Sustainable Computing: Informatics and Systems 35 (2022) 100683

Contents lists available at ScienceDirect

Sustainable Computing: Informatics and Systems

journal homepage: www.elsevier.com/locate/suscom

A ML-based resource utilization OpenCL GPU-kernel fusion model

Usman Ahmed [a], Jerry Chun-Wei Lin [a,*], Gautam Srivastava [b,c]

[a] Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway
[b] Department of Math and Computer Science, Brandon University, Brandon, Canada
[c] Research Centre for Interneural Computing, China Medical University, Taichung, Taiwan

ARTICLE INFO

Keywords:
Scheduling
Kernel fusion
High-performance computing
ML

ABSTRACT

Massive data parallelism can be achieved by using general-purpose graphics processing units (GPGPU) with the help of the OpenCL framework. When smaller data with higher GPU memory is executed, it results in a low resource utilization ratio and energy inefficiencies. Up until now, there is no existing model to share GPU for further execution. In addition, if the kernel pair requires the same computation resource, then kernel merging also results in a significant increase in execution time. Therefore, optimal device selection, as well as kernel merging, can significantly speed up the execution performance for a batch of jobs. This paper proposes a kernel merging method that leads to high GPU occupancy. As a result, it reduces execution time and increases GPU utilization. Additionally, a machine learning (ML)-based GPU sharing mechanism is presented to select pairs of kernels in OpenCL frameworks. The model first selects suitable architecture for the jobs and then merges GPU kernels for better resource utilization. From all the GPU candidates, the optimal pair of the kernel concerning data size is selected. The experimental results show that the developed model can achieve 0.91 F1-measure for device selection and 0.98 for the scheduling scheme of kernel merging.

1. Introduction

Recently, we have witnessed the emergence/evolution of Graphics Processing Units (GPU) as a viable alternative to the Central Processing Unit (CPU), which is used to speed up the execution of data-parallel applications. Program-oriented applications for General-Purpose Computing on GPU (GPGPU), Open Computing Language (OpenCL), and Compute Unified Device Architecture (CUDA) have emerged as the industry standard and requirement in many domains and applications. Among them, OpenCL has garnered widespread adoption by the industry due to its great support for running varied applications in heterogeneous architectures. OpenCL-based applications can be executed on a CPU, a GPU, or any other supported accelerators, thus making OpenCL-based applications are truly heterogeneous.

To utilize parallel architecture avoiding heavy execution of computer programs, programmers tend to map OpenCL applications to GPU. The suitability of the job architecture (i.e., CPU or GPU) depends on the type of computation and input data size. With the increase of parallel computing with the utilization of OpenCL-based applications, a scheduler needs to map jobs efficiently by concerning the type of applications. To attain high throughput and reduce execution time, the kernel-based scheduler helps to map the applications according to the computational requirements and device architecture.

In merging two kernels, two or more code segments of OpenCL are merged into a single kernel. The merged kernel is then executed serially according to the resource requirements. The lack of operating system support for the GPU kernel merging method has raised interest for researchers to implement the kernel merging method for multitasking on GPU. The merging results can increase the resource utilization and help to reduce the execution time of the GPU mapped jobs [1,2]. However, the issue with merging is that the identification of suitable candidates on particular data sizes is a non-trivial task. This task requires careful instrumentation to create an optimal combination. The irregular combination of candidate kernels results in the degradation of the execution time of GPU tasks. In this paper, we only consider merging two kernels since according to recent studies, the fusion of more than two kernels results in degrading computational performance.

1.1. Motivation

In real cases, some kernels can improve resource utilization; however, many kernels have poor performance due to data transfer and contention of the shared resources, i.e., GPU memory, bandwidth, and streaming cores, among others. We highlight this fact in Fig. 1, which represents the relative speedup of the merged kernel under sequential

* Corresponding author.
E-mail addresses: Usman.Ahmed@hvl.no (U. Ahmed), jerrylin@ieee.org (J.C.-W. Lin), SRIVASTAVAG@brandonu.ca (G. Srivastava).

https://doi.org/10.1016/j.suscom.2022.100683
Received 15 June 2021; Received in revised form 8 January 2022; Accepted 30 January 2022
Available online 12 February 2022

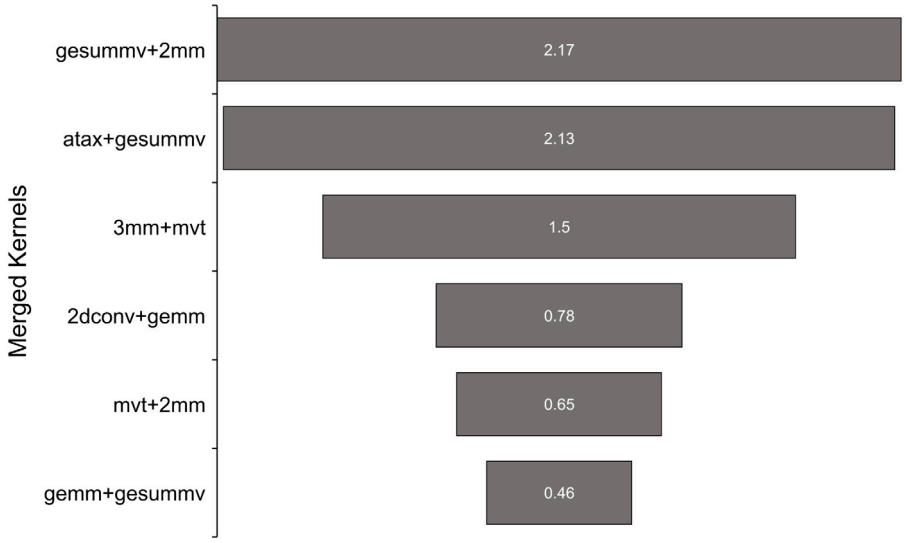

Fig. 1. Merged kernel speedup achieved over sequential kernel executions.

executions. These experimental jobs are taken from the Polybench suite [1,2]. In Fig. 1, the x-axis represents the number of the merged kernels, and the y-axis represents the achieved speedup after merging progress. Fig. 1 shows clearly that kernel merging can lead to high speedup. However, some kernels suffer from significant slowdown. This motivates us to propose a machine learning (ML)-based model, which is capable of forecasting potential speedups or slowdowns of the merged kernels.

In addition, the number of the merged kernels is also affected by input data size. For a few ranges of data size, the merged kernels could suffer from a slowdown. This is due to the ratio of GPU memory to bandwidth. Fig. 2 represents two merged kernels, i.e., `2dconv` and `tax`, under different data sizes over their sequential execution, which can be represented as the baseline model for comparison. The y-axis represents the gain speedup, where $2.17x$ is attained for a data size of $135,360$, and a slowdown is observed when $1,053,696$ data size is considered as the input data. This highlights the fact that a scheduler should consider the data size as an essential feature for merging two kernels. The kernel *gesummv, 2 mm, 3 mm and gemm,* perform well, while others, e.g., combination, slow down because of the data transfer overhead and communication cost. The serial execution of the merged kernel can improve performance. However, the pair of the kernel, type of dataset, data size, and amount of computation resource required to run the merged kernel is critical. The GPU resource requirement for the specific kernel is called resource allocation. Determining two kernel percentages of the allocation directly depends on the operations performed, called a mixing ratio. The allocation is essential decision making and can result in a slow down in performance; the device selection and kernel merging method can improve performance. However, selecting the kernel, data size, type of operations, and correct mixing ratio are critical, otherwise, it will result in low resource allocations. We proposed just in time compiler to create and schedule separate and merge kernels to heterogeneous platforms without the profiling of the applications.

1.2. Contribution

In previous works, RALB-HC (an OpenCL scheduler) maps jobs on clusters of CPU and GPU devices in a batch-processing manner [1]. The model considers the device suitability and speedup forecasting to select computing devices after job execution time and speedup device selection. The model performs a load balancing mechanism based on the computing capabilities. RALBHC increases the number of clusters throughput when compared to the state of the heuristics. As the number of applications utilizing GPUs-based accelerators increases, there should also be an increase in the utilization of complete resources by executing multiple compute kernels to improve resource utilization, as well as device occupancy [2]. However, RALBHC cannot perform scheduling by considering GPU as a shared resource. This paper proposes a machine learning (ML)-based scheduling method that combines two OpenCL kernels to increase GPU resource utilization for the batch of jobs. Similar to RALBHC, the proposed method first selects jobs based on the computation requirements and speedup attained on a specific device. In addition, a classification model is developed that is trained on the proposed features to predict whether the combination of two OpenCL kernels or their separate execution will produce better speedup performance. The following are the major contributions of this paper:

1. Extraction and selection of significant features are used to forecast the relative speedup of merged kernel execution.
2. A machine learning (ML)-based kernel merging classification model is designed to predict the kernel suitability of OpenCL kernels.
3. Experimental evaluation is then performed to show the performance of the proposed model in terms of execution time and prediction by using 15 mainstream benchmark applications.

2. Literature review

Scheduling problems [3–5] have been investigated for many years based on different domains and applications. Scheduling of computing kernels has been discussed in recent years, and some of them are still developed in progress [6–11]. As we can see, different models have been described based on different criteria, i.e., scheduling type, scheduling method, load-balancing, code instrumentation, applications or resource-aware, architecture support, and feature reduction, among others. Some works required code instrumentation and application profiling [12–14]. In general, the load imbalance frequently exists in the multiple heterogeneous machines [15]. In addition, the scheduler assigns the kernels to the computing devices (i.e., CPU or GPU) based on the workload [16–18]. As we know in general, the low overhead of the scheduling method helps to minimize execution time. However, up to now, the GPU-based kernel merging is not performed for the heterogeneous system and has not been discussed and studied yet.

Supervised learning models [8,9] have also been widely applied in many domains and applications. A kernel computation operation is used for predictive modeling. Data that comes in as a paired feature

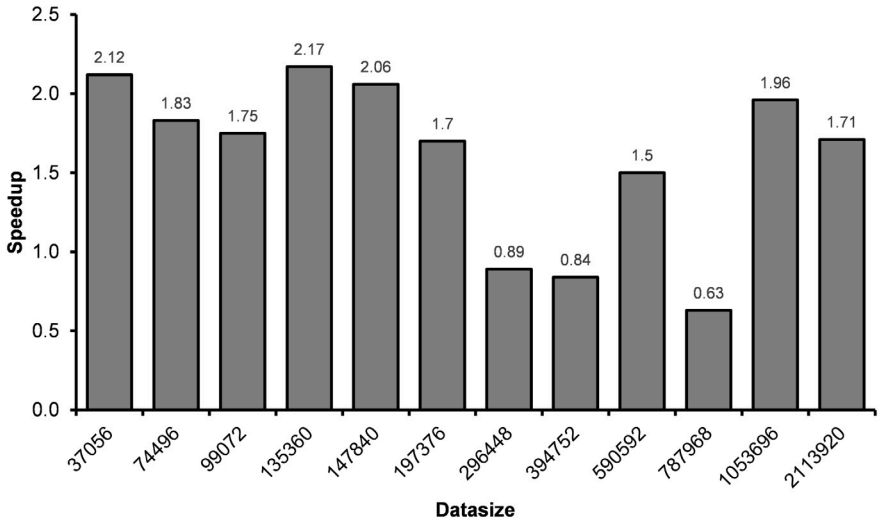

Fig. 2. `2dconv` and `atax` kernels are merged to speedup analysis on different data sizes.

vector is then labeled as output. There are two types of kernel code features, i.e., static and dynamic. The static code features are extracted from the data without any modification, i.e., loop operation, maths operation, and data type, while the dynamic code features include input data size. In most cases, the prediction model assigns the OpenCL kernel to a specific processor. This response is categorized as a classification problem, and most of the learning models find the functions mapping to minimize unseen data by measuring the loss function.

The application-based kernel scheduling is proposed for scheduling tasks [11]. It is performed by assigning the kernel to a particular processor. The drawback for Qilin index-based scheduling is that it has a code instrumentation overhead. In addition, a heterogeneous device scheduler for OpenCL is also proposed [10]. This technique solved the issue related to heterogeneity communication load and load balancing caused by the iterative computation. Huchant et al. [10] then presented a method that maps to a single OpenCL kernel. A heterogeneous device scheduler is proposed [6] that is considered as a multi-application scheduling method based on the application requirement and varied characteristics. It uses the iterative scheduling model, which is based on profiling of the data dependency and execution time. A greedy approach is designed to map the kernel on devices. However, this model does not require profiling as a learning method to predict the optimal set of devices based on the configurations. Augonnet et al. [7] proposed an innovative solution for scheduling devices based on CPU utilization and characteristics of the running applications. This developed StarPU model provides the execution environment for the runtime kernel. Becchi et al. [16] showed that optimization could be achieved with minimal usage of the hardware characteristics. However, the developed model requires an offline profiling overhead, and the proposed scheduler can use allocation data-parallel application without profiling.

Belviranli et al. [17] addressed the issue regarding the under-utilization of heterogeneous resources. The proposed HDSS partitioned the tasks among CPU and GPU to achieve low execution time and high throughput. However, the proposed model does not require job splitting and kernel code transformation. Binotto et al. [18] considered that the workload distribution is vital in a heterogeneous environment and addressed the cost issues by using a task scheduling system. However, this method requires online profiling and code instrumentation.

Gregg et al. [19] then developed a model that can profile the processing performance and address performance-related problems in generating partitions of code. A dynamic approach is considered to partition workload without profiling and training. The kernel splitting is done based on processor capabilities. Kaleem et al. [20] proposed the use of kernel splitting approach based on the profiling. The proposed model ensures lower execution time by combining two kernels. Choi et al. [21] addressed processor selection critical issues, predicted the execution time of the kernel, and used it to schedule the application on a CPU or a GPU device. Grewe et al. [9] designed a model that can improve performance by using the specific application on a suitable processing unit. A heterogeneous system is then also developed by partitioning the OpenCL programs. At compile time, application operations are then extracted as features. After that, a pre-trained support vector machine is used to train and predict the suitability of the application devices. Several in-depth analyses are then conducted to show the control flow divergence and its impact on the scheduling. In addition, the branch divergence analysis [8,9] is then used to train the classification model.

A computational framework is developed by using the source to source compiler to translate a kernel code into multi-device kernel code [22]. The static features and data transfer size is used as a feature to train the neural network, and the 0.87 prediction accuracy is achieved by the developed model. Wen et al. [23] addressed the issues in the resource utilization of the heterogeneous system. Based on the developed model, it can observe that the application-aware processor selection helps to increase system throughput and decrease turnout time. The code features, i.e., *number of instructions, load/store operations* and *input size, output size, and global work size* can also help to predict the speedup over the CPU.

To the best of our knowledge, GPU kernel merging speedup as well as device suitability for a given kernel has not yet been explored without code-splitting overhead, profiling, or indexing application executing time. Our research here proposes an approach to map the applications based on the application computation required and GPU capability. The proposed resource utilizer uses the prioritized list of resources and relative speed up to reduce the batch of the jobs in terms of execution time by the developed fusion model for kernel merging.

3. Methodology

In this section, we discuss the model development and Fig. 3 shows the workflow of the designed model. The dataset is collected by using a static feature extractor for CPU and GPU suitable application features. The computational oriented features are extracted i.e., hardware, code, and run-time as stated in the *GPU & GPU Execution* block and *Execution time* shown in Fig. 4. For the device suitability classifier, we use 137 kernels on separate CPU and GPU with different data sizes. For the merging classifier, all the applications (both merged and separated) are executed with several input sizes to create a job pool of 236 programs.

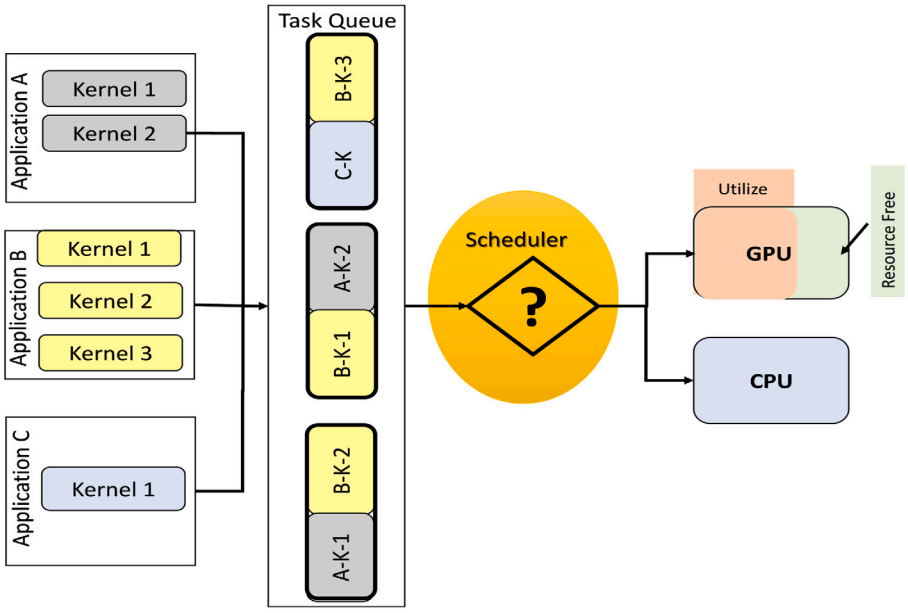

Fig. 3. Utilization in the heterogeneous computing environments.

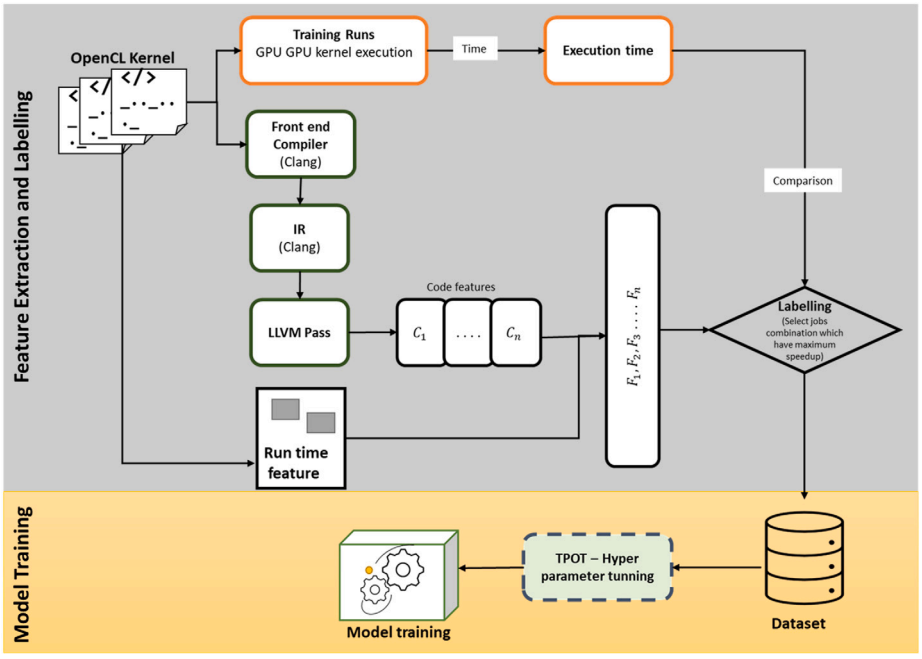

Fig. 4. Feature extraction and selection by using LLVM and TPOT.

After the features extraction stage, features are labeled by the most suitable device having minimum execution time in the block. The tree-based pipeline optimization method is adopted to feature reduction and classification model selection, as mentioned in the block *TPOT* of Fig. 4. The training process of the machine learning model is then performed in the final phase with developed parameters from TPOT and extracted features. Then, the trained model is then used in the online mode to produce device suitability and merging model. Each experiment is explained, and the evaluation results are reported as follows:

3.1. Dataset and feature extraction

For experimental setup, we used two datasets, namely AMD and Polybench, respectively [9,13,15,22]. The applications in both benchmarks contain mathematical computation, image processing, and pattern recognition tasks. These application-driven task helps to generalize better for the application used in the development sets. For different architecture, data labeling requires to be performed for certain applications. In total, 155 data-parallel kernel codes are used as shown in Table 2. We run the application under different input sizes. We used two CPUs (Haswell 3.2 GHz and Skylake i7-6700 3.4 GHz) and two

Table 1
Full features set.

Index	Feature
1	Data size
2	Total number of return statement
3	Total number of control statement
4	Total number of an allocation instruction
5	Total number of load instructions
6	Total number of store instructions
7	Total number of multiplication (float data type) operation
8	Total number of addition (integer data type) instruction
9	Total number of multiplication (integer data type) instruction
10	Total number of division (float data type) instruction
11	Total number of division (integer data type) instruction
12	Total number of condition check instruction
13	Total number of addition (float data type) instruction
14	Total number of addition (integer data type) instruction
15	Total number of subtraction (float data type)
16	Total number of subtraction (integer data type)
17	Total number of function call instruction
18	Total number of functions
19	Total number of blocks
20	Total number of instructions
21	Total number of float operation
22	Total number of integer operation
23	Total number of loop operation

GPUs (NVIDIA Geforce 760 and 740[1]) in the experiments. In addition, we used the minimum execution time and energy as output label as mentioned in the block *Labeling* of Fig. 4. We developed our LLVM pass for feature extraction in Table 1.

The purpose of the static code analyzer is to gather attributes of kernel code. These sets of attribute values represent the program behavior. The static analyzer consists of two parts; a clang LLVM parser [24] and a Python script. First, to ensure that the kernel code is error-free, the OpenCL kernel is Just-In-Time compiled using clang (front-end compiler). Then, the clang LLVM parser obtains features based on LLVM intermediate representation (IR). The python script uses regular expressions to detect features that are not available or detected by the LLVM (IR).

3.2. Feature selection

Since the feature set is extracted by non-domain experts and/or domain experts, the key attribute selection is vital. Fig. 5 shows the correlation matrix of the employed code features (mentioned in Table 1). Table 3 shows the relative feature importance. We reduce the feature set from 24 distinct features to 10 distinct features i.e., $1, 3, 5, 6, 9, 12, 16, 20, 21, \ldots, 22$. The data size is the most important factor as it directly affects resource allocation. If the data size of the selected kernel is full, filling the resource capacity, then the model will not select it as a candidate for the kernel merging task. A single kernel will perform independently without participating in the merging task. This is the reason that the information gain of the data size is high, among other features. The merged kernel has two independent datasets, as two different scientific applications have different formulations of the task. Therefore, merged kernels have separate datasets and are performed separately. The selection criteria for the features to be selected is high related importance and negative correlation among other features, as shown in block *feature selection* of Fig. 4. The highly correlated features result in model overfitting and lower predictive power.

[1] https://www.nvidia.com/en-us/

3.3. Data labeling

By running concurrent training runs of programs, both code features and run-time features are extracted. The feature vector, along with the execution time information, is used to develop the dataset. Similarly, the feature vector and the suitable class/label are used to develop the suitability classifier. The data labeling is performed by running all applications on CPU and all GPU a device. The device which has a lower execution time is labeled as a selected device for that application. The vital code factors are mentioned in Table 3 where data size *features 1* plays a vital role in the selection of CPU and GPU.

3.4. Machine learning classifier phase

The machine learning classifier process is the final step. We make use of two models as mentioned in Table 4. Device suitability classifies the suitable device for the submitted job. Then, GPU device jobs are further classified into the merger kernel or to be performed separately, as mentioned in Fig. 6. We used the genetic programming-based autoMl system that optimizes the features selections, model selection, and model hyperparameter to maximize the accuracy of the supervised learning task [25], which is called a Tree-based pipeline optimization method. The selected model is then used to train and compare with the traditional statistical learning methods. We used five classification models, i.e., *Random Forest, Decision Tree, Naïve Bayes, Tree-based pipeline optimization method, and KNN* in the experiments. The novelty and contribution of the proposed model is the extraction of computationally relevant features that helps in the distribution of resource based on the learning method.

3.5. Model training and testing

In this step, the model is trained and tuned according to hyper-parameters. The final model is trained on the tuning parameters and selected features as mentioned in Table 5. We used 137 applications with different data sizes and distributed 80% for training and 20% applications for testing purposes. The benchmark model on the specific architecture took less than a day; both models are trained offline. The overhead of feature extraction includes feature extraction from machine-level code. The overhead of the feature extraction is negligible as we used the llvm passer that extracted the mentioned features during just-in-time (JIT) compiling; the training and prediction are one-off-cost. For the experimentation, the hyper tuning progress of the training model took $2-3$ hours. However, training took 3 seconds, and prediction is under one millisecond.

4. Experimental results

The scheduling method is evaluated on the heterogeneous system with a CPU (Intel Core i5-4460) and a GPU (NVIDIA GeForce GTX 760). We used the Linux environment (UBUNTU 18.04) for the experimentation and compiler (GCC 5.4.0). The empirical evaluation was conducted to evaluate the performance of the proposed model. We used a K-fold cross-validation method for our experiments. The dataset is cross-validated k times; the subset of every single time is selected for testing, and $K-1$ subsets are used for training. In this way, every data point is validated and hence removes any bias from the results.

The results mentioned in Table 6 are obtained from hold on policy 70/30 split ratio using five selected classifiers. The results in Table 6 show classifier performance on a reduced feature set. Moreover, it is seen that the number of OpenCL kernel supporting CPU devices is significantly less as compared to GPU devices, and particularity to have several samples of a given class under-represented compared to other classes gives rise to the "class imbalance" problem. To handle this problem, we used SMOTE [27] to tackle the curse of imbalanced data. The performance of a classifier is measured with three evaluation

Table 2
Benchmarks details.

Suits	Benchmark	Input data size	Versions
AMD	Matrix multiplication	1,769,472–12,582,912	3
	Binomial options	32,768–294,912	9
	Bitonic sort	32,768–268,435,456	13
	Fast walsh transform	8,192–221,184	17
	Matrix transpose	131,072–536,870,912	6
	Discrete cosine transformation	2,097,152–1,887,436,800	17
	Floyd warshall	524,288–25,690,112	6
Polybench	3MM (3 Matrix Multiplications)	7,000,000–17,920,000	3
	GEMM (Matrix-multiply)	3,000,000–27,000,000	5
	GESUMMV (Scalar, vector and matrix multiplication)	8,012,000–1,800,180,000	17
	MVT (Matrix vector product and transpose)	4,016,000–900,240,000	17
	ATAX (Matrix transpose and multiplication)	4,012,000–900,180,000	17
	2MM (2 matrix multiplications)	5,000,000–45,000,000	5
	2DCONV (2D convolution kernel)	2000000–1,568,000,000	17
	3DCONV (3D convolution kernel)	1,000,000–1,728,000,000	17
Own developed	Matrix–vector multiplication	4,202,496—1,514,299,392	16

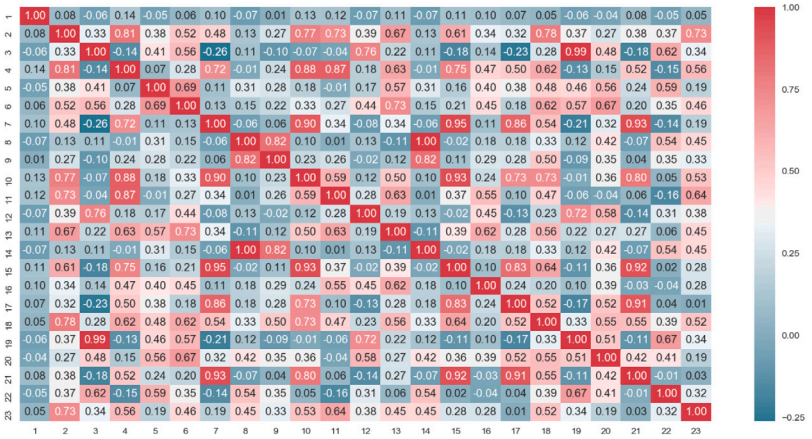

Fig. 5. Correlation analysis.

Table 3
Feature importance and reduced feature set.

Rank	No.	Feature	Information Gain
1	1	Data size	0.45
2	9	Total no of multiplication (integer) instruction	0.08
3	19	Total no of blocks	0.07
4	17	Total no of function call instruction	0.07
5	20	Total no of instructions	0.05
6	3	Total no of control statement	0.04

Table 4
Models output class and description.

Model	Type	Output
Device Suitability Predictor	Classification	Model needs to predict application suitability on specific CPU or GPU
Merging classifier	Classification	Model needs to predict application weather to merge two kernel or not

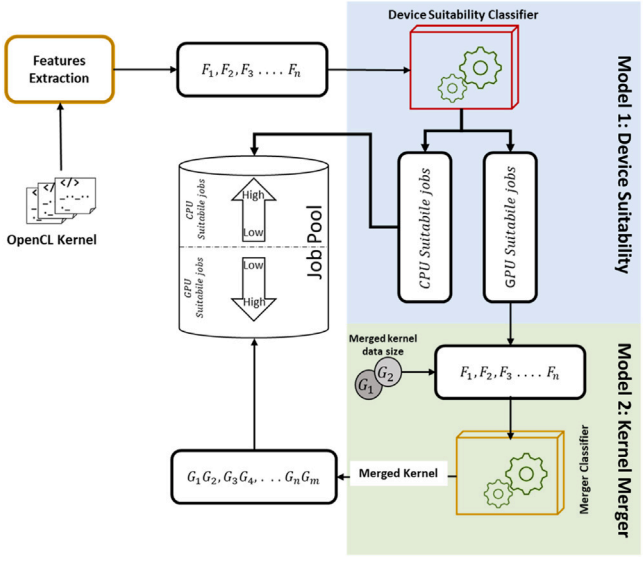

Fig. 6. A framework for the GPUGPU kernel merging.

metrics, for example, Precision, Recall, and F-measure. In Table 6, the SMOTE class balancing is performed. Table 6 demonstrates that when kernel selection is performed using the device suitability dataset, then TPOT produces the higher F-measure as compared to the other models. However, the performances of Naïve Bayes and KNN are the very low end, indicating that the kernel selection for the specific resource has an enormous impact on the performance of the classifiers.

In Table 6, the experiment is conducted on 16,000 samples using reduced features set for the merging classifier. We compared the results with another kernel merging method that used machine learning classification [26]. Table 6 demonstrates that when the reduced feature

Table 5
Device suitability model tune parameters.

Methods	Hyper tuning
Stacking estimator	estimator = GaussianNB()
Features selector	PCA (iterated-power = 7 ,svd-solver = "randomized")
Classification model	XGBClassifier (learning-rate = 0.5, max-depth = 8 ,min-child-weight = 1, n-estimators = 100, nthread = 1,subsample = 0.5)

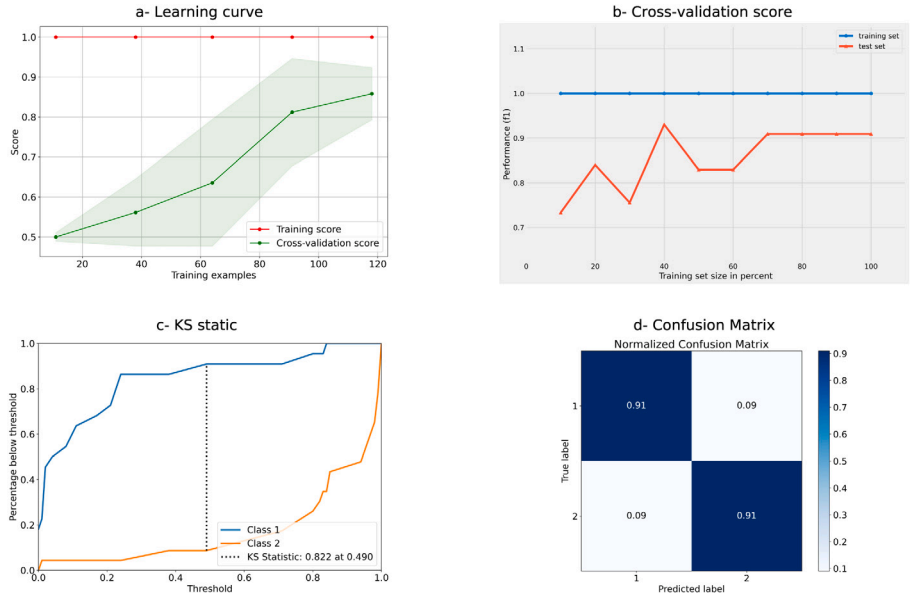

Fig. 7. Device suitability classification results.

Table 6
Reduced feature set device suitability and merger comparison with traditional classifiers.

Classifier	Device suitability		
	F-measure	Precision	Recall
Tpot	0.91	0.91	0.92
RandomForest	0.75	0.75	0.79
DecisionTree	0.50	0.56	0.50
Naïve	0.23	0.26	0.36
KNN	0.37	0.39	0.37
Classifier	Merger classifier		
	F-measure	Precision	Recall
Tpot	0.98	0.99	0.99
FusionCL [26]	0.98	0.97	0.99
RandomForest	0.93	0.94	0.94
Decision Tree	0.61	0.63	0.62
Naïve	0.62	0.58	0.59
KNN	0.58	0.59	0.58

set is used with device suitability, then TPOT produces the higher F-measure as compared to *FusionCL*, and others. FusionCL used the heuristic-based method for further load balance scheduling of the task. We only proposed using the Just In Time (JIT) compiler kernel selection method to improve speed up against single kernel execution. However, the other classifier is on the very low end, indicating that the reduced features have a large impact on the performance of the classifiers. The higher recall indicated that the tuned parameters returned the most relevant results. The higher precision shown in Table 6 indicates that the TPOT predicts the relevant class results more correctly than the irrelevant. The merger classifier has improved results of 0.98 as compared to device suitability of 0.91 F-measure. This indicates that the model after selection of device regarding the computing capability and data size of the kernel, accurate selections for kernel for merging is achieved with higher F-measure.

The reason behind the improved accuracy of the tree-based pipeline method is the usage of genetic programming to evolve the sequence of pipeline operators. The pipeline operators include (e.g., random forest, the operator selects the number of trees and KNN the weights (uniform or distribute) or the number of nearest neighbor). On another hand, the tree-based pipeline method used genetic programming to maximize the classification accuracy and minimize the model complexity [25]. Therefore, the method is based on Pareto optimization to optimize the two different objectives concurrently. Since the dataset varies in terms of the number of features, diverse samples, and patterns, therefore no globally optimal solution optimizes both objectives. The pipelines reproduction is done according to the NSGA-II selection method [25]. In this way, TPOT on each generation is able to add new pipelines operator to improve the fitness function and intelligent search for the high-performing ensemble learning method. The final output contains the optimized data transformations and machine learning model with maximized classification accuracy and minimize model complexity with references to the learning data.

In our case, traditional model hyper tuning is done concerning the grid search method whereas the proposed model output is mentioned in Table 6. The proposed meta-learning method has the base model, i.e., extreme gradient boosting and stacking layer of Naïve Bayes model. The extreme gradient boosting method uses the residuals and not the actual class labels. Therefore, the base estimators are regression trees, i.e., 100 in our case and are controlled with the scaled learning rate, i.e., 0.5 as mentioned in Table 6. This enables gradual improvement of the base model at each step. The naive base stacker incorporates strong model inference based on the independence assumptions. The data transformer in our case is PCA that helps to convert the correlated features into non-correlated ones. This optimization helps to achieve high accuracy as compared to the other traditional method. As traditional method suffers from data standardization issues, low resilience to overlapping patterns, and high sensitivity to noisy data. Another reason includes that model is not able to converge for the given training

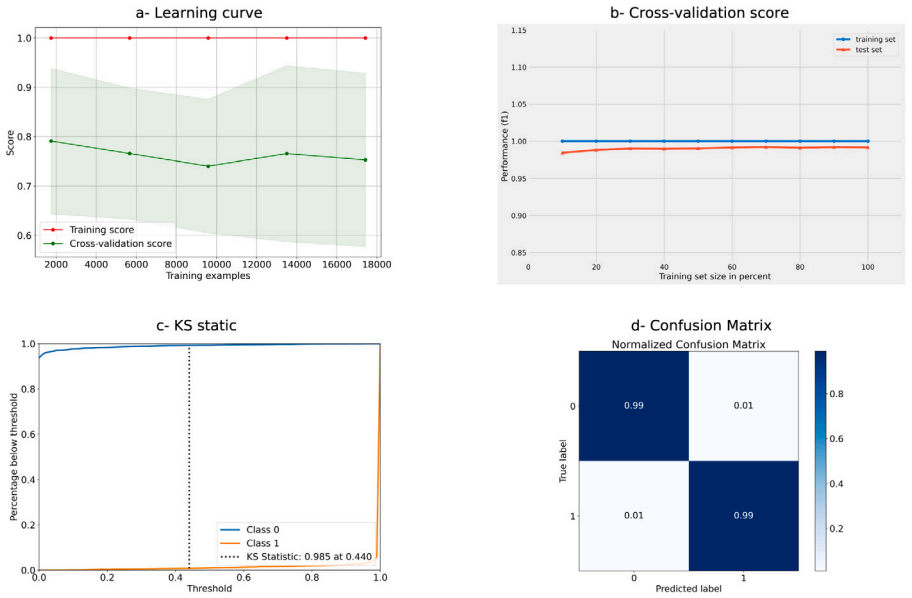

Fig. 8. Kernel merging classification results.

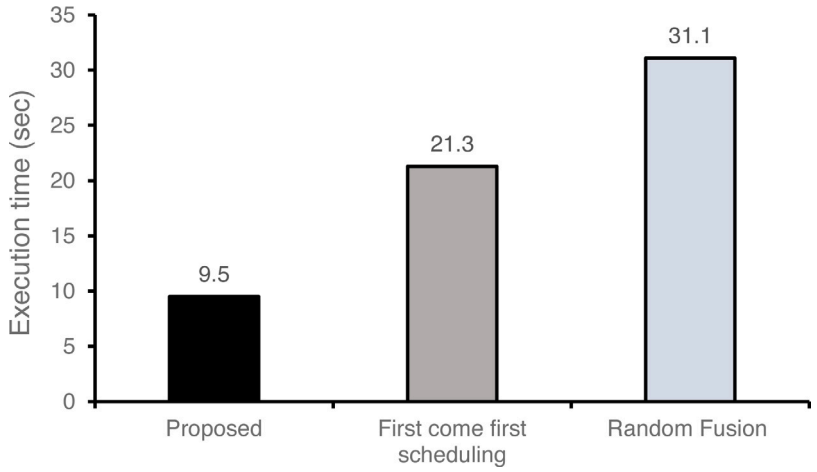

Fig. 9. Kernel merging method comparison with first come first and random fusion method.

samples due to instances are not linear separable with respect to the classes.

In Figs. 7 and 8, a summary of all the results is reported where the best classification model is used for both classifiers. The TPOT model achieved the highest classification score. The results are reported for the learning curve, cross-validation score, KS static test, and confusion matrix. Here, the higher value tells about the correct prediction of both classifiers. The KS test helps to compare the reference probability distribution between samples. Therefore, the higher value of KS, which is 0.82 and 0.98 in Fig. 7, shows that a suitability classifier can, mostly, correctly predict whether an application should be mapped to the processor. If the value is lower, it indicates that the suitability classifier is not exact and cannot identify whether an application is a device suitable or unsuitable. The scale of the F-measure is 1.0: perfect prediction, 0.9: excellent prediction, 0.8: good prediction, 0.7: mediocre prediction, 0.6: poor prediction, 0.5: and random prediction is less than 0.5 [28]. If the percentage value of Precision and Recall is lower for the suitability classifier, then applications are not suitable for processors. Such a processor would have resulted in longer job pool execution time and lower system throughput. The suitability classifier will only get a high F1-score if both Precision and Recall are high. Therefore, in our case, device suitability is 0.91, and the merging classifier has a 0.82 F1-score.

In Fig. 9, improvement of the kernel fusion on GPU is illustrated. The GPU mapped kernel is compared with other scheduling schemes. The X-axis represents scheduling methods whereas the Y-axis represents the execution time of the GPU mapped jobs. The proposed method represents the merging method that used the learning-based predictor based on job suitability. As seen the proposed model achieved a better performance after the kernel merging. The First Come First Serve (FCFS) method mapped the kernel according to its arrival time and executed it serially. The results show that the model achieved the 2.2 times reduced execution time when compared execution without predictor. The random fusion method assigns GPU kernel randomly merged. The indicates that the proposed technique achieved the reduced execution time. It also shows that GPU occupancy is improved with the kernel selection method. Therefore, the selection of pair of the kernel in the job pool improves its overall performance. The selection of kernels for merging and then merging with the appropriate method helps to improve utilization, the results shows. The considerable reduction is the result of the selection of appropriate kernels in the job pool.

5. Conclusion and future work

The use of GPU in computing systems has made GPU applications a new direction due to data parallelism. However, running concurrent kernels (GPU sharing between kernels) is not supported due to architecture limitations. This results in the wastage of precious resources. A kernel merging method is proposed in this paper to increase GPU utilization and reduce wastage of GPU energy consumption to assist in solving this problem. The machine learning (ML)-based kernel merging identifies a pair of kernels from the submitted batch of jobs as a result of merging their results in improving resource utilization. The extracted features help to produce a high F-measure for device selection and merging of the kernel with 0.91 and 0.98 values, respectively. The limitation of the approach is that the submitted data size should be in the GPU global memory range. In the future, we intend to use the extracted features to predict percentage usage of GPU resources that will be able to allocate a more efficient merged kernel. The proposed method can also be used as part of job selection for the load balance resource allocations. The impact and type of kernel can also be evaluated in future works to show which kernels execute most efficiently under certain loads.

CRediT authorship contribution statement

Usman Ahmed: Writing – original draft, Formal analysis, Investigation, Conceptualization, Methodology. **Jerry Chun-Wei Lin:** Writing – review & editing, Investigation, Supervision, Conceptualization. **Gautam Srivastava:** Writing – review & editing, Conceptualization, Investigation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] U. Ahmed, M. Aleem, Y. Noman Khalid, M. Arshad Islam, M. Azhar Iqbal, Ralbhc: A resource-aware load balancer for heterogeneous cluster, in: Concurrency and Computation: Practice and Experience, e5606.

[2] U. Ahmed, J.C.W. Lin, G. Srivastava, M. Aleem, A load balance multi-scheduling model for opencl kernel tasks in an integrated cluster, Soft Comput. (2020) 1–14.

[3] A. Liu, Y. Yang, Q. Xing, H. Yao, Y. Zhang, Improved collaborative particle swarm algorithm for job shop scheduling optimization, Adv. Sci. Lett. 4 (6–7) (2011) 6–7.

[4] A.-J.L. Liu, Y. Yang, Q.-S. Xing, H. Lu, Y.-D. Zhang, Multi-population genetic algorithm in multiobjective fuzzy and flexible job shop scheduling, Comput. Integ. Manuf. Syst. 17 (9) (2011) 1954–1961.

[5] Y. Liu, Ai-Junand Yang, Q.-s. Xing, H. Lu, Y.-D.Z. Zhang, Z.-Y. Zhou, G.-H. Wu, X.-H. Zhao, Dynamic scheduling on multi-objective flexible job shop, Comput. Integ. Manuf. Syst. 17 (12) (2011) 2629–2637.

[6] O.E. Albayrak, I. Akturk, O. Ozturk, Effective kernel mapping for opencl applications in heterogeneous platforms, in: The International Conference on Parallel Processing Workshops, 2012, pp. 81–88.

[7] C. Augonnet, S. Thibault, R. Namyst, P.-A. Wacrenier, Starpu: a unified platform for task scheduling on heterogeneous multicore architectures, Concurr. Comput.: Pract. Exper. 23 (2) (2011) 187–198.

[8] A. Ghose, S. Dey, P. Mitra, M. Chaudhuri, Divergence aware automated partitioning of opencl workloads, in: The India Software Engineering Conference, 2016, pp. 131–135.

[9] D. Grewe, M.F. O'Boyle, A static task partitioning approach for heterogeneous systems using opencl, in: International Conference on Compiler Construction, 2011, pp. 286–305.

[10] P. Huchant, M.-C. Counilh, D. Barthou, Automatic opencl task adaptation for heterogeneous architectures, in: European Conference on Parallel Processing, 2016, pp. 684–696.

[11] C.-K. Luk, S. Hong, H. Kim, Qilin: exploiting parallelism on heterogeneous multiprocessors with adaptive mapping, in: The Annual IEEE/ACM International Symposium on Microarchitecture, 2009, pp. 45–55.

[12] A.M. Aji, A.J. Peña, P. Balaji, W.-c. Feng, Multicl: Enabling automatic scheduling for task-parallel workloads in opencl, Parallel Comput. 58 (2016) 37–55.

[13] Y.N. Khalid, M. Aleem, R. Prodan, M.A. Iqbal, M.A. Islam, E-osched: a load balancing scheduler for heterogeneous multicores, J. Supercomput. 74 (10) (2018) 5399–5431.

[14] B. Pérez, J.L. Bosque, R. Beivide, Simplifying programming and load balancing of data parallel applications on heterogeneous systems, in: Annual Workshop on General Purpose Processing using Graphics Processing Unit, 2016, pp. 42–51.

[15] Y. Wen, M.F. O'Boyle, Merge or separate?: multi-job scheduling for opencl kernels on cpu/gpu platforms, in: The General Purpose GPUs, 2017, pp. 22–31.

[16] M. Becchi, S. Byna, S. Cadambi, S. Chakradhar, Data-aware scheduling of legacy kernels on heterogeneous platforms with distributed memory, in: ACM Symposium on Parallelism in Algorithms and Architectures, 2010, pp. 82–91.

[17] M.E. Belviranli, L.N. Bhuyan, R. Gupta, A dynamic self-scheduling scheme for heterogeneous multiprocessor architectures, ACM Trans. Archit. Code Optim. 9 (4) (2013) 57.

[18] A.P. Binotto, C.E. Pereira, A. Kuijper, A. Stork, D.W. Fellner, An effective dynamic scheduling runtime and tuning system for heterogeneous multi and many-core desktop platforms, in: IEEE International Conference on High Performance Computing and Communications, 2011, pp. 78–85.

[19] C. Gregg, M. Boyer, K. Hazelwood, K. Skadron, Dynamic heterogeneous scheduling decisions using historical runtime data, in: The Workshop on Applications for Multi-and Many-Core Processors, 2011, pp. 1–12.

[20] R. Kaleem, R. Barik, T. Shpeisman, C. Hu, B.T. Lewis, K. Pingali, Adaptive heterogeneous scheduling for integrated gpus, in: The International Conference on Parallel Architecture and Compilation Techniques, 2014, pp. 151–162.

[21] H.J. Choi, D.O. Son, S.G. Kang, J.M. Kim, H.-H. Lee, C.H. Kim, An efficient scheduling scheme using estimated execution time for heterogeneous computing systems, J. Supercomput. 65 (2) (2013) 886–902.

[22] K. Kofler, I. Grasso, B. Cosenza, T. Fahringer, An automatic input-sensitive approach for heterogeneous task partitioning, in: ACM International Conference on Supercomputing, 2013, pp. 149–160.

[23] Y. Wen, Z. Wang, M.F. O'boyle, Smart multi-task scheduling for opencl programs on cpu/gpu heterogeneous platforms, in: The International Conference on High Performance Computing, 2014, pp. 1–10.

[24] C. Lattner, Llvm and clang: Next generation compiler technology, in: The BSD Conference, Vol. 5, 2008, pp. 1–20.

[25] R.S. Olson, J.H. Moore, Tpot: A tree-based pipeline optimization tool for automating machine learning, in: Workshop on Automatic Machine Learning, PMLR, 2016, pp. 66–74.

[26] Y.N. Khalid, M. Aleem, U. Ahmed, R. Prodan, M.A. Islam, M.A. Iqbal, Fusioncl: a machine-learning based approach for opencl kernel fusion to increase system performance, Computing (2021) 1–32.

[27] N.V. Chawla, K.W. Bowyer, L.O. Hall, W.P. Kegelmeyer, Smote: synthetic minority over-sampling technique, J. Artificial Intelligence Res. 16 (2002) 321–357.

[28] F.A. Narudin, A. Feizollah, N.B. Anuar, A. Gani, Evaluation of machine learning classifiers for mobile malware detection, Soft Comput. 20 (1) (2016) 343–357.