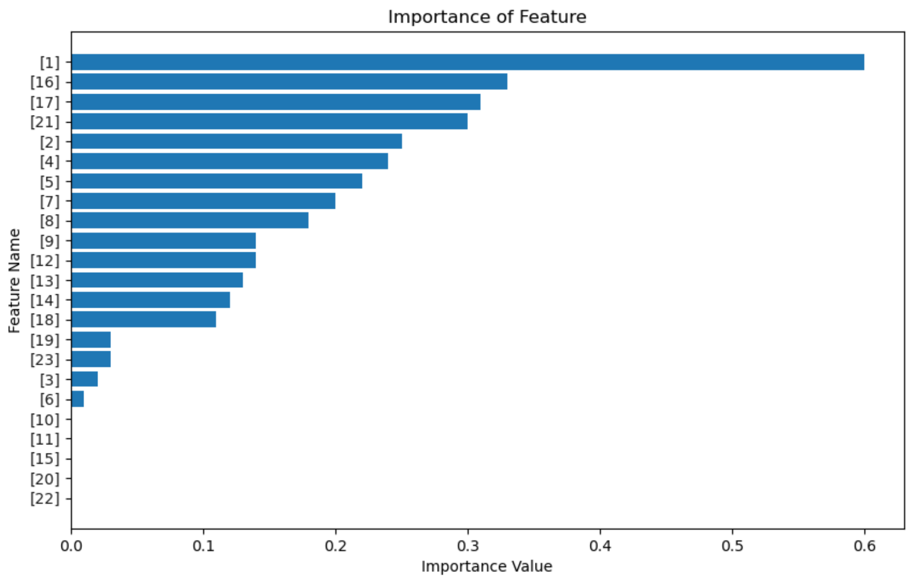


FIGURE 4.4: Tree-Based Feature Values

After the correlation analysis of the features and importance of features, the important and influential features are selected that are used by the machine learning model for prediction of the execution time on CPU device and GPU device. The decision tree based importance of feature method is used for finding importance of a feature. The Classification and Regression Trees (CART) [34] of decision tree provide a value to each feature on the basis of the Information Gain. The tree-based method assign a value between 0 and 1 to each feature according to their importance. The importance of a feature is that how much the feature is influencing on the class label of the data-set. A feature value 1 or approaching to 1 means the feature is important. The feature having value zero or approaching to zero means the feature is not that much important. The table 4.5 are the features marked as important by the tree-based model. In this research thesis the top features from the feature set are then selected for the machine learning-based prediction of execution time for CPU device and GPU device. The top features along with their score is shown in figure 4.4

Table 4.5: Important Features from the Feature Set

S.No	**Feature Name**
1	Data Size
2	Total number of Return Statements
4	Total number of Allocation Instructions
5	Total number of Load Instructions
7	Total number of Multiplication (Float Data type) Operations
8	Total number of Addition (Integer Data type) Instructions
16	Total number of Subtraction (Integer Data type) Instructions
17	Total number of Function Call Instructions
21	Total number of Float Operations

4.6 Machine Learning Model Selection

This section explains the machine learning prediction classifier selection. Machine learning is hot area of nowadays research and is widely used for prediction purposes. There are two types of machine learning. One is supervised machine learning and other is unsupervised machine learning. Supervised learning is type of machine learning in which the output class/class labels are provided to a classifier for training and testing the model, While unsupervised learning does not have output class. Clustering is an example of unsupervised machine learning.

Supervised learning have further two types: classification and regression.

Classification is type of supervised learning in which the output class label is categorical (discrete) while the regression have numerical (continuous) output class label.

In this research regression models are used as the output class i.e. CPU execution time and GPU execution time is a numeric value. A data-set is generated from the extracted static code features and dynamic workload. The data is pre-processed before using for training of the classifier for the prediction of the execution time. The complete working strategy is depicted in figure 4.5.

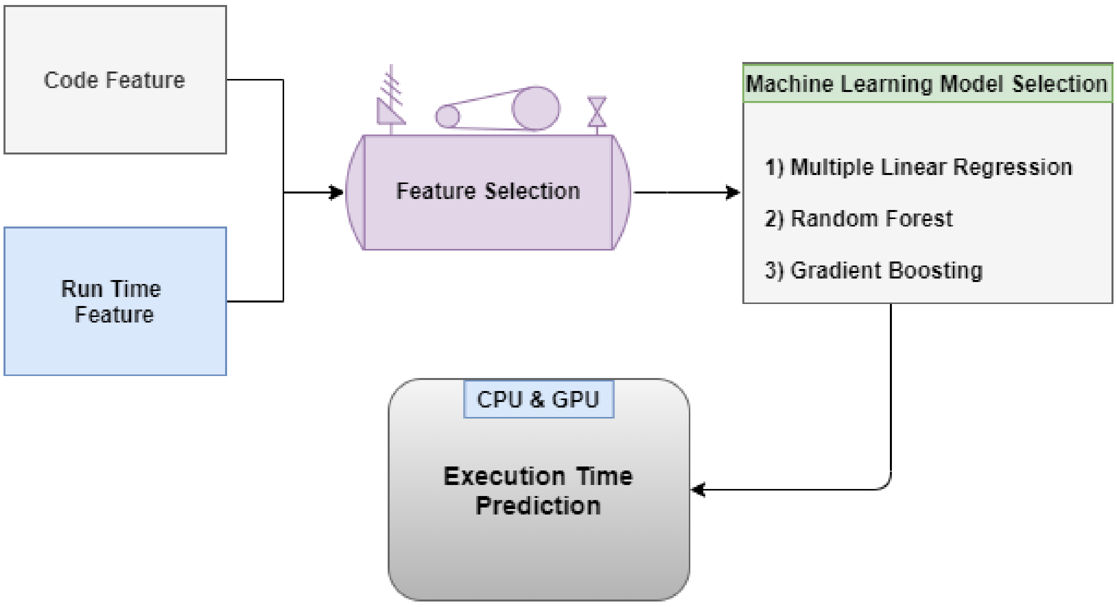


FIGURE 4.5: Machine Learning Model

After the Pre-processing completes, feature engineering is applied on the data-set in the feature selection portion. The feature engineering select important and influential features from the available extracted features for training and testing of the classifiers.

A detailed explanation of feature selection is discussed in section 4.5. Three machine learning regression models, Multiple Linear Regression, Random Forest Regression, and Gradient Boosting Regression models are trained for the prediction of the execution time of jobs on CPU device and GPU device. The multiple linear regression does not fit well on training and testing data because the data is not in linear form. The details of multiple linear regression model are depicted in table 4.6. The random forest model fits very well on training and testing models, and is applied for the prediction of the execution time of OpenCL applications on CPU device and GPU device. The number of trees in random forest model is 1000 and

TABLE 4.6: Multiple Linear Regression Model Specification

Parameter	**Shape**
Test size	0.2
Train size	0.8
Random state	42
Regressor	LinearRegression()

the random state is kept 45. The details of random forest model are depicted in table 4.7. The predicted execution time for CPU device and GPU device by the random forest model is further utilized for the designing of the work-stealing scheduler. The random forest model is further utilized because it has maximum evaluating values than multiple linear regression and gradient boosting regression models.

TABLE 4.7: Random Forest Model Specification

Parameter	**Shape**
Test size	0.2
Train size	0.8
Random state	45
Regressor	RandomForestRegressor()
n_estimators	1000

The Gradient Boosting model also gives maximum accurate results. Gradient boosting model has good results over the multiple linear regression model but have lower results from random forest model. Therefore we only uses the results of random forest model further for the prediction of the execution time of jobs on CPU device and GPU device.

The gradient boosting model is based on trees. The n_estimators in the table 4.8 is the total number of trees used in the model. The max depth means the number of nodes in each tree. In the model the value of max depth is 4. The value of minimum split in the model is 5 and the learning rate is 0.01. The details of gradient boosting is mentioned in table 4.8.

Table 4.8: Gradient Boosting Model Specification

Parameter	**Shape**
Test size	0.2
Train size	0.8
Random state	44
Regressor	GradientBoostingRegressor()
n_estimators	1000
Max depth	4
Min_sample_split	5
Learning rate	0.01

4.6.1 Model Training and Testing

Two data-sets CPU_dataset and GPU_dataset are created and used for training and testing the machine learning model for the prediction of the execution time of jobs on CPU device and GPU device. One data-set contain values of all features of benchmark applications and their execution time on CPU device, while other data-set contain all feature values and execution time on GPU device. Two random forest models are trained, CPU model is trained on CPU_dataset for the prediction of the execution time of jobs for CPU device while GPU model is trained on GPU_dataset for the prediction of the execution time of jobs for GPU device. Both the models are executed on two data-sets, one data-set having values of all 23 features and one data-set having only the important and influencing features determined by correlation analysis and tree-based information gain ratio. The complete details of the three machine learning models is given in table 4.10.
The data-set contain 1165 tuples and 25 columns. The data-set is split in to two parts with the ratio of 20% and 80%. Table 4.9 shows the complete details of the data-set. The 80% of the data-set along with the output class label is provided to the model for training purpose. The remaining 20% is used for testing the trained model. The testing is performed as only the features values are provided to the model and the model predict the output class label for it.The training and testing fitness of the model is depicted in figure 4.6.

TABLE 4.9: Detail of Data-set used for Prediction

Set	**Instances**	**Percentage**
Training set	931	80%
Testing set	234	20%
Complete data-set	1165	100%

The predicted class labels through the random forest model are compared to the actual values of the testing data-set. Figure 4.7 shows the graph of actual and predicted values.

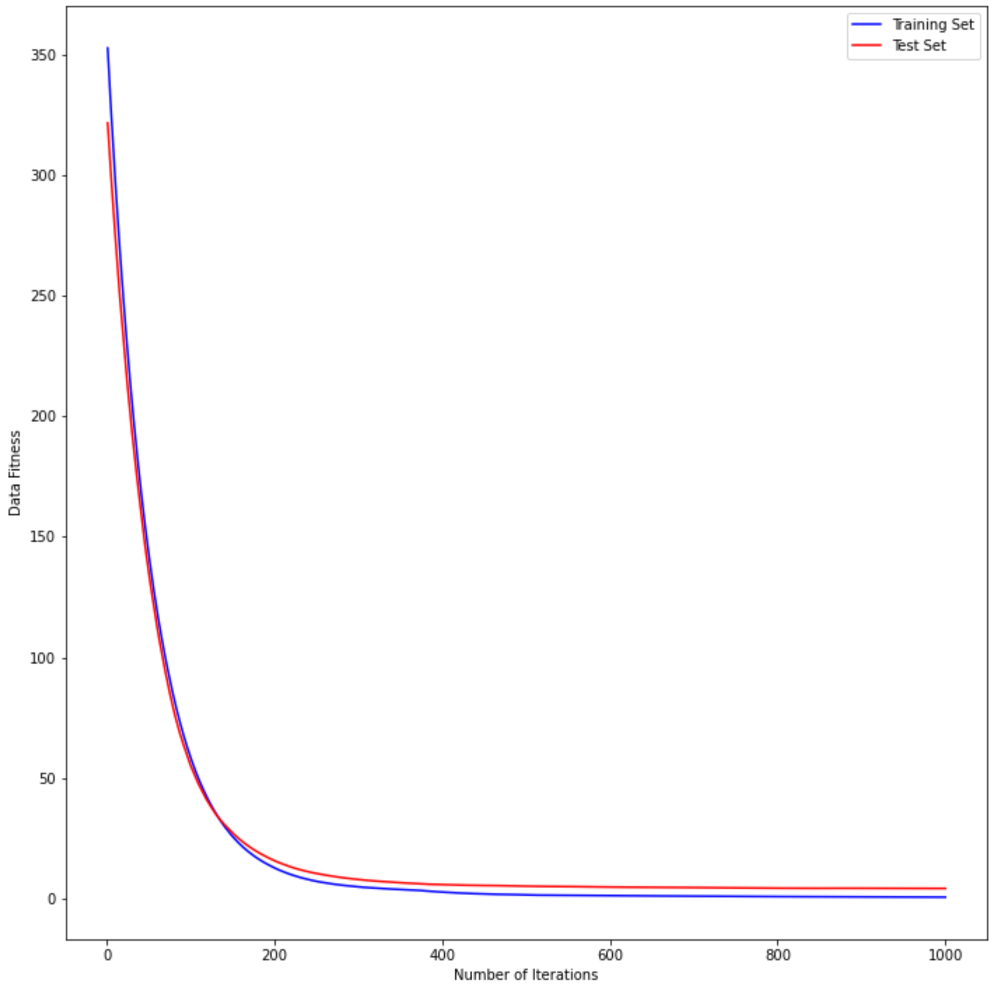


FIGURE 4.6: Training and Testing Fitness

The figure 4.8 shows the correlation of the actual and the predicted values of the machine learning model. When the data is closely fitted on the linear line, it indicates that the model have predicted the class labels with maximum accuracy.

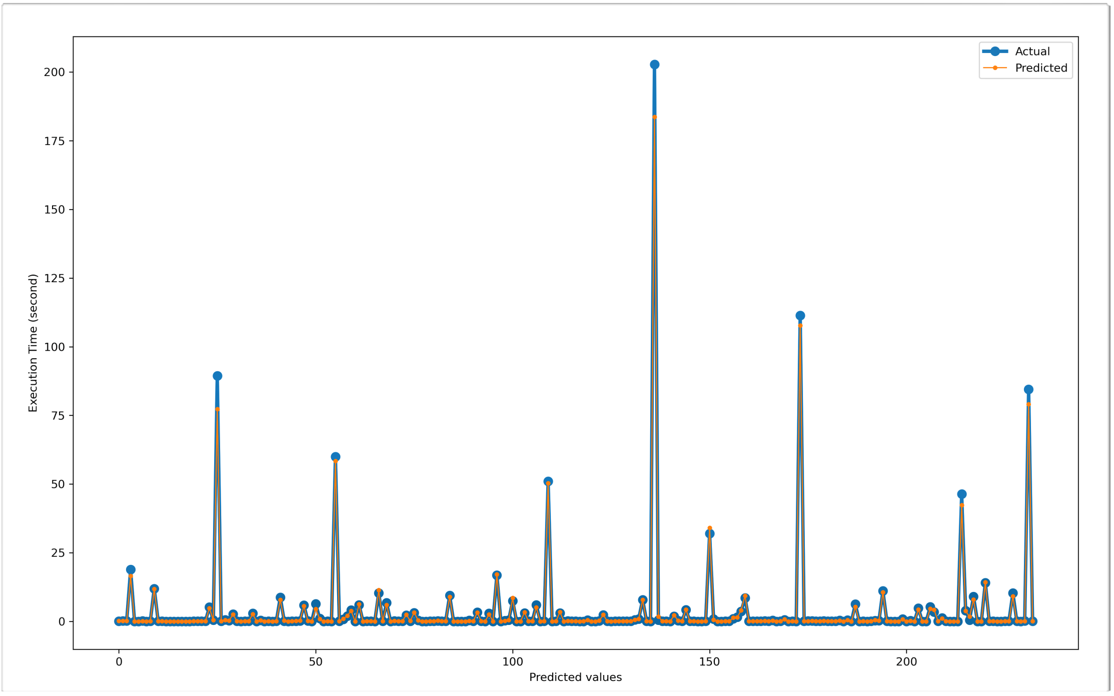


FIGURE 4.7: Actual and Predicted Values

The error rate of the predictive models in machine learning for regression are assessed by evaluating multiple performance metrics like R^2, Root Mean Square Error (RMSE), Mean Square Error (MSE), and Mean Absolute Error (MAE). The concept of evaluation in regression is that comparing the predicted values with the actual target values. The table 4.10 shows the comparison of three machine learning models i.e. Multiple linear regression, random forest, and gradient boosting regression model. This table shows the comparison of the evaluation of the machine learning models on four different data-sets i.e. C_all column means all feature values and the CPU execution time values, C_Reduced column shows the execution time of CPU with reduced features, G_All is the data-set having all features and GPU execution time, and G_Reduced is the dataset having only important features with GPU execution time. The reduced features mean only that features marked important and influential by the tree based information gain.
It is noted in the experiments that multiple linear regression model performs well on all features while the random forest model and gradient boosting model improves their R^2 value and other evaluating metrics values by reducing the number of features. Because of the improvement in results only important features are further utilized by the random forest and gradient boosting models.

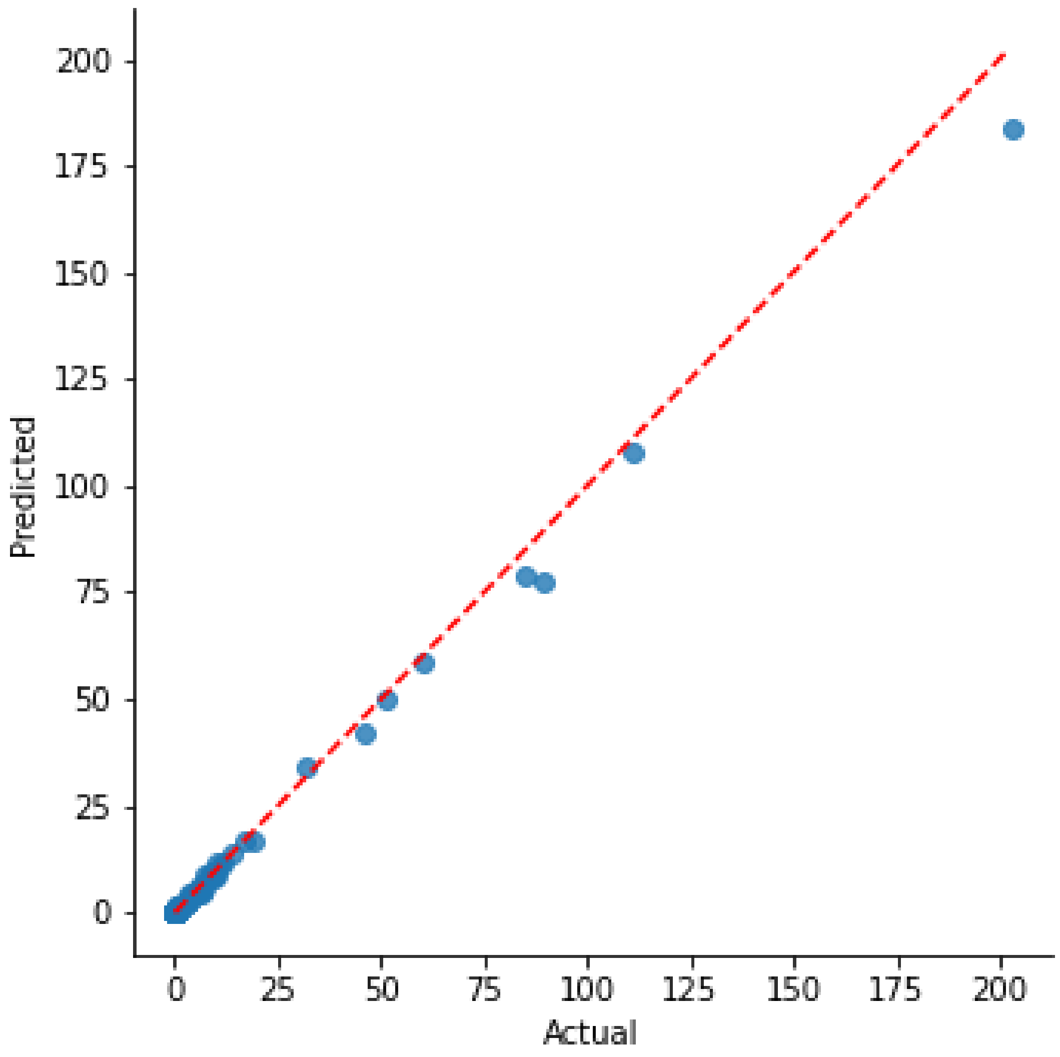


FIGURE 4.8: Actual and Predicted Values Fitness

R^2 is coefficient of determination. It is statistical measure showing the coefficient that how accurate the predicted values fit as compared to the actual values. R^2 is normally in the range of 0 and 1. The 0 mean 0% and 1 mean 100% but the percentage does not shows the accuracy. It only indicates the fitting of actual and predicted values. The higher the R^2 value the better the model is and smaller the R^2 value inferior the model is. In our case the random forest and gradient boosting models fits very accurately as compare to multiple linear regression. The random forest and gradient boosting models have almost same values but multiple linear regression model is 88% lower than both models. It is because the data is not in linear form. The figure 4.9 shows the comparison of R^2 values of multiple linear regression, gradient boosting and random forest model.

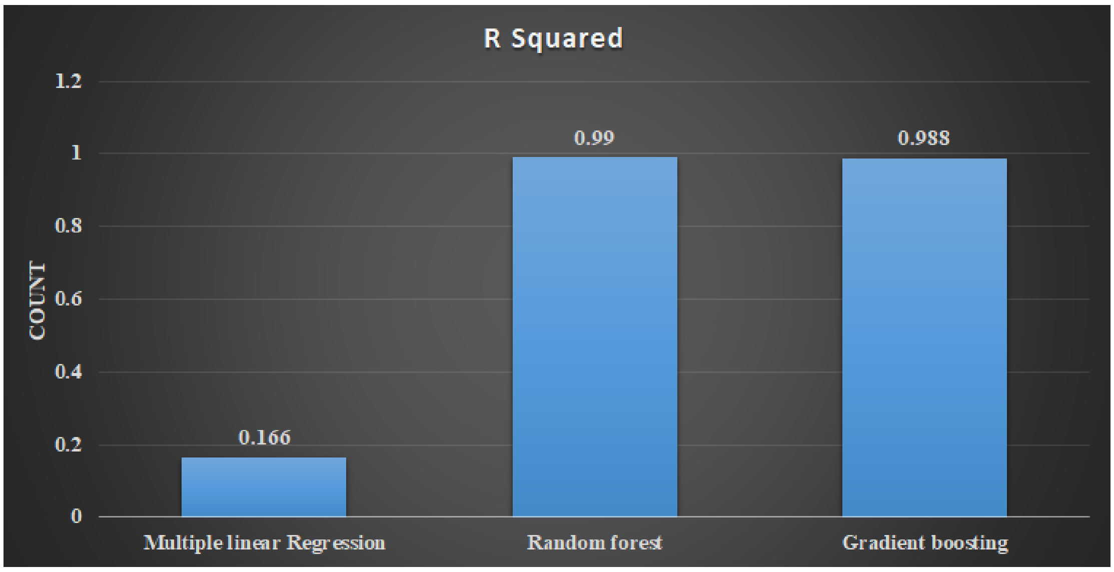


FIGURE 4.9: R-Squared Comparison

TABLE 4.10: Machine Learning Models Specification

Classifier	**EM**	**C_All**	**C_Reduced**	**G_All**	**G_Reduced**
MLR	R^2	0.31	0.157	0.314	0.166
MLR	MAE	26	38.18	5.24	7.56
MLR	MSE	5934	7180	227.8	277
MLR	RMSE	77	85	15.1	16
RF	R^2	0.99	0.99	0.99	0.99
RF	MAE	1.97	2.103	0.339	0.353
RF	MSE	73.35	35.37	2.61	2.64
RF	RMSE	8.56	5.95	1.62	1.63
GB	R^2	0.924	0.96	0.966	0.988
GB	MAE	8.04	3.94	1.018	0.676
GB	MSE	948.6	311.77	15.56	4.071
GB	RMSE	30.8	17.66	3.95	2.018

The mean absolute error (MAE) is evaluation metric used for the evaluation of the results of regression models. MAE is the sum of the absolute differences between the actual values and predicted values. Figure 4.10 is the representation of MAE. In the case of mean absolute error the lower the value the precise the model is and vice versa. In our prediction analysis the random forest have the minimum value among gradient boosting and multiple linear regression models. The random forest have 44% small value from gradient boosting and 80% from multiple linear regression. The figure 4.11 shows the comparison of the mean square error values

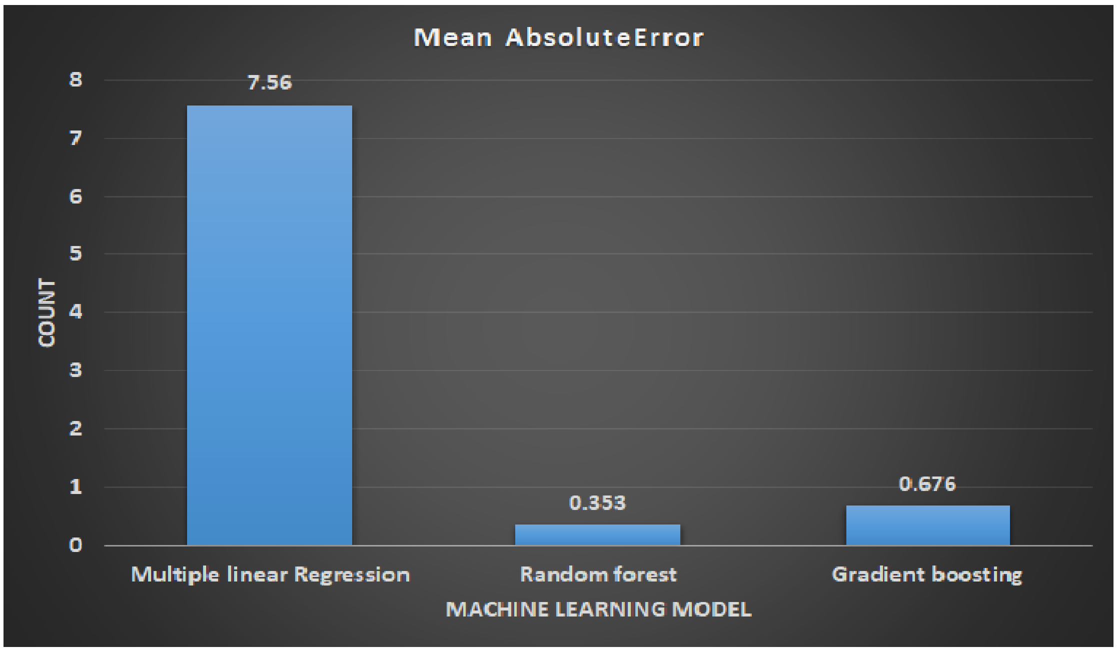


FIGURE 4.10: Mean Absolute Error Comparison

of the three machine learning models. It shows that the multiple linear regression have the maximum value among the three models. The random forest model again has less value from both multiple linear regression and gradient boosting machine learning models. The random forest model has 99% smaller value than multiple linear regression model and 35% smaller value than gradient boosting regression model. Root mean square error is the standard deviation of the prediction error from the line of best fit. In case of root mean square error the smaller the value the lower the error in the model and vice versa. Figure 4.12 shows the comparison of the values of the RMSE of the machine learning models.

4.7 Work Stealing Scheduler

In this research the concept of work-stealing scheduler is used for scheduling job-s to processing devices in heterogeneous system. The work-stealing strategy is briefly described in section 3.1. All the jobs are provided to the model.

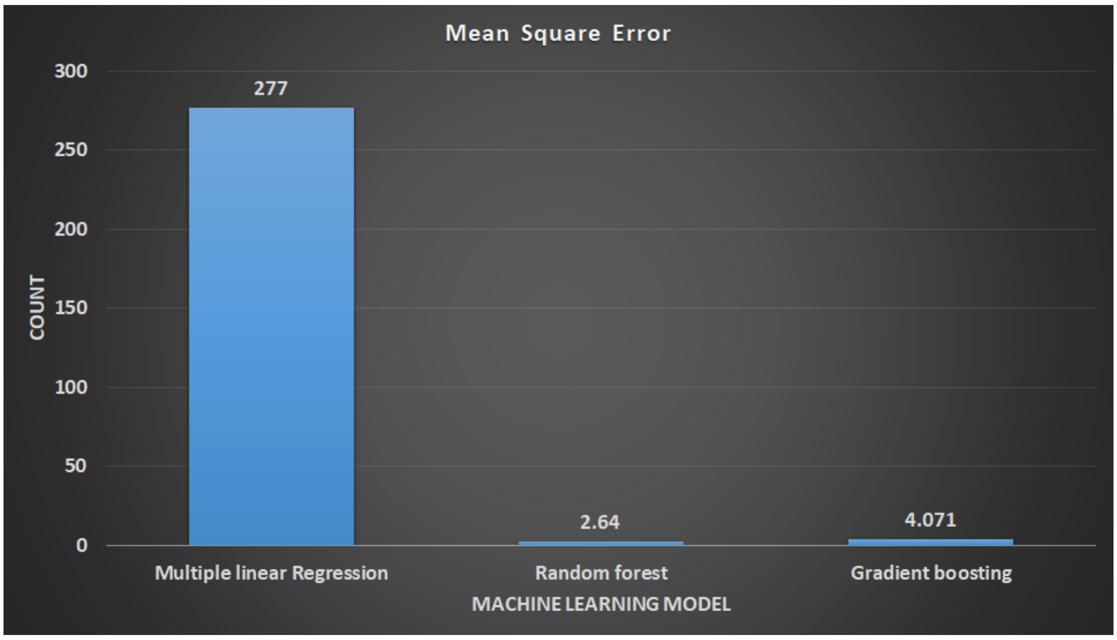


FIGURE 4.11: Mean Square Error Comparison

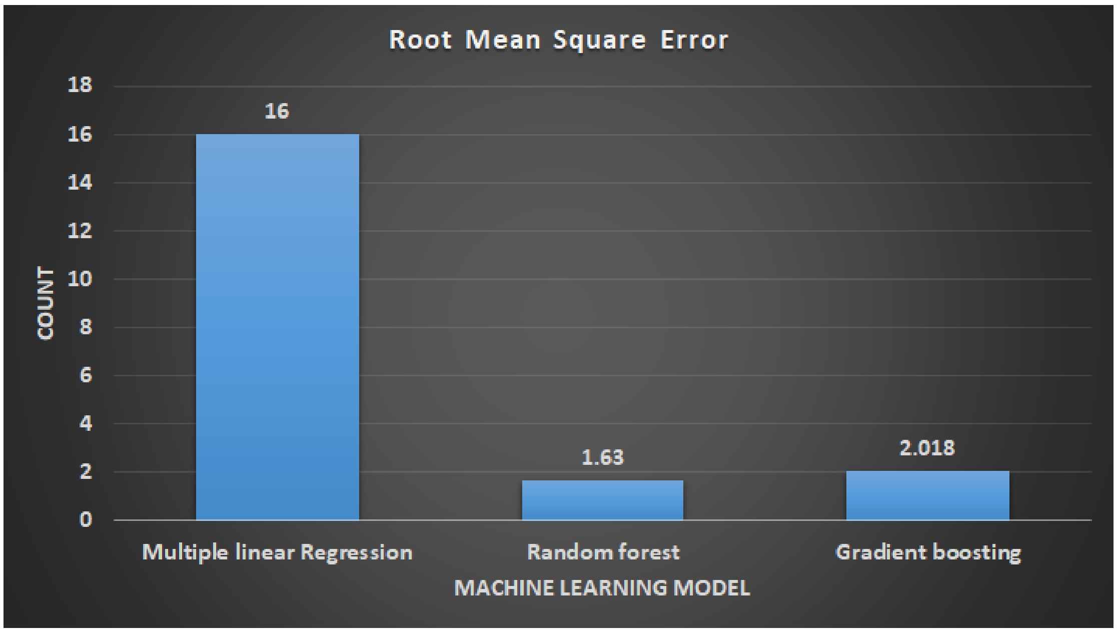


FIGURE 4.12: Root Mean Square Error Comparison

The model have predicted the execution time of each job on CPU device and G-PU device through machine learning. The predicted execution time of GPU is subtracted from predicted execution time of CPU. The new values of the jobs are sorted in ascending order. The jobs having a negative value are CPU suitable while jobs having positive value are GPU suitable. Table 4.11 shows a sample of the jobs of the model. Each application is mentioned along with their input data size, execution time predicted for CPU, execution time predicted for GPU, and difference of CPU and GPU. All the jobs are sorted in ascending order on the basis of the difference column mentioned in table 4.12.

TABLE 4.11: OpenCL Jobs with Input Size and Predicted Values

App	**Input Size**	**Pred C_time**	**Pred G_time**	**Difference**
3mm	2800	4.312111	2.4415846	1.8705264
3mm	3000	5.112258	2.9505387	2.1617193
Gemm	1024	0.0926515	0.295397	-0.20275
Gemm	2048	2.6347962	1.7251938	0.909602
Atax	5000	0.0128255	0.0209392	-0.00811
Gesummv	9500	0.0373109	0.1524943	-0.1151834
Gesummv	8000	0.0282827	0.1352338	-0.1069511
Mvt	400	0.0002719	0.0006853	-0.00041
DCT	1024	0.1389051	0.0913682	0.047537
DCT	2048	0.1407294	0.0817043	0.059025

The jobs having negative value are considered as CPU suitable as they have less execution time on CPU device as compared to GPU device. The jobs having positive value are considered as GPU suitable as they have less execution time on GPU device as compared to CPU device. In the table 4.12 the first 5 programs are CPU suitable as they have negative difference value while the last 5 programs are GPU suitable as they have positive difference value. The sum of the execution time of CPU suitable jobs is 0.17134 while sum of the execution time of GPU suitable jobs is 7.29032. There is a huge difference between the overall execution time of CPU and GPU. If CPU suitable jobs are assigned to CPU and GPU suitable jobs are assigned to GPU and no scheduling is performed, the CPU executes their jobs and will waits for the GPU to execute their programs. It causes an increase in the overall execution time of jobs and also under-utilization of the CPU. If both the CPU execution time and GPU execution time is equal then assign the jobs to their suitable device having less execution time. If overall execution time of CPU is more than overall execution time of GPU, the work-stealing scheduler move

Table 4.12: OpenCL jobs with Sorted Difference Time

Application	**Input Data Size**	**Sorted Difference**
Gemm	1024	-0.20275
Gesummv	9500	-0.1151834
Gesummv	8000	-0.1069511
Atax	5000	-0.00811
Mvt	400	-0.00041
DCT	1024	0.047537
DCT	2048	0.059025
Gemm	2048	0.909602
3mm	2800	1.8705264
3mm	3000	2.1617193

one job from the end of CPU queue to GPU as it will have least predicted execution time. When GPU execute the job with penalty, the scheduler check if there is any job remaining in CPU queue, if a job is available the scheduler move the last job from CPU deque to GPU deque. The scheduler repeat the process until all the jobs of CPU are executed and both the processing devices completes the execution of jobs at almost on the same time. If the overall execution time of GPU jobs is more than CPU jobs then the scheduler move the job from GPU deque to CPU deque having least execution time on GPU device, and repeat the process until the jobs execution is completed.
The execution time of each job is predicted through machine learning for both CPU device and GPU device and it is for sure that job having less predicted execution time on CPU device will have more execution time on GPU and job having less predicted execution time on GPU device will have more execution time on CPU device. The job with less predicted execution time for CPU device have more execution time on GPU device because a penalty is also added in the form of time as executing a non-suitable job that causes increase in execution time on GPU device. If the predicted execution time value is smaller, the penalty will also be smaller and if the predicted execution time value is larger the penalty will also be

larger. When one job from CPU deque moves to GPU deque it causes decrease in overall execution time of CPU deque and increase in overall execution time of GPU deque. Moving a job from GPU deque causes decrease in overall execution time of GPU device but increase in overall execution time of CPU device. It is because of executing a program on non-suitable device. If the overall execution time of device having maximum time, reduces and both CPU and GPU devices completes their execution almost on same time it indicates that overall execution time is reduced and resource utilization is achieved with maximum throughput. The complete scenario of the work stealing scheduler is shown in the following figure 4.13.

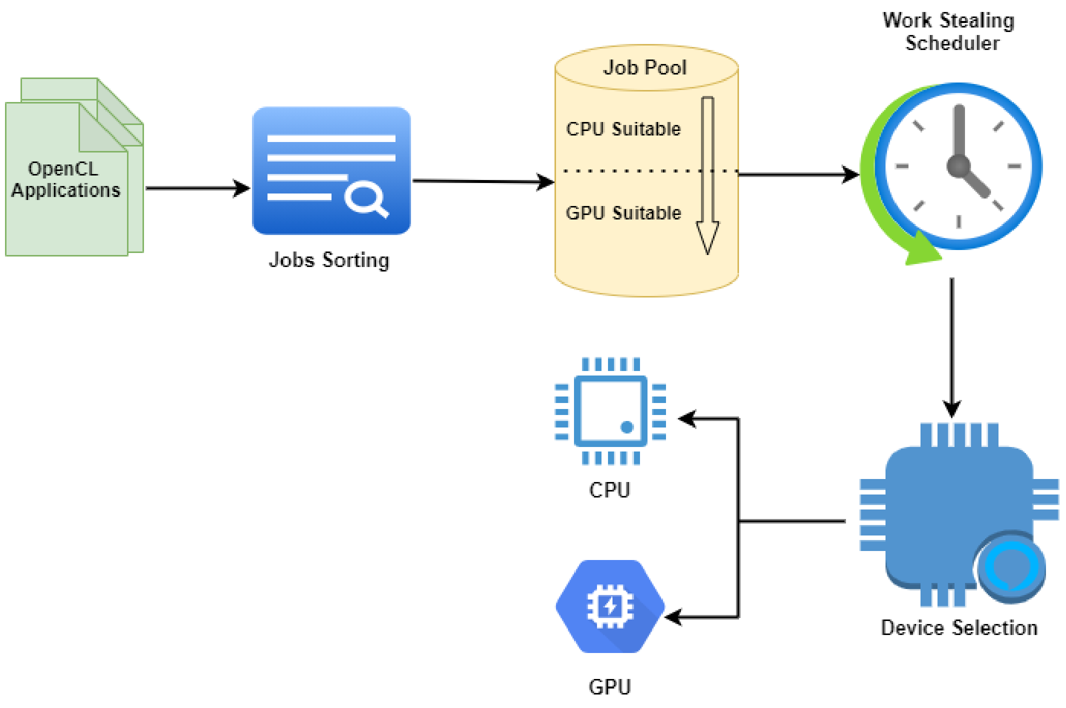


FIGURE 4.13: Work Stealing Scheduler

4.7.1 Work Stealing Performance and Comparison

In this research, a total 120 programs are selected from two benchmarks suits polybench and AMD. The job pool contain 35 CPU suitable jobs and 85 GPU suitable jobs. CPU suitable means the jobs have less predicted execution time on

CPU while GPU suitable jobs have less predicted execution time on GPU. The two performance metrics execution time and utilization of devices are elaborated. The figure 4.14 shows the comparison of all the scheduling heuristics on the basis of the execution time.

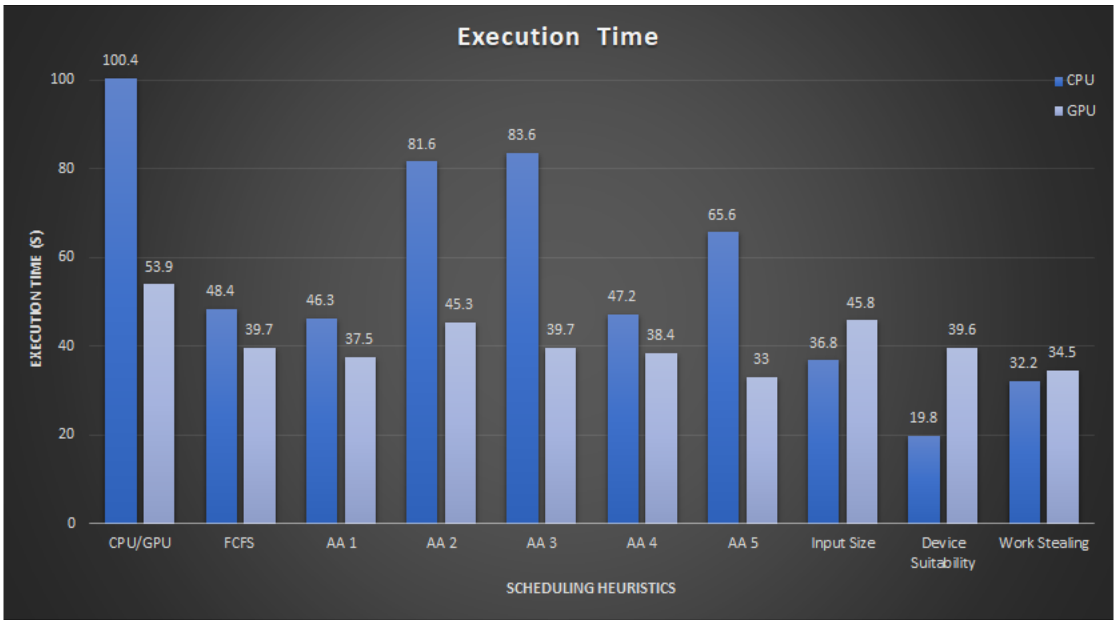


FIGURE 4.14: Execution Time of Scheduling Heuristics

The figure 4.14 shows that none of the scheduler have less execution time on both CPU device and GPU device except our proposed model work stealing. The device suitability have less execution time on CPU but have more on GPU device and have less utilization of the devices as compared to the proposed scheduler. The alternate assignment 5 have less execution time on GPU device but have more execution time on CPU device as compared to work stealing scheduler.

When all the jobs are executed on CPU only, it got 65.63% more execution time from the proposed scheduler and 0% utilization of devices as GPU was idle on that time. When all the jobs are executed only on GPU device it takes 35.9% more execution time from the proposed scheduler and under-utilization of the devices. The first come first serve takes 28.7% more time for the execution of all the jobs. We performed the experiment of assigning alternate jobs to CPU and GPU device five times. Each time shuffle all the jobs randomly and assigned each job alternatively. The alternate assignment 1 have 25.5% more execution time. The alternate assignment 2 have 57.7%, alternate assignment 3 have 58.7%, alternate assignment 4 have 26.9% and alternate assignment 5 have 52.6% more execution time than the proposed scheduler. The input size have 75.3%, the device suitability have 87.1% more execution time than the proposed scheduler.

The figure 4.15 shows the utilization of the devices by the schedulers. The graph shows that none of the scheduler have more utilization of device except our proposed model work stealing. We executed all the jobs on CPU device only and got 65% increase in execution time and 0% utilization of devices as GPU was idle on that time. When all the jobs are executed only on GPU device the overall execution time got an increase of 56% and under-utilization of the devices.

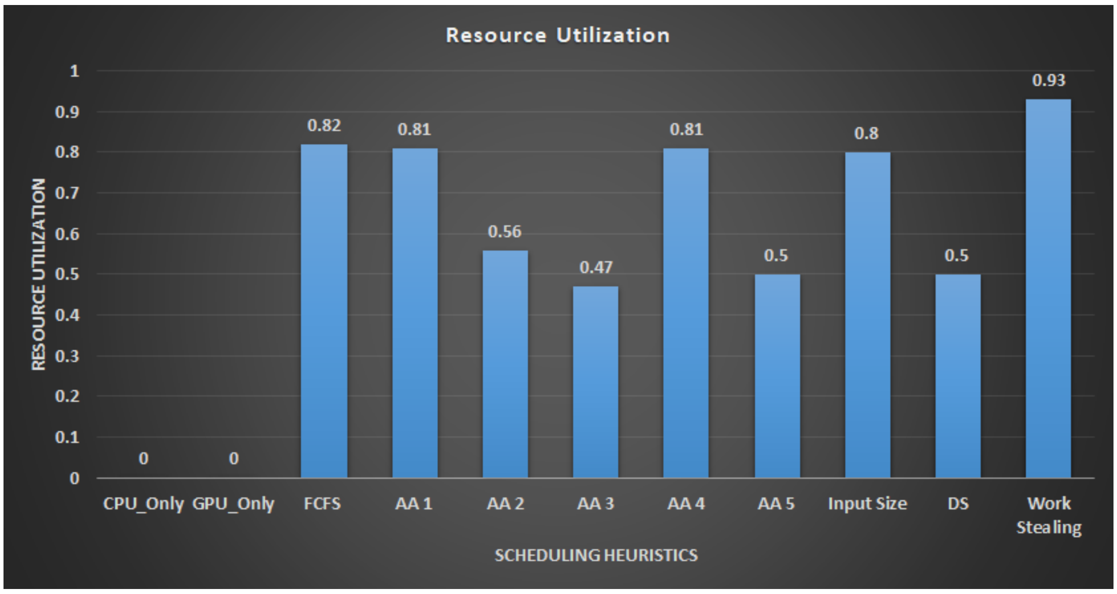


FIGURE 4.15: Resource Utilization of Scheduling Heuristics

When the jobs are executed on first come first serve bases 82% utilization of devices occur which is still less than the proposed scheduler. We performed the experiment of assigning alternate jobs to CPU and GPU device five times. Each time shuffle all the jobs randomly and assigned each job alternatively. The alternate assignment 1 utilizes 81% of the processing devices. The alternate assignment 2 only utilizes 56% of the processing devices. The alternate assignment 3 utilizes only 47% of the processing devices which is the least utilization of the devices. The alternate assignment 4 also utilizes 81% of the processing devices. The alternate assignment 5 utilizes 50% of the processing devices. The scheduler on input base utilizes the devices up to 80%. The device suitability have only utilized 50% the processing devices. Our proposed scheduler have utilized 93% of the devices which is the highest of all prescribed schedulers. The figure 4.16 shows the improvement of time of each scheduling scheme with respect to the proposed work stealing scheduler. The figure shows that none of the scheduling schemes have

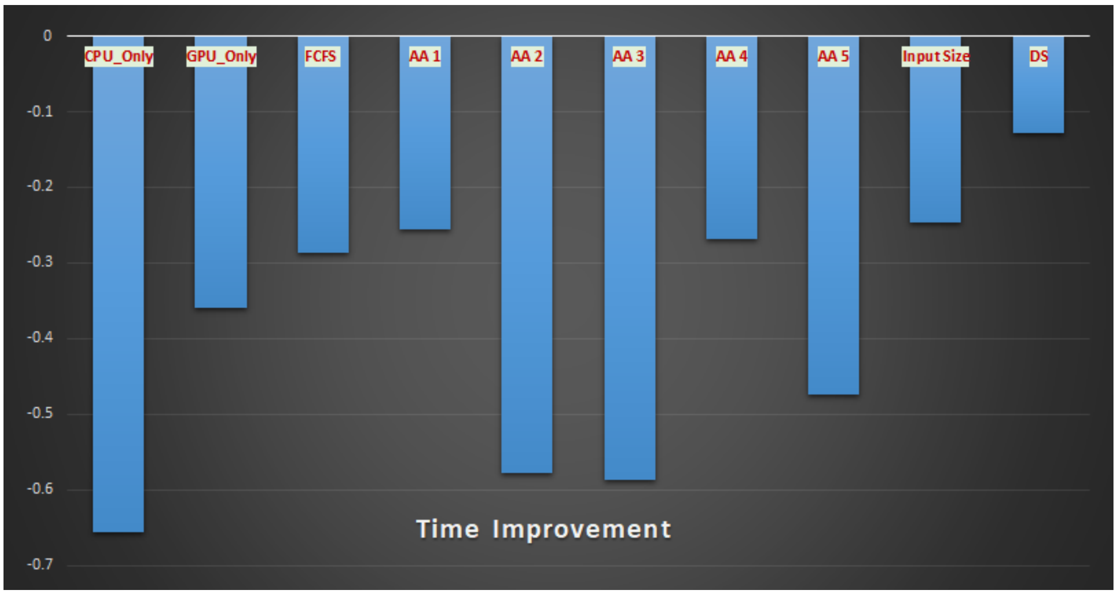


Figure 4.16: Time Improvement of Scheduling Heuristics w-r-t Proposed Scheduler

improved the time from the proposed work stealing scheduler. All the schedulers have a negative value of time improvement with respect to proposed work stealing scheduler, which indicates that these schedulers have not exceeded from the work stealing scheduler in time improvement. The CPU_Only have the worst improvement of execution time of jobs. The device suitability have more improvement in the execution time of jobs than other schedulers.

The figure 4.17 shows the throughput of the work stealing scheduler with other heuristics. Throughput is the ratio between the total number of programs and the maximum execution time of programs. The graph shows that the work stealing scheduler have maximum throughput than every other scheduling scheme. The reason behind the maximum throughput is that the proposed scheduler utilizes the executing devices. The proposed work-stealing scheduler has 65.5% increase in throughput from CPU_Only, 35.9% from GPU_Only, 28.7% from FCFS, 25.5%

maximum throughput from Alternate Assignment 1, 57.7% from AA2, 58.6% from AA3, 27% from AA4, 47.4% from AA5, 24.7% improvement from Input Size, and only 12.9% improvement from Device Suitability.

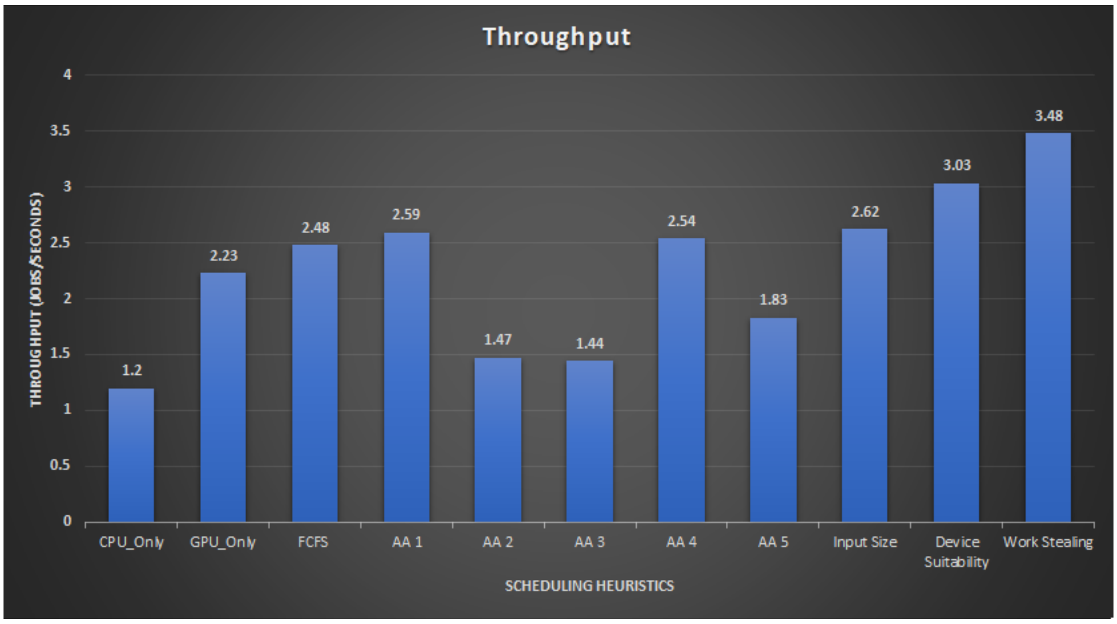


FIGURE 4.17: Throughput of Scheduling Heuristics

4.8 Performance Discussion

The experimental results in figure 4.14 and 4.15 shows that the proposed scheduling scheme outperforms all other schemes in the form of overall execution time of job pool and utilization of processing devices. The performance improvement of the scheduling scheme is due to the usage of the machine learning for the prediction of execution time. As the execution time of each job in the job pool is predicted already through machine learning, the jobs are arranged in the job pool with respect to the execution time. For load balancing the work stealing concept is used through which the utilization of the processing devices takes place. The proposed scheduling scheme have less execution time of jobs on CPU except the device suitability and less execution time of jobs on GPU except alternate assignment number 5. The work stealing scheduler outperforms all scheduling schemes in utilization of the processing devices.

Furthermore the research questions were following

1. How to analyze optimization techniques for execution time predictors through machine learning?
   (a) Which set of features play an important role to predict data-parallel application execution time?
2. How to design and develop a load-balanced scheduler to achieve minimal execution time, maximal throughput, and improved resource utilization?

The section 4.6 covers the question 1. Three machine learning models multiple linear regression, gradient boosting regression, and random forest regression models were implemented for the prediction of the execution time of the jobs on CPU device and GPU device. Among them the random forest model achieve the maximum results. The part a) of the question 1 is covered by section 4.4 and section 4.5. These sections covers the correlation of features with each other and importance of the features with-respect-to the class label. The table 4.10 shows that using all the features causes an increase in the evaluation metrics values while using only important features, improve the results except multiple linear regression. The section 4.7 covers the answer of question 2. Section 4.7 have briefly described the work stealing scheduler, their evaluation, and comparison of the execution time, resource utilization, and throughput with other scheduling heuristics. It shows that the proposed work-stealing scheduler have 65.6% less execution time, 93.3% utilization of the resources, and 65.5% increase in throughput.

4.9 Comparison with State-of-the-art Techniques

The section 4.8 briefly discuss the performance of the work stealing scheduler and provide answers to the research questions stated in chapter 1. The table 4.13 shows the comparison of the work-stealing scheduler result with other state-of-the-art techniques. The results shows that the proposed model has overcome most of the heuristics in the execution time and utilization of the devices. The proposed model has more reduction in the execution time than Khalid et al.[4], Ahmed et al.[5], and Khalid et al.[13].

Table 4.13: Comparison of Work Stealing Performance with other Models

Ref	**App**	**F**	**BM**	**Performance**	**ML**
Proposed	120	23	AMD Polybench	R^2:0.99, 65.5% reduction in exec time, 93% utilization	MLR, Random Forest, Gradient Boosting
[4] Khalid et al. 2018	18	-	AMD parboil Polybench Rodinia	8.1% reduction in exec time. 7.07% throughput	-
[5] Ahmed et al. 2019	930	30	AMD Polybench Own	31% reduction in exec time. 67.8% utilization. 147.35% throughput	Gradient Boosting
[13] Khalid et al. 2019	199	23	AMD Parboil Polybench Rodinia	38% reduction in execution time	Random Forest, Gradient Boosting
[3] Ahmed et al. 2021	930	23	AMD Polybench	R^2:0.76, F-measure: 0.91	Decision Tree, Naive Bayes, Random Forest, KNN

F = Feature

BM = Benchmark

ML = Machine Learning

Chapter 5

Conclusion and Future Work

The programmer maps jobs to processing devices in a heterogeneous system. They map suitable job to a suitable device which causes devices idleness and increase in execution time. The scheduler are used for mapping jobs to executing devices. The decision of the jobs assignment should be balanced to achieve the maximum utilization of devices and reduction in execution time of overall jobs. This is a hard task for programmers to decide about the jobs in heterogeneous environment. In this research thesis, a novel job scheduling scheme is introduced which is based on machine learning execution time prediction and uses the concept of work stealing algorithm. The scheduler uses the predicted execution time of a job on CPU and GPU device and schedule the jobs to the processing devices considering the utilization of devices. The main contribution of the research is as follow;

1. Analysis of multiple scheduling heuristics in heterogeneous environment.
2. A machine learning based model development for the prediction of the jobs execution time on a CPU device and also on GPU device.
3. A scheduling heuristic that consider the execution time of the overall jobs in the job pool and the utilization of the processing devices.
4. Comparison and analysis of the proposed scheduler with other scheduling techniques in the form of execution time and utilization of resources.

5.1 Future Work

In this research thesis, a Load-Balanced Task Scheduler for Heterogeneous Systems based on Machine Learning is proposed that distributes the OpenCL applications among the processing devices, considering the reduction of the overall execution time of jobs in the job pool and the utilization of the processing devices. The proposed scheduler maps only static jobs to the processing devices and considers only the integrated Graphics processing units. In future, this framework can be extended to dynamic allocation of jobs to processing devices and usage of the discrete graphic processing units.

Bibliography

[1] N. Tsog, M. Becker, F. Bruhn, M. Behnam, and M. Sjödin, "Static allocation of parallel tasks to improve schedulability in cpu-gpu heterogeneous real-time systems," in *IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, vol. 1, pp. 4516–4522, IEEE, 2019.

[2] Y. Wen, Z. Wang, and M. F. O'boyle, "Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms," in *2014 21st International conference on high performance computing (HiPC)*, pp. 1–10, IEEE, 2014.

[3] U. Ahmed, J. C.-W. Lin, G. Srivastava, and M. Aleem, "A load balance multi-scheduling model for opencl kernel tasks in an integrated cluster," *Soft Computing*, vol. 25, no. 1, pp. 407–420, 2021.

[4] Y. N. Khalid, M. Aleem, R. Prodan, M. A. Iqbal, and M. A. Islam, "E-osched: a load balancing scheduler for heterogeneous multicores," *The Journal of Supercomputing*, vol. 74, no. 10, pp. 5399–5431, 2018.

[5] U. Ahmed, M. Aleem, Y. Noman Khalid, M. Arshad Islam, and M. Azhar Iqbal, "Ralb-hc: A resource-aware load balancer for heterogeneous cluster," *Concurrency and Computation: Practice and Experience*, p. e5606, 2019.

[6] T. Wenjie, Y. Yiping, Z. Feng, L. Tianlin, and S. Xiao, "A work-stealing based dynamic load balancing algorithm for conservative parallel discrete event simulation," in *2017 Winter Simulation Conference (WSC)*, pp. 798–809, IEEE, 2017.

[7] M. E. Belviranli, L. N. Bhuyan, and R. Gupta, "A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 4, pp. 1–20, 2013.

[8] H. J. Choi, D. O. Son, S. G. Kang, J. M. Kim, H.-H. Lee, and C. H. Kim, "An efficient scheduling scheme using estimated execution time for heterogeneous computing systems," *The Journal of Supercomputing*, vol. 65, no. 2, pp. 886–902, 2013.

[9] R. Kaleem, R. Barik, T. Shpeisman, C. Hu, B. T. Lewis, and K. Pingali, "Adaptive heterogeneous scheduling for integrated gpus," in *2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)*, pp. 151–162, IEEE, 2014.

[10] A. Ghose, L. Dokara, S. Dey, and P. Mitra, "A framework for opencl task scheduling on heterogeneous multicores," *Parallel Processing Letters*, vol. 27, no. 03n04, p. 1750008, 2017.

[11] K. Moren and D. Göhringer, "Automatic mapping for opencl-programs on cpu/gpu heterogeneous platforms," in *International Conference on Computational Science*, pp. 301–314, Springer, 2018.

[12] B. Taylor, V. S. Marco, and Z. Wang, "Adaptive optimization for opencl programs on embedded heterogeneous systems," *ACM SIGPLAN Notices*, vol. 52, no. 5, pp. 11–20, 2017.

[13] Y. N. Khalid, M. Aleem, U. Ahmed, M. A. Islam, and M. A. Iqbal, "Troodon: A machine-learning based load-balancing application scheduler for cpu–gpu system," *Journal of Parallel and Distributed Computing*, vol. 132, pp. 79–94, 2019.

[14] D. Grewe and M. F. O'Boyle, "A static task partitioning approach for heterogeneous systems using opencl," in *International conference on compiler construction*, pp. 286–305, Springer, 2011.

[15] M. Boyer, K. Skadron, S. Che, and N. Jayasena, "Load balancing in a changing world: dealing with heterogeneity and performance variability," in *Proceedings of the ACM International Conference on Computing Frontiers*, pp. 1–10, 2013.

[16] M. Becchi, S. Byna, S. Cadambi, and S. Chakradhar, "Data-aware scheduling of legacy kernels on heterogeneous platforms with distributed memory," in *Proceedings of the twenty-second annual ACM symposium on Parallelism in algorithms and architectures*, pp. 82–91, 2010.

[17] J. Lee, M. Samadi, and S. Mahlke, "Orchestrating multiple data-parallel kernels on multiple devices," in *2015 International Conference on Parallel Architecture and Compilation (PACT)*, pp. 355–366, IEEE, 2015.

[18] D. Grewe, Z. Wang, and M. F. O'Boyle, "Opencl task partitioning in the presence of gpu contention," in *International Workshop on Languages and Compilers for Parallel Computing*, pp. 87–101, Springer, 2013.

[19] A. Munshi, "The opencl specification," in *2009 IEEE Hot Chips 21 Symposium (HCS)*, pp. 1–314, IEEE, 2009.

[20] Y. Wen and M. F. O'Boyle, "Merge or separate? multi-job scheduling for opencl kernels on cpu/gpu platforms," in *Proceedings of the general purpose GPUs*, pp. 22–31, 2017.

[21] Z. Wang and M. O'Boyle, "Machine learning in compiler optimization," *Proceedings of the IEEE*, vol. 106, no. 11, pp. 1879–1901, 2018.

[22] S. Alsubaihi and J.-L. Gaudiot, "A runtime workload distribution with resource allocation for cpu-gpu heterogeneous systems," in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pp. 994–1003, IEEE, 2017.

[23] Z. Wang, L. Zheng, Q. Chen, and M. Guo, "Cap: co-scheduling based on asymptotic profiling in cpu+ gpu hybrid systems," in *proceedings of the 2013*

international workshop on programming models and applications for multicores and Manycores, pp. 107–114, 2013.

[24] J. Lee, M. Samadi, Y. Park, and S. Mahlke, "Skmd: Single kernel on multiple devices for transparent cpu-gpu collaboration," *ACM Transactions on Computer Systems (TOCS)*, vol. 33, no. 3, pp. 1–27, 2015.

[25] P. Huchant, M.-C. Counilh, and D. Barthou, "Automatic opencl task adaptation for heterogeneous architectures," in *European conference on parallel processing*, pp. 684–696, Springer, 2016.

[26] O. E. Albayrak, I. Akturk, and O. Ozturk, "Effective kernel mapping for opencl applications in heterogeneous platforms," in *2012 41st International Conference on Parallel Processing Workshops*, pp. 81–88, IEEE, 2012.

[27] X. Liu, H.-A. Ounifi, A. Gherbi, W. Li, and M. Cheriet, "A hybrid gpu-fpga based design methodology for enhancing machine learning applications performance," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–15, 2019.

[28] M. A. D. Guzman, R. Nozal, R. G. Tejero, M. Villarroya-Gaudo, D. S. Gracia, and J. L. Bosque, "Cooperative cpu, gpu, and fpga heterogeneous execution with enginecl," *The Journal of Supercomputing*, vol. 75, no. 3, pp. 1732–1746, 2019.

[29] E. Canhasi, "Evaluating the efficiency of cpus, gpus and fpgas on a near-duplicate document detection via opencl.," *J. Comput. Sci.*, vol. 14, no. 5, pp. 699–704, 2018.

[30] A. Navarro, F. Corbera, A. Rodriguez, A. Vilches, and R. Asenjo, "Heterogeneous parallel_for template for cpu–gpu chips," *International Journal of Parallel Programming*, vol. 47, no. 2, pp. 213–233, 2019.

[31] T. Wenjie, Y. Yiping, Z. Feng, L. Tianlin, and S. Xiao, "A work-stealing based dynamic load balancing algorithm for conservative parallel discrete event simulation," in *2017 Winter Simulation Conference (WSC)*, pp. 798–809, IEEE, 2017.

[32] Y. Wang, W. Ji, F. Shi, and Q. Zuo, "A work-stealing scheduling framework supporting fault tolerance," in *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 695–700, IEEE, 2013.

[33] F. Kreiliger, J. Matejka, M. Sojka, and Z. Hanzálek, "Experiments for predictable execution of gpu kernels," *OSPERT 2019*, p. 23, 2019.

[34] S. Singh and P. Gupta, "Comparative study id3, cart and c4. 5 decision tree algorithm: a survey," *International Journal of Advanced Information Science and Technology (IJAIST)*, vol. 27, no. 27, pp. 97–103, 2014.