Soft Computing
https://doi.org/10.1007/s00500-020-05152-8

METHODOLOGIES AND APPLICATION

A load balance multi-scheduling model for OpenCL kernel tasks in an integrated cluster

Usman Ahmed[1] · Jerry Chun-Wei Lin[1] · Gautam Srivastava[2,3] · Muhammad Aleem[4]

Abstract
Nowadays, embedded systems are comprised of heterogeneous multi-core architectures, i.e., CPUs and GPUs. If the application is mapped to an appropriate processing core, then these architectures provide many performance benefits to applications. Typically, programmers map sequential applications to CPU and parallel applications to GPU. The task mapping becomes challenging because of the usage of evolving and complex CPU- and GPU-based architectures. This paper presents an approach to map the OpenCL application to heterogeneous multi-core architecture by determining the application suitability and processing capability. The classification is achieved by developing a machine learning-based device suitability classifier that predicts which processor has the highest computational compatibility to run OpenCL applications. In this paper, 20 distinct features are proposed that are extracted by using the developed LLVM-based static analyzer. In order to select the best subset of features, feature selection is performed by using both correlation analysis and the feature importance method. For the class imbalance problem, we use and compare synthetic minority over-sampling method with and without feature selection. Instead of hand-tuning the machine learning classifier, we use the tree-based pipeline optimization method to select the best classifier and its hyper-parameter. We then compare the optimized selected method with traditional algorithms, i.e., random forest, decision tree, Naïve Bayes and KNN. We apply our novel approach on extensively used OpenCL benchmarks, i.e., AMD and Polybench. The dataset contains 653 training and 277 testing applications. We test the classification results using four performance metrics, i.e., F-measure, precision, recall and R^2. The optimized and reduced feature subset model achieved a high F-measure of 0.91 and R^2 of 0.76. The proposed framework automatically distributes the workload based on the application requirement and processor compatibility.

Keywords Machine learning · Classification · Feature selection · OpenCL · Optimization

Communicated by V. Loia.

✉ Jerry Chun-Wei Lin
jerrylin@ieee.org
http://www.ikelab.net

Usman Ahmed
usman.ahmed@hvl.no
https://www.researchgate.net/profile/Usman_Ahmed24

Gautam Srivastava
srivastavag@brandonu.ca

Muhammad Aleem
m.aleem@nu.edu.pk

[1] Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway

[2] Department of Mathematics and Computer Science, Brandon University, Brandon, Canada

[3] Research Centre for Interneural Computing, China Medical University, Taichung City, Taiwan

[4] National University of Computer and Emerging Sciences, Islamabad, Pakistan

Published online: 15 July 2020

1 Introduction

Today, most systems are equipped with multi-core processors. Due to power consumption and transistor density constraints, the ever-increasing clock frequency trend is no longer possible (Stone et al. 2019; Wen et al. 2014). Therefore, multi-core architectures have been developed as a solution to problems like power consumption, heat dissipation and transistor density (Wen et al. 2014). In multi-core architectures, multiple identical CPUs are integrated on the same integrated circuit. The system has the same type of processor called a *homogeneous system*. The hardware manufacturer increases computing power (in terms of parallelism) by increasing the number of cores. Nowadays, developers use parallel programming to speed up applications (Stone et al. 2019; Krishna et al. 2013; Wen et al. 2014). An application is partitioned into parallel portions, each executing on a separate processor core. The parallel framework has further been strengthened by utilizing a specialized processing unit having many-cores, such as a graphical processing unit (GPU). The multi-core CPUs and many-core GPUs trend has initiated a new paradigm for computation processing called *heterogeneous computing*. Heterogeneous system architecture (HSA) systems utilize multiple processor types (CPUs and GPUs). HSAs are defined as systems that have multiple cores and are able to increase performance not only through the act of adding additional cores but also through working with special capabilities to handle more complex tasks while also being able to maintain a high level of energy efficiency. Due to immense data generation and huge processing power, the new application generating workloads with diverse requirements. The central processing unit (CPU) is unable to handle these diverse requirements. Heterogeneous computing, however, is designed to help and enable the efficient use of diverse processors like the CPU and GPU to handle these new emerging workloads efficiently. Intelligently utilizing the diverse processors helps and enables new experiences while maximizing throughput and reducing turnaround time. Employing diverse processors provides various opportunities to find at least the best combinations that will truly excel at completing a particular workload. Some processors are rather inefficient at specific jobs while excelling others. Once we realize that each type of processor has its strength, we can opportunistically and intelligently choose the appropriate one for the specific workload. With the help of heterogeneous computing, different processors can be designed to work together, enabling new user experiences.

1.1 OpenCL: heterogeneous programming framework

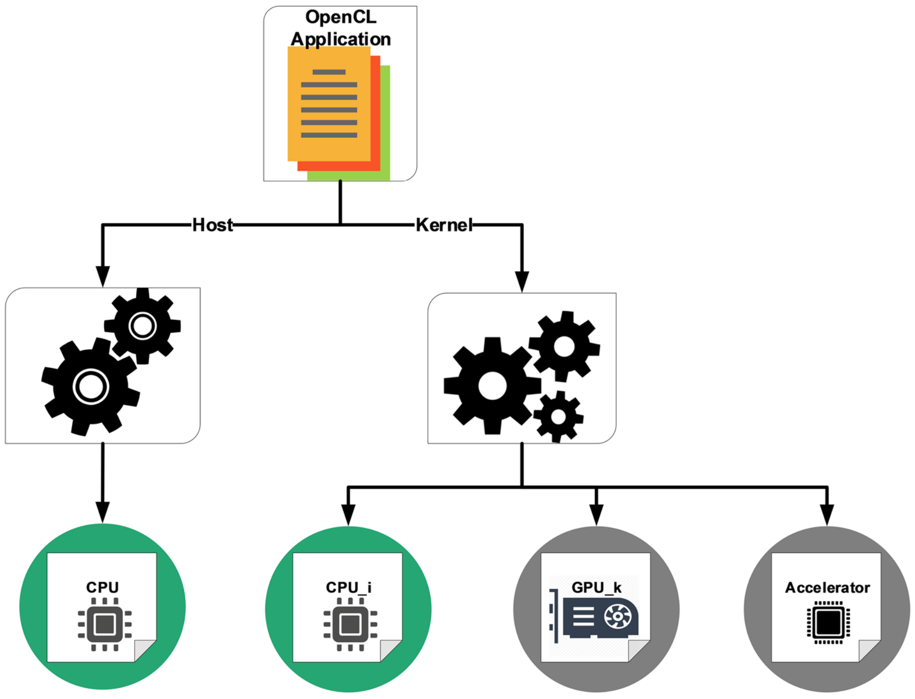
Open compute language (OpenCL) (Stone et al. 2019) was developed to execute parallel tasks on heterogeneous multi-core architectures. The OpenCL framework is being supported by major vendors in the hardware industry, i.e., NVIDIA, AMD and Intel. Figure 1 describes the OpenCL application execution model. The serial portion of the OpenCL program is executed from the host using the CPU, and data-parallel task (kernel) is executed on the accelerator device, i.e., GPU or CPU. This framework provides consistent execution over the entire heterogeneous core. In comparison with the CPU device, some applications perform better on the GPU device. The reason is its parallel nature. However, some applications also perform better on the CPU because of the sequential nature of the task. Programmers will usually assign processes to one of the GPU/CPU, and thus, the other unit will remain idle. As an example, if tasks are assigned to a GPU device, this leaves the CPU idle waiting for the scheduled tasks to complete, as shown in Fig. 1. In this research, we will be using OpenCL because of its portability and a large number of supported compute-devices.

1.2 Application scheduling on heterogeneous machines

The act of scheduling in reference to heterogeneous machines has previously been studied extensively. We have also seen many interesting solutions proposed (Krishna et al. 2013; Ghose et al. 2016; Grewe and O'Boyle 2011; Luk et al. 2009). The scheduler decides a particular data-parallel application should be assigned to which accelerator in heterogeneous architecture. The proposed schedules are only worthwhile when a known amount of work prior to being executed is available (Grewe and O'Boyle 2011; Luk et al. 2009; Grewe et al. 2013). As a general rule, scheduling algorithms, for the most part, carry little overhead but do not always provide optimal task partitioning. Some researchers perform task mapping to a compute-unit at runtime. The main advantage of runtime task scheduling is that the decision to map the task is more optimal (considering the runtime attributes of the application and machine) (Belviranli et al. 2013; Choi et al. 2013; Gregg et al. 2010; Ravi and Agrawal 2010; Augonnet et al. 2011). Scheduling decisions can be adjusted during the execution of a program. Significant cons of run time scheduling are the increased complexity and higher scheduling overhead.

The supervised machine learning model has also been used and proven to be useful in learning optimized

Fig. 1 OpenCL task execution



scheduling (Ahmed et al. 2019b). Code features are used to characterize an application (Grewe and O'Boyle 2011; Ghose et al. 2017). The code features include the number of instructions as well as parallel runtime parameters such as the number of work items. At compile time, an application abstract syntax tree is generated by using a compiler named as CLang and LLVM (Lattner 2008). The abstract syntax tree gives information about application behavior as follows:

1. number and type of operations used in the application
2. count of barrier occurrences
3. number of blocks within the application
4. count for the load operation performed by the application
5. count of store operations performed by the application (Wen et al. 2014)

The count of each code feature (number and type of operation, barrier, blocks, load/store operation) in an application is used as features values. The features in the feature vector are classified into two types, i.e., static features and dynamic features. Static code features such as the number of `int` operations and local memory access percentage are extracted at compile time, while dynamic features extracted include input workload. All feature values combine to form a feature vector. These values are then used as input to a predictive model that is based on machine learning. The predictor is trained on the extracted feature vectors. The features are selected based on their contribution to predicting the output. The motivation behind using a reduced feature set (for predictive models) is therefore to reduce the amount of redundant data, which in turn reduces over-fitting issue, as well as improving accuracy, and finally decreasing training time.

1.3 Motivation

The data-parallel application attains higher performance for GPU-based execution. At the same time, there are some scientific applications (i.e., dot product or bread first search) that are inadequately performed on GPUs. The same applications often attain varying performance for different input data sizes (Khalid et al. 2019; Ahmed et al. 2019a). The applications that attain less gain should take a different strategy based on its input dimensions and type of operations. As allocating all the applications on GPU will result in the load imbalance and suboptimal execution time for a job pool. Therefore, the recognition of application type and its computing processor are significant (Sakhnini et al. 2019).

The motivation behind using device suitability is that it helps to map the job to specific machine (containing more suitable device with respect to applications code features) in the cluster and then the more suitable device (may not the faster one), i.e., a CPU or a GPU is selected for mapping. In Fig. 2, three scheduling schemes have been shown (i.e., machine learning-based device suitability model (Single node—heterogenous processor, i.e., CPU and GPU), GPU only (using only GPU) and Oracle (best device mapping). We run OpenCL job pool of 6 application

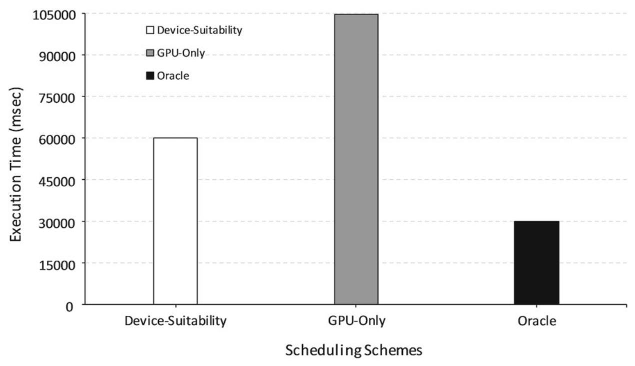


Fig. 2 Motivation behind the usage of device suitability classification model. Device suitability: *scheduling by using machine learning classification*, GPU only: *scheduling only on GPU device* and Oracle heuristic: *scheduling the application to best-known device*

samples with the already mentioned scheduling schemes. Two applications in the job pool are suitable for the CPU (i.e., GEMM and Matrix-Vector Multiplication) and remaining four are suitable for the GPU (i.e., Matrix Multiplication, Bitonic Sort, Monte Carlo Asian DP, Black Scholes). The device suitability scheduling scheme assigned the jobs by predicting the suitable hardware resource. The GPU-only setting assigns it by running all application on GPU only. Finally, *Oracle* assigns the best-known setting for each device. The Oracle-based scheduling performs 3.5× better than GPU only, whereas has 2× better performance in comparison to device suitability predictor. However, if the number of heterogeneous devices exceeds from a single CPU to *i*th CPU and single GPU to *j*th GPU, then the mapping becomes a very difficult job. In this study, we proposed a multi-node device suitability model that optimally maps job among multiple devices using machine learning.

The execution time of the job pool can be reduced if the data-parallel application mapped to appropriate devices. The application cannot be mapped based on the arrival time or free resources. This may cause load imbalance and longer execution time. The application requirement should be considered to map the application to the appropriate device, i.e., computation requirement of the program, data size and number of instructions. The data-parallel application can have a lower execution time on a GPU while the sequential application has a lower execution time on CPU, which shows that smart mapping of the applications is required. Therefore, there is a need for a scheduling mechanism, which automatically maps the application to the proper device by utilizing the application as well as the hardware requirement of the submitted application. From Fig. 2, it can be concluded that optimize scheduling method can be designed by considering the device suitability. The designed scheduling methods result in lower execution time of the submitted applications.

Most of the methods required a data-parallel application code splitting overhead to split tasks among CPU & GPU device. This data-parallel application code splitting will result in additional time overhead. The existing solution proposed a profiling-based scheduling method. They used code instrumentation and profiled time to the scheduled application to a processing unit. This profiling required time overhead.

In particular, the following are the main contributions of the research:

1. A mechanism to extract the set of features that plays an important role to predict data-parallel application device suitability.
2. A unified framework to develop a machine learning-based classifier to predict the suitable processing unit in a heterogeneous cluster.
3. Analyze optimization technique to design device suitability classifier.
4. A demonstration of data imbalance problem and its solution.

2 Literature review

Task scheduling is a non-trivial problem that requires optimal mapping of tasks to the processor so that the overall execution time of applications is reduced. Scheduling decisions become more complicated when we have a heterogeneous cluster in which each compute-unit has a diversified set of characteristics. The heterogeneous multi-core architecture comprises different processors, i.e., central processing units (CPUs) and graphics processing units (GPUs). The applications required to perform latency-sensitive tasks require execution on the CPU as it takes advantage of out-of-order execution, branch-prediction and scalar capabilities (Lee et al. 2010). The GPU has multi-threading capabilities, so an application that requires performing parallel tasks will use the GPU (multiple core architecture) (Hechtman and Sorin 2013). The CPU has a limited number of powerful and complex cores that are generalized to execute different types of applications efficiently. In contrast, the GPU contains a large number of simplified cores that are mainly specialized to execute data-parallel portions of the program.

Therefore, while scheduling the heterogeneity of computing devices mapping computation to processors effectively should also be considered. We have seen a collection of researchers propose scheduling algorithms for heterogeneous platforms (Luk et al. 2009; Becchi et al. 2010; Huchant et al. 2016; Pérez et al. 2016; Ravi et al. 2012). In a network, specific servers collectively composed to perform a particular task. The allocation of task on those resources should generally be based upon criteria for the highest priority level provided that each such resource can act as an individual agent. However, the main challenges arise in the distribution of task and services (Iftikha and Jangsher 2019; Aloqaily 2019; Daraghmeh et al. 2019). Many of the papers use the notion of splitting data-parallel application between the CPU/GPU, while many others have improved throughput and resource utilization by scheduling pools of applications. The machine learning-based predictive modeling is considered to be a powerful method for optimizing parallel programs (Grewe and O'Boyle 2011; Ghose et al. 2017; Kofler et al. 2013; Wen and O'Boyle 2017; Taylor et al. 2017; Ahmed et al. 2019b). The predictive model is trained to learn from its set of examples and have adaptive behavior for varying platforms. By using the scheduling technique (Grewe and O'Boyle 2011; Ghose et al. 2017; Kofler et al. 2013; Wen and O'Boyle 2017; Taylor et al. 2017), severe load imbalance is introduced between CPU_i and GPU_k due to CPU_i only managing execution of kernel on GPU_k and taking no part in actual computation. The idle time that CPU_i spent while waiting for GPU_k to complete kernels execution is not desirable. Ideally, a schedule is required that can schedule the data-parallel application to both CPU and GPU in such a way that all processors in the cluster can complete processing at the same time. In this way, energy consumption and heat dissipation due to idling processor are reduced but, more importantly, the execution time of Job Pool will also be reduced significantly. The optimal device selection is key for any scheduler schemes in a heterogeneous environment.

Pérez et al. (2016) gave a Maat library which performs load balancing of a single kernel. According to Perez et al., the programmer does not optimally utilize heterogeneity as they consider the CPU device for sequential tasks, whereas GPU is a parallel task. Known as an inflexible approach eventually leads to wastage of computational power (Pérez et al. 2016). Through the use of Maat, the user will need to through the kernel program build up a parallel version, which selects a load balancing method and runs it on all the available resources. In Pérez et al. (2016) approach, there is no need for extra programming effort, as the same raw kernel code is utilized. Moreover, at runtime, the predictive model can determine device suitability, as well as application time estimation, is made to achieve maximal throughput.

Luk et al. (2009) have also addressed the problem of optimizing the utilization of available resources. They have focused on the need for automated mapping of processing elements to the available resources (Luk et al. 2009). According to Luk et al. (2009), programmer utilization of heterogeneous platform can adapt according to hardware/software configuration. Therefore, a system name Qilin is proposed that utilizes machine learning to classify kernel code. The kernel code is partitioned into the CPU and GPU device. The Qilin shows the execution time of the applications in the database. The recorded information is then utilized by the Qilin to project execution time of new arrived application and to schedule it accordingly. Whenever the hardware configuration changes, Qilin initiates a new training session. The Qilin requires offline profiling and code partition overhead, whereas the proposed method does not require these overheads.

Huchant et al. (2016) have proposed an automatic runtime technique that schedules OpenCL kernel code across Heterogeneous devices. Huchant et al. (2016) have given a technique that can solve issues causing from the heterogeneity, i.e., communications, load balancing and issues caused by the iterative computation. The technique is divided into two main approaches, i.e., static and dynamic. In the static phase, kernel code is transformed into partition ready kernels, which are then mapped into different devices. The execution time of the mapped kernel is noted and then in a dynamic phase, queuing off the partitioned kernel is adjusted to achieve optimized throughput. This technique differs from our technique as it mapped an OpenCL kernel which is single. However, it is noted here that our work manages to schedule OpenCL applications as a pool.

Albayrak et al. (2012) have addressed the need for optimal mapping among different heterogeneous devices, CPU or GPU. According to the authors, in a multi-application environment, different kernels exhibit different characteristics. Some of them run faster on the GPU; others may refer to execute on CPU due to data transfer cost. However, there is a need to map the kernel to the proper device to improve the overall performance of an application. Albayrak et al. (2012) have proposed a profiling-based scheduling method to map OpenCL application (Albayrak et al. 2012). The data dependencies and execution time are profiled. Then, by the use of a greedy algorithm, the kernel is scheduled to device, i.e., CPU or GPU. The proposed algorithm can achieve the optimal result for scheduling multiple kernels of a single application only. However, the proposed scheme does not require offline profiling overhead. The method can schedule single and multi-kernel application within a batch of the job pool.

In a similar study entitled "A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures", Belviranli et al. examined the issues in resource utilization of the heterogeneous environment. They proposed a scheduling mechanism named as HDSS (Belviranli et al. 2013). It partitions the workload among processing units, i.e., CPU and GPU. This results in improvement of kernel execution time. HDSS has two phases, i.e., profiling phase and adaptive phase. The computation power of each processing unit is evaluated by assigning the same number of loop operations in the profiling phase, while remaining loop operation is assigned based on the processing speed in the adaptive phase. Both phases help in balancing the load on heterogeneous computing devices. The proposed method is not dependent on job splitting and any kind of raw code transformation.

Heterogeneous computing systems get improved performance by utilizing the powerful CPU as well as the GPU. The device selection is the most critical factors in determining the performance of application (Choi et al. 2013). Therefore, Choi et al. have estimated the execution time, which determines the schedule of the application on a CPU or a GPU device. The model requires an execution history of application to train and predict application, which has finished the job earlier. The total execution time of the application (on that device) and the execution time of the currently executing application are used to estimate finish time of an application.

Grewe and O'Boyle (2011) addressed the prediction of suitable processing helps to achieve optimized results. The author proposes the partitioning mechanism for OpenCL program. They extract the static features during compile time. Then pre-trained model SVM is utilized to predict whether to map a kernel to a CPU, GPU or to partition the kernel among available computing devices. The authors' *GPU-only model* achieves 91% accuracy, whereas the *CPU-only model* achieved 95% accuracy.

Ghose et al. (2016) give a model that analyzes the branch divergence and use that analysis as a code feature. The trained model achieved the accuracy of 89% to predict the CPU–GPU inclusive application and 81.23% to predict the application to be partitioned. However, the proposed scheduler ensures the multi-node scheduling of tasks. Kofler et al. (2013) proposed an ANN-based predictive model. The primary task of a predictive model is to dynamically partition the given task on a CPU and a GPU. Kofler et al. (2013) used Insieme source to source compiler to translate a kernel code into multi-device kernel code. The dynamic partition is based on the artificial neural network (ANN) predictive model. The feature set includes static code features and dynamic input sensitive features (e.g., data-transfer size of the split-able buffer). The partitioning task is further to improve from 2 to 7% by using principal component analysis. The test set achieved 87.5% results. The authors have partitioned the program and achieved high accuracy. Our proposed scheduler selects an optimal device as well as do a scheduled task on a cluster of devices by using the application device performance of the selected device. Moreover, the proposed schedule does not require kernel splitting.

Wen and O'Boyle (2017) address that certain application performance is maximized when assigned to the single computing device and sometimes sharing among computing device results in improved performance. The author's predictive models determine whether an application kernel required to combine with other kernels or not. The model also uses code static and dynamic features. The decision tree classification model is used to trained and then classify the kernel to a suitable device, i.e., CPU or GPU. The second's classification model determine whether to run the kernel on a single device or merge it with another kernel.

Tsog et al. (2019) use a static allocation-based method to map sequential application on CPU and parallel on GPU. The authors' model was able to balance load among CPU and GPU. However, the static approach required offline profiling that increases execution time. Moreover, the profiling becomes more complicated when allocation is required to be performed under multi-heterogeneous nodes. In another study (Alizadeh and Momtazpour 2020), Alizadeh et al. proposed a scheduling mechanism to characterize the kernels and then predict the concurrent execution. The model can achieve a high accuracy of 91.7%. However, the mode able to achieve high accuracy on a single node where prediction is made between CPU and GPU. The task becomes problematic on a distributed system that involves parallel execution of applications among many CPUs and many GPUs.

Khan et al. (2019) present a heuristic-based scheduling mechanisms that reduce the execution time of the cluster. The smart scheduler presented does the code instrumentation and divides the application load among different nodes. Their results showed an improved throughput of the system. However, the code instrumentation required additional time as it split the kernel code and divides them among different machines in different time constraints.

2.1 Critical analysis

After the comprehensive analysis of state-of-the-art approaches, techniques for heterogeneous scheduling on a heterogeneous machine were found. The majority of heterogeneous scheduling schemes do not address the problem of overloading, which results in longer execution time and low resource utilization (Wen et al. 2014; Taylor et al. 2017; Ghose et al. 2016; Wen and O'Boyle 2017; Kofler et al. 2013; Grewe and O'Boyle 2011; Alizadeh and Momtazpour 2020). There are several techniques that do not consider device suitability, and this often results in low resource utilization (Luk et al. 2009; Becchi et al. 2010; Huchant et al. 2016; Pérez et al. 2016; Ravi et al. 2012). A few techniques use the machine learning approach to predict the suitable device and then split the kernel code among the CPU and GPU (Grewe and O'Boyle 2011; Kofler et al. 2013; Wen and O'Boyle 2017; Taylor et al. 2017). The kernel splitting overhead required additional time overhead. Most of the research uses default classification settings. Default settings mostly result in under performance of the machine learning classification model (Ahmed et al. 2019a; Ahmed et al. 2020, Amrollahi et al. 2020): the feature extraction, selection, model selection, hyper-parameter tuning and evaluation need to be optimized. Most research avenues do not consider multi-node application splitting or merger technique and the load balancing issue in the cluster of a heterogeneous environment. If a large number of kernels always favors one device then the overall throughput tends to decrease. This result can be observed in Table 1.

3 Methodology

This section describes the implementation of our model. Figure 3 explains the workflow that consists of three phases. The first phase is data collection, which uses the static analyzer to extract the CPU and GPU suitable application features. In the second phase, features are filter based on their composition, i.e., hardware, code and runtime (dynamic) as mentioned in the block *CPU and GPU Execution* and *Execution time* of Fig. 3. After that, the feature vector is labeled and stored in the database to be applied in the next phase as mentioned in the block *Dataset* of Fig. 3. Then, features selection methods, i.e., information gain and correlation analysis, are applied to reduce the feature vector as mentioned in the block *Feature selection* of Fig. 3. The final phase is machine learning classification, where feature vectors are trained on the machine learning

Table 1 Critical review of state-of-the-art machine learning heterogeneous scheduling approaches

References	Dataset	Feature extraction	Feature selection	Model selection	Model hyper-tuning	Traditional MLC	Output classes	Performance metrics	Processors
Wen et al. (2014)	NVIDIA, OpenCL, AMD and Parboil	✔	✔	✗	✗	SVM	Binary	Acc: 89%	Single Node—Two Processors
Taylor et al. (2017)	Rodinia and Parboil	✔	✗	✗	✗	SVM	Binary	N/A	Single Node—Two Processors
Ghose et al. (2016)	NVIDIA, AMD, Rodinia and Parboil	✗	✗	✗	✗	Decision tree	Binary	Acc: 89%	Single Node—Two Processors
Wen and O'Boyle (2017)	Parboil and Polybench	✔	✗	✗	✗	Decision tree	Binary	N/A	Single Node—Two Processors
Kofler et al. (2013)	Custom—University of Stuttgart	✔	✔	✔	✗	ANN–SVM	Binary	N/A	Single Node—Two Processors
Grewe and O'Boyle (2011)	SHOC, Parboil, NVIDIA and ATI Stream	✔	✗	✗	✗	SVM	CPU-Only|GPU-Only	Acc: 91%|95%	Single Node—Two Processors
Alizadeh and Momtazpour (2020)	NVIDIA and Custom	✔	✗	✗	✗	Random forest	Binary	Acc: 91%	Single Node—Two Processors
Proposed	NVIDIA, AMD and Parboil	✔	✔	✔	✔	**TPOT**	Multi-class (6)	Acc: 91	**Multi-Node—Multi-Processors**

Acc, accuracy; Binary, two classes; SVM, support vector machine

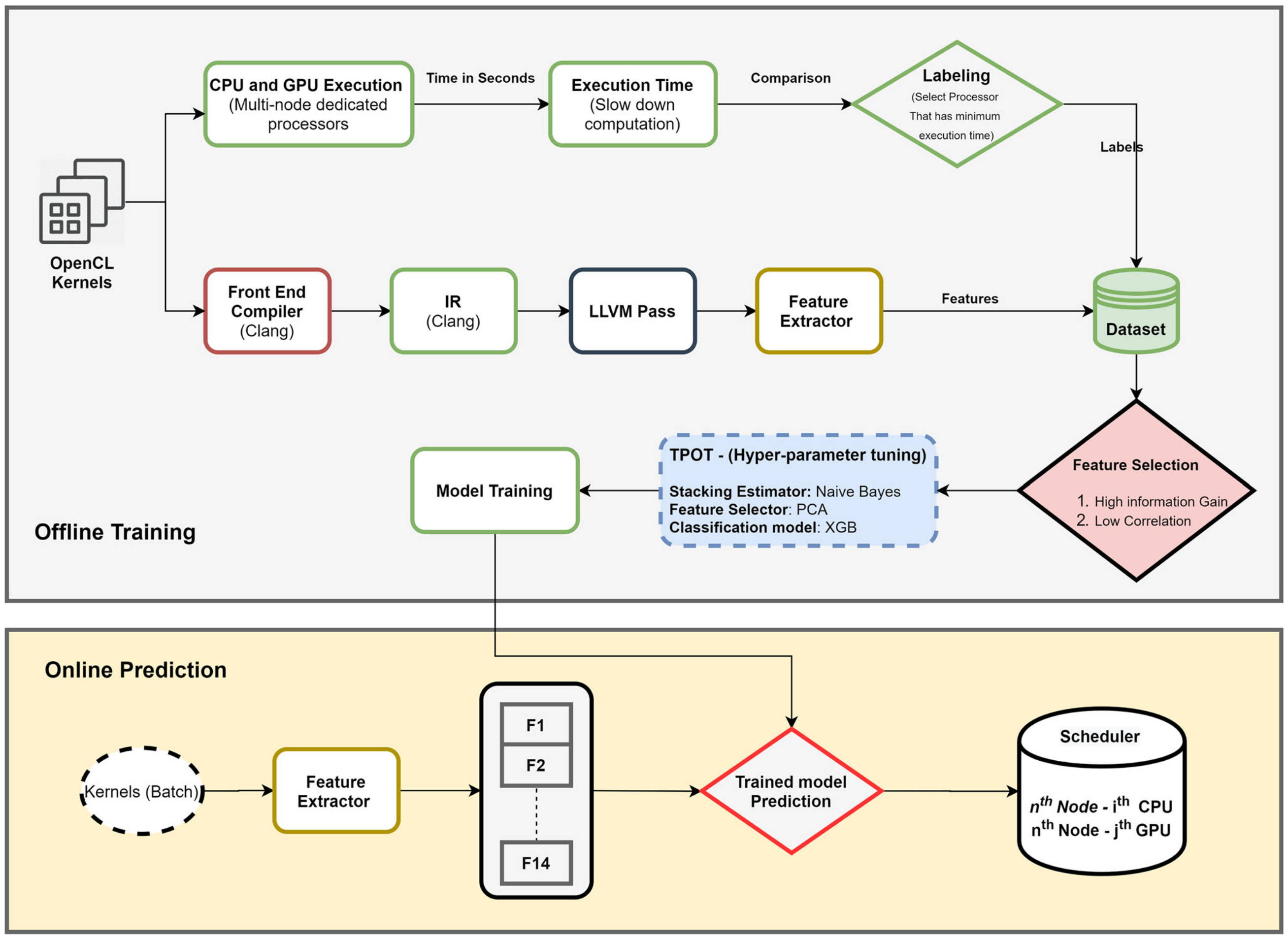


Fig. 3 The flowchart of the designed methodology

classifier to produce a detection model. Each experiment is explained in detail, and evaluation is performed comprehensively. The offline and online training code is publicly available on the link.

3.1 Dataset

Two benchmark suites are used, i.e., AMD and Polybench (Wen and O'Boyle 2017; Kofler et al. 2013; Grewe and O'Boyle 2011; Khalid et al. 2018). The benchmarks contain applications related to pattern recognition, image processing and mathematical computation (Wen and O'Boyle 2017; Kofler et al. 2013; Grewe and O'Boyle 2011; Khalid et al. 2018). The dataset is of size 155 data-parallel applications as shown in Tables 4 and 5. The application is run with different input sizes shown in the tables. We then execute the applications are then on two CPUs (Haswell 3.2 GHz and Skylake i7-6700 3.4 GHz) and two GPUs (Nvidia Geforce 760 and 740). The processor name is used label and the processor that have minimum execution time is used as output label as mentioned in the block *Labeling* of Fig. 3. The LLVM-based static code analyses extract the code features mentioned in Table 2. The dataset is divided by using hold-on policy, i.e., training set contains the 653 (70%) instances whereas testing set contains 277 (30%) instances.

3.2 Feature extraction

We can see the overview of the feature extractor in as mentioned in the block *Front End Compiler, IR, LLVM Pass and Feature extractor* of Fig. 3. The structure of the features can quickly identify the program behavior. Firstly `clang` (front end compiler) compiles the code as mentioned in the block *Front End Compiler* of Fig. 3. Then, the proposed static analyzer extracted the feature based on the intermediate representation of LLVM (Lattner 2008). We also use extracts some features that are not available in IR. The method only extracts the static code features and does not execute the program for profiling. We mention the list of features set in Table 2.

Table 2 Static analyzer extracted features

Index	Features set
1	Input data size
2	Return statement
3	Control statement
4	Allocation instruction
5	Load instructions
6	Store instructions
7	Multiplication (Float Datatype) operation
8	Addition (Integer Datatype) instruction
9	Multiplication (Integer Datatype) instruction
10	Division (Float Datatype) instruction
11	Division (Integer Datatype) instruction
12	Condition Check instruction
13	Addition (Float Datatype) instruction
14	Addition (Integer Datatype) instruction
15	Subtraction (Float Datatype)
16	Subtraction (Integer Data type)
17	Function call instruction
18	Functions
19	Blocks
20	Instructions
21	Float operation
22	Integer operation
23	Loop operation

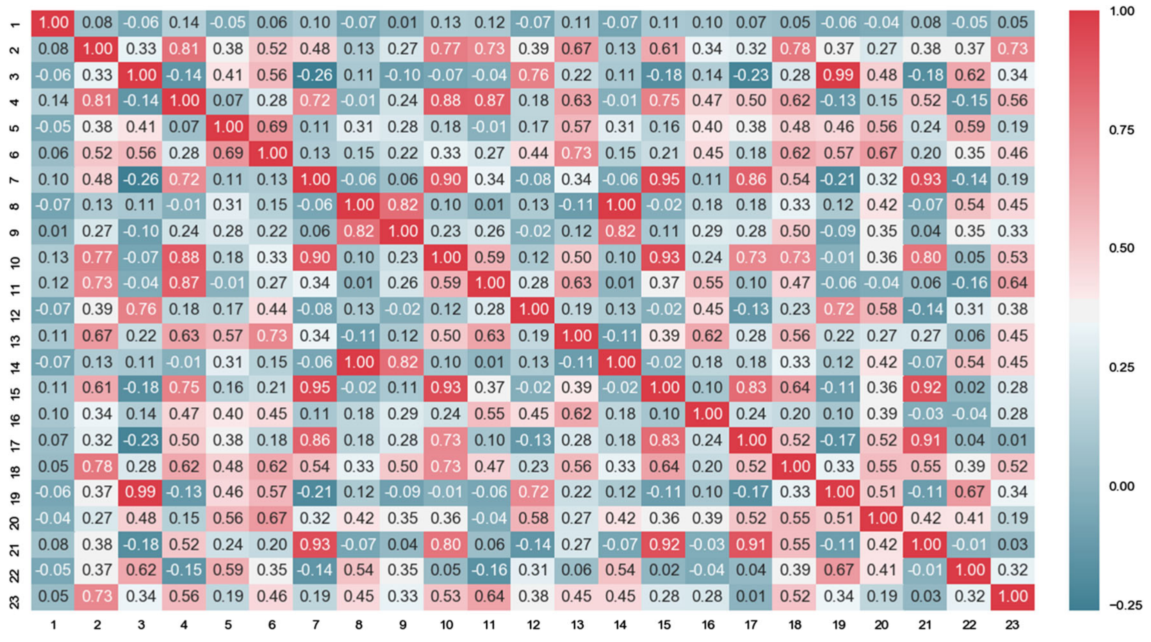
3.3 Feature selection

The feature set consists of 24 distinct features. Whether the non-domain expert collects the dataset or the domain experts provide it, the selection of key attributes is very important. Figure 4 shows the correlation matrix of the employed code features (mentioned in Table 2).

The feature importance ranking and correlation analysis is mentioned in Fig. 4. The feature which has high information and negative correlation is used in the training process as shown in block *feature selection* of Fig. 3. The highly correlated data will result in lower accuracy because of lower predictive power, and hence, it should be evaluated empirically. Figure 4 shows that the features 0, 6, 15, 12, 16, 8, 22 and 20 have a negative correlation (Ahmed et al. 2019a). The tree-based feature selection also validated the observation by ranking the same features on top and mentioned in Fig. 5. We mention classification model selected features in Table 3 and selection decision is mentioned in the block *Feature selection* of Fig. 3.

3.4 Machine learning classifier phase

The machine learning classifier phase is the final phase. This phase determines the appropriate model for application device suitability. The benchmark dataset consists of three CPUs and three GPUs. So, it contains six output classes, making it a multiclass problem. Based on the authors review as described in Table 4, this section describes the selected classifiers used in the current experiments.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1	1.00	0.08	-0.06	0.14	-0.05	0.06	0.10	-0.07	0.01	0.13	0.12	-0.07	0.11	-0.07	0.11	0.10	0.07	0.05	-0.06	-0.04	0.08	-0.05	0.05
2	0.08	1.00	0.33	0.81	0.38	0.52	0.48	0.13	0.27	0.77	0.73	0.39	0.67	0.13	0.61	0.34	0.32	0.78	0.37	0.27	0.38	0.37	0.73
3	-0.06	0.33	1.00	-0.14	0.41	0.56	-0.26	0.11	-0.10	-0.07	-0.04	0.76	0.22	0.11	-0.18	0.14	-0.23	0.28	0.99	0.48	-0.18	0.62	0.34
4	0.14	0.81	-0.14	1.00	0.07	0.28	0.72	-0.01	0.24	0.88	0.87	0.18	0.63	-0.01	0.75	0.47	0.50	0.62	-0.13	0.15	0.52	-0.15	0.56
5	-0.05	0.38	0.41	0.07	1.00	0.69	0.11	0.31	0.28	0.18	-0.01	0.17	0.57	0.31	0.16	0.40	0.38	0.48	0.46	0.56	0.24	0.59	0.19
6	0.06	0.52	0.56	0.28	0.69	1.00	0.13	0.15	0.22	0.33	0.27	0.44	0.73	0.15	0.21	0.45	0.18	0.62	0.57	0.67	0.20	0.35	0.46
7	0.10	0.48	-0.26	0.72	0.11	0.13	1.00	-0.06	0.06	0.90	0.34	-0.08	0.34	-0.06	0.95	0.11	0.86	0.54	-0.21	0.32	0.93	-0.14	0.19
8	-0.07	0.13	0.11	-0.01	0.31	0.15	-0.06	1.00	0.82	0.10	0.01	0.13	-0.11	1.00	-0.02	0.18	0.18	0.33	0.12	0.42	-0.07	0.54	0.45
9	0.01	0.27	-0.10	0.24	0.28	0.22	0.06	0.82	1.00	0.23	0.26	-0.02	0.12	0.82	0.11	0.29	0.28	0.50	-0.09	0.35	0.04	0.35	0.33
10	0.13	0.77	-0.07	0.88	0.18	0.33	0.90	0.10	0.23	1.00	0.59	0.12	0.50	0.10	0.93	0.24	0.73	0.73	-0.01	0.36	0.80	0.05	0.53
11	0.12	0.73	-0.04	0.87	-0.01	0.27	0.34	0.01	0.26	0.59	1.00	0.28	0.63	0.01	0.37	0.55	0.10	0.47	-0.06	-0.04	0.06	-0.16	0.64
12	-0.07	0.39	0.76	0.18	0.17	0.44	-0.08	0.13	-0.02	0.12	0.28	1.00	0.19	0.13	-0.02	0.45	-0.13	0.23	0.72	0.58	-0.14	0.31	0.38
13	0.11	0.67	0.22	0.63	0.57	0.73	0.34	-0.11	0.12	0.50	0.63	0.19	1.00	-0.11	0.39	0.62	0.28	0.56	0.22	0.27	0.27	0.06	0.45
14	-0.07	0.13	0.11	-0.01	0.31	0.15	-0.06	1.00	0.82	0.10	0.01	0.13	-0.11	1.00	-0.02	0.18	0.18	0.33	0.12	0.42	-0.07	0.54	0.45
15	0.11	0.61	-0.18	0.75	0.16	0.21	0.95	-0.02	0.11	0.93	0.37	-0.02	0.39	-0.02	1.00	0.10	0.83	0.64	-0.11	0.36	0.92	0.02	0.28
16	0.10	0.34	0.14	0.47	0.40	0.45	0.11	0.18	0.29	0.24	0.55	0.45	0.62	0.18	0.10	1.00	0.24	0.20	0.10	0.39	-0.03	-0.04	0.28
17	0.07	0.32	-0.23	0.50	0.38	0.18	0.86	0.18	0.28	0.73	0.10	-0.13	0.28	0.18	0.83	0.24	1.00	0.52	-0.17	0.52	0.91	0.04	0.01
18	0.05	0.78	0.28	0.62	0.48	0.62	0.54	0.33	0.50	0.73	0.47	0.23	0.56	0.33	0.64	0.20	0.52	1.00	0.33	0.55	0.55	0.39	0.52
19	-0.06	0.37	0.99	-0.13	0.46	0.57	-0.21	0.12	-0.09	-0.01	-0.06	0.72	0.22	0.12	-0.11	0.10	-0.17	0.33	1.00	0.51	-0.11	0.67	0.34
20	-0.04	0.27	0.48	0.15	0.56	0.67	0.32	0.42	0.35	0.36	-0.04	0.58	0.27	0.42	0.36	0.39	0.52	0.55	0.51	1.00	0.42	0.41	0.19
21	0.08	0.38	-0.18	0.52	0.24	0.20	0.93	-0.07	0.04	0.80	0.06	-0.14	0.27	-0.07	0.92	-0.03	0.91	0.55	-0.11	0.42	1.00	-0.01	0.03
22	-0.05	0.37	0.62	-0.15	0.59	0.35	-0.14	0.54	0.35	0.05	-0.16	0.31	0.06	0.54	0.02	-0.04	0.04	0.39	0.67	0.41	-0.01	1.00	0.32
23	0.05	0.73	0.34	0.56	0.19	0.46	0.19	0.45	0.33	0.53	0.64	0.38	0.45	0.45	0.28	0.28	0.01	0.52	0.34	0.19	0.03	0.32	1.00

Fig. 4 Correlation analysis

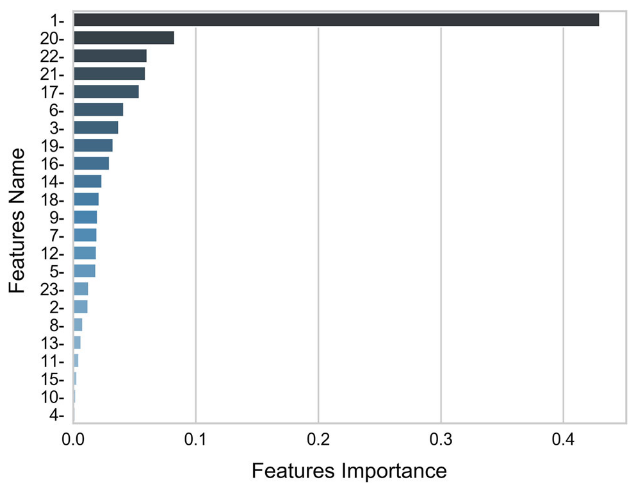


Fig. 5 Feature selection by using tree-based algorithm

Random forest Tin Kam proposed a bagging method named as random forest. Random forest is an ensembles classifier which contains the number of decision tree-based models (Tchernykh et al. 2016; Ahmed et al. 2019b; Reddy et al. 2020). The random number of feature is divided among each tree-based classifier. The voting mechanism is adopted while predicting the unknown class.

Decision tree Decision tree is a tree-based classifier which contains root, nodes and leaf nodes (Ahmed et al. 2018). The class label is assigned to each leaf node, and decision-making ability is applied to the internal nodes. On classification, an initial node with the high value of information starts making the decision. The decision tree is categorized as logic-based learning.

Naïve Bayes Naïve Bayes is a Bayes theorem-based probabilistic classifier. It is used in different types of real-world problems (Zafar et al. 2017). It takes the probability of each feature and calculates likelihood to classify an instance to a class.

KNN *K*-Nearest Neighbor (KNN) is known to be the most straightforward classification techniques. This method needs very less or no prior knowledge. Typically in supervised learning, the dataset is divided into training and testing sample. In the training set, the actual truth or output class is provided. The true class is used to be trained in the samples or features. KNN is an instant-based learning, also known as lazy learning classification (Ishtiaq et al. 2019).

TPOT The TPOT (Tchernykh et al. 2016; Ahmed et al. 2019a) uses genetic programming to construct features, selecting the machine learning model, and tune the selected model parameter. We provide the labeled data for each application to classification class to determine the device suitability. The hyper-parameter tuned model is shown in block *Tpot—hyper-parameter tuning* of Fig. 3. The data labeling is performed by running all application on CPU$_i$ and all GPU$_k$ a device. The device which has lower execution time is labeled as a selected device for that application.

3.5 Online prediction

In propose model, the collection of benchmark suits took less than a day. Both prediction models are trained offline. The overhead of using device suitability predictor includes the feature extraction and making the predictions. The overhead of feature extraction is negligible (approximated 1s in total) as a feature is extracted at compile time. The prediction model training is performed once and it is a one-off-cost. In total, the overhead of the prediction model is negligible, i.e., 3 s. The user submits OpenCL applications and the input dataset information to run the application.

Table 3 Selected feature set

No.	Top features name	Description
0	Input data size	Kernel input dataset dimension
6	Multiplication (Float Datatype) operation	Kernel code total number of multiplication operation on float data type
8	Multiplication (Integer Datatype) instruction	Kernel code total number of multiplication instruction on integer data type
9	Division (Float Datatype) instruction	Kernel code total number of division operation on float data type
11	Condition check instruction	Kernel code total number of branches
12	Addition (Float Datatype) instruction	Kernel code total number of addition instruction on float data type
14	Subtraction (Float Datatype)	Kernel code total number of subtraction operation on float data type
15	Subtraction (Integer Datatype)	Kernel code total number of multiplication operation on float data type
16	Function call instruction	Kernel code total number of function calls
18	Blocks	Kernel code total number of blocks (condition statements)
19	Instructions	Kernel code total number of machine instruction
20	Float operation	Kernel code total number of float operation
21	Integer operation	Kernel code total number of integer operations
22	Loop operation	Kernel code total number of loop operations

Table 4 Full features set comparison with and without class balancing (SMOTE)

Classifier	F-measure	Precision	Recall	R^2
Without smote				
Tpot	0.67	0.72	0.72	0.24
Random forest	0.67	0.72	0.72	0.26
Decision tree classifier	0.45	0.51	0.41	0.53
Naive	0.19	0.21	0.23	0.33
KNN	0.39	0.45	0.39	0.47
With smote				
Tpot	0.91	0.91	0.92	0.73
Random forest	0.84	0.84	0.84	0.28
Decision tree classifier	0.55	0.62	0.50	0.07
Naive	0.27	0.30	0.37	0.03
KNN	0.46	0.47	0.51	0.13

Table 5 Reduced feature set comparison with and without class balancing (SMOTE)

Classifier	F-measure	Precision	Recall	R^2
Without smote				
Tpot	0.69	0.72	0.68	0.26
Random forest	0.76	0.78	0.75	0.31
Decision tree classifier	0.45	0.51	0.41	0.53
Naive	0.19	0.21	0.23	0.33
KNN	0.39	0.45	0.39	0.47
With smote				
Tpot	0.91	0.91	0.92	0.76
Random forest	0.75	0.75	0.79	0.11
Decision tree classifier	0.50	0.56	0.50	0.08
Naive	0.23	0.26	0.36	0.01
KNN	0.37	0.39	0.37	0.55

After that feature extractors proposed features (details are described in Table 3) of each submitted job. The extracted features are provided along with input data size information to device suitability trained model. Then, prediction of nth node ith processor or nth node jth processor (best processing device for the job) is made as mentioned in block *online prediction* of Fig. 3.

4 Experiments and results

4.1 Evaluation measures

To evaluate the classification, standard metrics, i.e., precision, recall and F-measure are used (Ahmed et al. 2019b). We use R^2, another statistical method to evaluate the goodness of fit of a model. The R^2 interpret the amount of variation in the predicted and actual class. So it is the percentage of variation that input features can predict in output class, which effectively means that if output class changes by a percentage p, then input features can still predict that change in output class.

5 Results

The result mentioned in Tables 4 and 5 obtained from hold on policy 70-30 split ratio using five selected classifiers. Tables 4 and 5 show classifier performance on full feature set shown in Table 2 and reduced feature set shown in Table 3, respectively. Moreover, it is seen that the number of OpenCL kernel supporting the CPU device is significantly less when compared to the GPU device and particularity to have a number of samples of a given class under-represented compared to other classes gives rise to the "class imbalance" problem. In order to handle this problem, we used SMOTE a Python toolbox to tackle the curse of imbalanced data. The classifier's performance is measured with four evaluation metrics, i.e., precision, recall, F-measure and R^2. After getting feature selection and model selection, the models are trained and tested on the dataset. The prediction model performance is mentioned in Fig. 6. The ROC curve for class 1 is 0.92, class 2 is 0.90, class 3 is 0.99, class 4 is 1 and class 5 is 0.99. The high precision–recall curve value for all class signifies the excellent prediction. However, the precision–recall curve of class 6 is 0.88, which can be observed in Fig. 8. We show the ROC curve for the training data in Fig. 7. The mean ROC value for classification is 0.98. Class 2 achieved the precision–recall curve of 0.96 and 3, 4, 5 achieves 1.

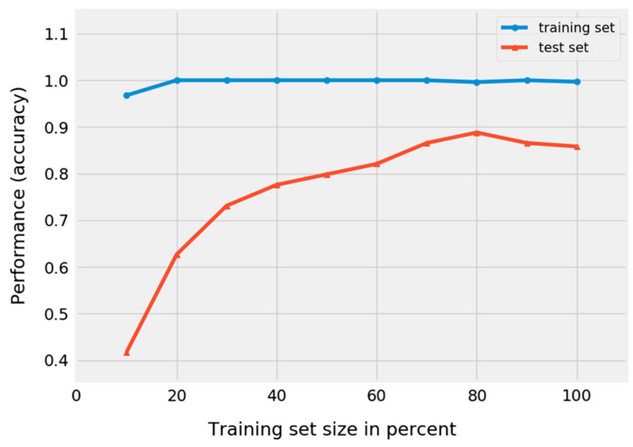


Fig. 6 Device classification model using TPOT

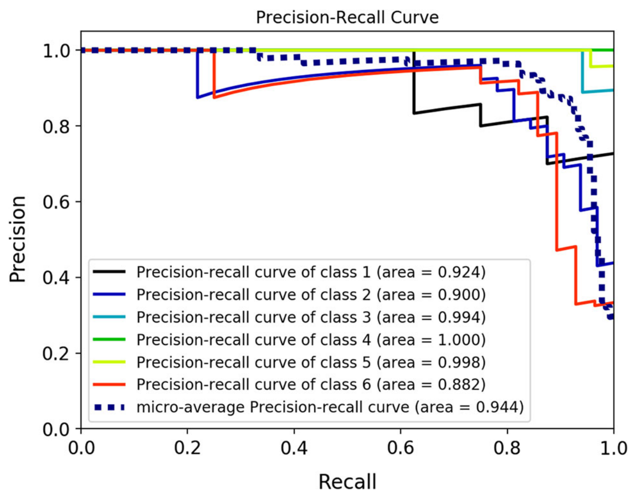


Fig. 8 Device classification model precision–recall curve

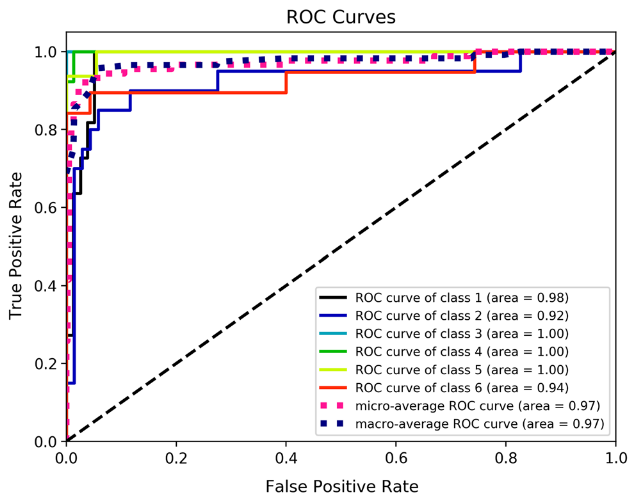


Fig. 7 Device classification model ROC

The *F*-measure score is 0.88, which presents that the model can produce perfect classification results. The correctly predicted classes are very high and the FPR is low, which can be observed in (Fig. 7).

In Table 4, an experiment is conducted on 930 samples by using full features set shown in Table 2. Table 4 demonstrates that when the full feature is used without class balancing, then random forest produces the higher R^2 as compared to the TPOT. However, the R^2 is the very low end which indicated that class imbalance problem has a large impact on the performance of the classifiers. In Table 4, when SMOTE class balancing is performed, TPOT performance increased to 35% and random forest also increased to 25%. More importantly, R^2 of TPOT is increased to 180%.

In Table 5, an experiment is conducted on 930 samples by using reduced features set shown in Table 3. Table 5 demonstrates that when the reduced feature is used without class balancing, then random forest produces the 19% higher R^2 as compared to the TPOT. However, the R^2 is the very low end which indicated that class imbalance problem has a large impact on the performance of the classifiers using reduced feature set. In Table refffws, when SMOTE class balancing is performed, TPOT performance increased to 31% and random forest *F*-measure reduced. More importantly, R^2 of TPOT is increased to 192%. After feature reduction, TPOT produces a higher recall as well. The higher recall indicated that tune parameters returned the most relevant results. The higher precision shown in Table 5 indicates that the TPOT predicts the relevant class results in more correctly than the irrelevant.

5.1 Application of device suitability model

In a heterogeneous cluster environment, programmers map applications to specific devices. This decision is not optimal in a multi-node or cluster of a heterogeneous system. The number of jobs is submitted to the scheduler. The scheduler maps the application to the computing devices. The decision about the work distribution should be balanced to achieve maximal throughput. It is very difficult for a programmer to decide the mapping of jobs to a variety of heterogeneous computing devices. Multi-node, device suitability schedule batch of jobs in a load-balanced manner while effectively utilizing heterogeneity that is inherent to heterogeneous computing devices.

6 Conclusion and future work

In a heterogeneous cluster environment, programmers map application to specific devices. This decision is not optimal in a multi-node or cluster of a heterogeneous system. The number of jobs is submitted to the scheduler. The scheduler maps the application to the computing devices. The decision about the work distribution should be balanced to achieve maximal throughput. In this study, present the framework to classify OpenCL applications based on their device suitability. The tree-based pipeline optimization strategy is used to select the optimal model for its hyper-parameter. The feature selection is performed by using correlation analysis and feature importance algorithm. The trained model predicts the processors that can optimally handle the OpenCL program in a cluster. The prediction is based on newly developed LLVM-based static analyses. The features represent runtime application behavior. To incorporate dynamic behavior, we also added run time features. The model is to build and trained offline. The model trained and tested on OpenCL benchmarks. Experimental results show that the trained model outperforms different classifiers and achieves an *F*-measure of 0.91. In future extensions of this work, the proposed study can be extended to energy-efficient heterogeneous device classification. In that case, the data labeling will be decided based on the minimum energy consumption. Moreover, the proposed model can be utilized in sensor networks. Evolutionary computation can also be considered to improve our scheduling model as the extension for our further study.

Acknowledgements Open Access funding provided by Western Norway University Of Applied Sciences.

Compliance with ethical standards

Conflict of interest The authors declare that there are no conflicts of interest in this paper.

Ethical standard This article does not contain any studies with human participants performed by any of the authors.

References

Ahmed U, Zafar L, Qayyum F, Islam MA (2018) Irony detector at semeval—2018 task 3: irony detection in English tweets using word graph. In: The international workshop on semantic evaluation, pp 581–586

Ahmed U, Aleem M, Noman Khalid Y, Arshad Islam M, Azhar Iqbal M (2019a) RALB-HC: a resource-aware load balancer for heterogeneous cluster. In: Concurrency and computation: practice and experience, p e5606

Ahmed U, Liaquat H, Ahmed L, Hussain SJ (2019b) Suggestion miner at semeval-2019 task 9: suggestion detection in online forum using word graph. In: The international workshop on semantic evaluation, pp 1242–1246

Ahmed U, Waqas H, Afzal MT (2020) Pre-production box-office success quotient forecasting. Soft Comput 34:6635–6653

Albayrak OE, Akturk I, Ozturk O (2012) Effective kernel mapping for OpenCL applications in heterogeneous platforms. In: The international conference on parallel processing workshops, pp 81–88

Alizadeh NS, Momtazpour M (2020) Machine learning-based interference detection in GPGPU concurrent kernel execution. In: The international computer conference. Computer Society of Iran, pp 1–4

Aloqaily M, Ridhawi IA, Salameh HB, Jararweh Y (2019) Data and service management in densely crowded environments: challenges, opportunities, and recent dvelopments. IEEE Commun Mag 57:81–87

Amrollahi M, Hadayeghparast S, Karimipour H, Derakhshan F, Srivastava G (2020) Enhancing network security via machine learning: opportunities and challenges. In: Choo K-KR, Dehghantanha A (eds) Handbook of big data privacy. Springer, Berlin, pp 165–189

Augonnet C, Thibault S, Namyst R, Wacrenier PA (2011) Starpu: a unified platform for task scheduling on heterogeneous multicore architectures. Concurr Comput Pract Exp 23(2):187–198

Becchi M, Byna S, Cadambi S, Chakradhar S (2010) Data-aware scheduling of legacy kernels on heterogeneous platforms with distributed memory. In: ACM symposium on parallelism in algorithms and architectures, pp 82–91

Belviranli ME, Bhuyan LN, Gupta R (2013) A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures. ACM Trans Archit Code Optim 9. Article no. 57

Choi HJ, Son DO, Kang SG, Kim JM, Lee H-H, Kim CH (2013) An efficient scheduling scheme using estimated execution time for heterogeneous computing systems. J Supercomput 65:886–902

Daraghmeh M, Al Ridhawi I, Aloqaily M, Jararweh Y, Agarwal A (2019) A power management approach to reduce energy consumption for edge computing servers. In: Fourth international conference on fog and mobile edge computing, pp 259–264

Ghose A, Dokara L, Dey S, Mitra P (2017) A framework for opencl task scheduling on heterogeneous multicores. Parallel Process Lett 27:1750008

Ghose A, Dey S, Mitra P, Chaudhuri M (2016) Divergence aware automated partitioning of OpenCL workloads. In: The India software engineering conference, pp 131–135

Gregg C, Brantley J, Hazelwood K (2010) Contention-aware scheduling of parallel code for heterogeneous systems. In: USENIX workshop on hot topics in parallelism, pp 1–10

Grewe D, O'Boyle MF (2011) A static task partitioning approach for heterogeneous systems using OpenCL. In: The international conference on compiler construction, pp 286–305

Grewe D, Wang Z, O'Boyle MFP (2013) Portable mapping of data parallel programs to OpenCL for heterogeneous systems. In: IEEE/ACM international symposium on code generation and optimization, pp 1–10

Hechtman BA, Sorin DJ (2013) Exploring memory consistency for massively threaded throughput-oriented processors. In: Annual international symposium on computer architecture, pp 201–212

Huchant P, Counilh MC, Barthou D (2016) Automatic OpenCL task adaptation for heterogeneous architectures. In: European conference on parallel processing, pp 684–696

Iftikha Z, Jangsher S, Qureshi HK, Aloqaily M (2019) Resource efficient allocation and RRH placement for backhaul of moving small cells. IEEE Access 7:47379–47389

Ishtiaq A, Islam MA, Iqbal MA, Aleem M, Ahmed U (2019) Graph centrality based spam SMS detection. In: The international Bhurban conference on applied sciences and technology, pp 629–633

Khalid YN, Aleem M, Prodan R, Iqbal MA, Islam MA (2018) E-OSched: a load balancing scheduler for heterogeneous multicores. J Supercomput 74:5399–5431

Khalid YN, Aleem M, Ahmed U, Islam MA, Iqbal MA (2019) Troodon: a machine-learning based load-balancing application scheduler for CPU–GPU system. J Parallel Distrib Comput 132:79–94

Khan NA, Latif MB, Pervaiz N, Baig M, Khatoon H, Baig MZ, Burney A (2019) Smart scheduler for CUDA programming in heterogeneous CPU/GPU environment. In: The international conference on computer modeling and simulation, pp 250–253

Kofler K, Grasso I, Cosenza B, Fahringer T (2013) An automatic input sensitive approach for heterogeneous task partitioning. In: The international conference on supercomputing, pp 149–160

Krishna M, Wang Z, O'Boyle MFP (2013) Smart, adaptive mapping of parallelism in the presence of external workload. In: IEEE/ACM international symposium on code generation and optimization, pp 1–10

Lattner C (2008) LLVM and clang: next generation compiler technology. BSD Conf 5:1–10

Lee VW, Kim C, Chhugani J, Deisher M, Kim D, Nguyen AD, Satish N, Smelyanskiy M, Chennupaty S, Hammarlund P, Singhal R, Dubey P (2010) Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. In: Annual international symposium on computer architecture, pp 451–460

Luk CK, Hong S, Kim H (2009) Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping. In: Annual IEEE/ACM international symposium on microarchitecture, pp 45–55

Pérez B, Bosque JL, Beivide R (2016) Simplifying programming and load balancing of data parallel applications on heterogeneous systems. In: Annual workshop on general purpose processing using graphics processing unit, pp 42–51

Ravi VT, Agrawal G (2010) A dynamic scheduling framework for emerging heterogeneous systems. In: IEEE international conference on high performance computing, pp 1–10

Ravi VT, Becchi M, Jiang W, Agrawal G, Chakradhar S (2012) Scheduling concurrent applications on a cluster of CPU–GPU nodes. In: IEEE/ACM international symposium on cluster, cloud and grid computing, pp 140–147

Reddy GT, Reddy MPK, Lakshmanna K, Kaluri R, Rajput DS, Srivastava G, Baker T (2020) Analysis of dimensionality reduction techniques on big data. IEEE Access 8:776–788

Sakhnini J, Karimipour H, Dehghantanha A, Parizi RM, Srivastava G (2019) Security aspects of internet of things aided smart grids: a bibliometric survey. Internet Things. https://doi.org/10.1016/j.iot.2019.100111

Stone JE, Gohara D, Shi G (2019) Opencl: a parallel programming standard for heterogeneous computing systems. Comput. Sci. Eng. 12:66–73

Taylor B, Marco VS, Wang Z (2017) Adaptive optimization for OpenCL programs on embedded heterogeneous systems. In: ACM SIGPLAN/SIGBED conference on languages, compilers, and tools for embedded systems, pp 11–20

Tchernykh A, Lozano L, Schwiegelshohn U, Bouvry P, Pecero JE, Nesmachnow S, Drozdov AY (2016) Online bi-objective scheduling for IaaS clouds ensuring quality of service. J Grid Comput 14:5–22

Tsog N, Becker M, Bruhn F, Behnam M, Sjödin M (2019) Static allocation of parallel tasks to improve schedulability in CPU–GPU heterogeneous real time systems. Annu Conf IEEE Ind Electron Soc 1:4516–4522

Wen Y, O'Boyle MF (2017) Merge or separate? Multi-job scheduling for OpenCL kernels on CPU/GPU platforms. In: The general purpose GPUs, pp 22–31

Wen Y, Wang Z, O'Boyle MFP (2014) Smart multi-task scheduling for OpenCL programs on CPU/GPU heterogeneous platforms. In: The international conference on high performance computing, pp 1960–1965

Zafar L, Afzal MT, Ahmed U (2017) Exploiting polarity features for developing sentiment analysis tool. In: The international workshop on emotions, modality, sentiment analysis and the semantic web, pp 1–10