Journal of Theoretical and Applied Information Technology
15th July 2018. Vol.96. No 13
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

STATIC MAPPING FOR OPENCL WORKLOADS IN HETEROGENEOUS COMPUTER SYSTEMS

[1]HENDRA RAHMAWAN, [2]KUSPRIYANTO, [3]YUDI SATRIA GONDOKARYONO

School of Electrical Engineering and Informatics, Institut Teknologi Bandung, Jalan Ganesha No.10, Bandung 40132, Indonesia

E-mail: [1]hrahmawan@gmail.com, [2]kuspriyanto@yahoo.com, [3]ygondokaryono@stei.itb.ac.id

ABSTRACT

Today, heterogeneous computer systems (HCS) commonly rely on CPU and GPU, for processing elements, and OpenCL, for the programming framework. In an HCS, a workload should execute on its best processor to achieve its best speedup. OpenCL currently entirely lefts the selection for the best-fit processor, termed as workload mapping, to programmers. However, the NP-completeness of the workload mapping task indicates it is not a trivial task to do manually by programmers so that effective computational approaches are necessary. This research proposes a static mapping method for OpenCL workloads that automatically select the best-fit processor for the workloads. The method accepts static features of a workload and utilizes K-Nearest Neighbor algorithm to classify the workload to either CPU or GPU. The static features are collected using LLVM/Clang compiler framework. To increase the accuracy of classification while keep maintaining the physical meaning of features, the features are reduced using feature selection approaches. Two feature selection models, filter model and wrapper model, are used in this research. This approach was evaluated using k-fold cross-validation against 18 OpenCL kernels obtained from standard benchmark packages. According to the evaluation results, the workload mapping accuracy was in the range of 93% to 97% indicating the method is well applicable in the HC environment with two processors. Floating-point operations and vector-integer operations, or floating-point operations and vector-global memory access are the combinations of features that a have significant contribution to the classification of workloads. The main contribution of the method in this research, compared to previous related research, lies in its capability to state features that are significant in the classification process.

Keywords: *Heterogeneous Computing, Workload Mapping, OpenCL, K-Nearest Neighbor, GPU*

1. INTRODUCTION

Heterogeneous computing systems (HCS), as a subset of high-performance computing systems, has been recognized as a new platform in computing and also offers a high gain in performance at a reasonable cost and power consumption [1-3]. Various computing workloads in different fields, ranging from trivial matrix-vector multiplication program to most complicated bioinformatics programs, share three common computing characteristics: data intensive, compute intensive, and control intensive [3]. Processing devices should be able to accommodate those heterogeneous workload characteristics, and heterogeneity of HCS inherently gives capabilities for processing different types of workloads which are not well addressed by conventional homogeneous computing systems [1].

HCS is usually built using existing off-the-shelf machines or processors. Thus, it will be relatively low-cost, comply with industry standards, and use proven technology. Basing to Flynn taxonomy for classifying parallel architectures [4], HCS can be defined as a synergetic use of processors having architectures of SISD (single instruction-stream single data-stream), SIMD (single instruction-stream multiple data-stream), MIMD (multiple instruction-stream multiple data-stream), or MISD (multiple instruction-stream single data-stream). As today it is difficult to find real SISD machines and no real implementations of MISD architecture, existing HCS systems incorporate only SIMD and MIMD architecture. SIMD machines are well suited for processing data-intensive and compute-intensive workloads, whereas MIMD machines would be able to process control-intensive workloads properly [3, 5].

MIMD architecture is generally represented by general purpose CPUs. On another spectrum of architectures, graphics processing units (GPUs), digital signal processing (DSPs) processors, and field programmable gate arrays (FPGAs) are some devices represent SIMD architecture. Those devices are also known as coprocessors or accelerators. With the rise of general-purpose computing on GPU (GPGPU), GPU becomes the most widely used accelerators in HCS. From another point of view, HCS could also be defined as the use of coprocessors to increase the performance of computing instead of using only a single type of CPU. Accelerator is a term used with coprocessor interchangeably since a few years back, describing its capabilities to accelerate workloads processing. A CPU will offload its workloads to accelerators to accelerate workloads processing.

In addition to hardware aspects, software frameworks are also required to support HCS in order to be optimally used by the programmer. OpenCL is an open standard of the HCS framework developed by several vendors, such as Intel, AMD, and NVidia, cooperating in Khronos consortium. Accelerators or coprocessors and general purpose CPUs are termed as devices in OpenCL. The support of OpenCL for wide range of devices makes it frequently used in HCS programming. However, the selection of devices to execute workloads in OpenCL is left entirely to the programmer. Programmers should be able to match their workloads with available devices, considering characteristics of workloads and devices, with the intention of achieving good performance. Inaccurate decisions by running workloads on unsuitable processing elements will bring workloads execution into suboptimal performance. This workload-processor matching or mapping problem is an NP-complete problem rather than trivial one [7]. Thus, not only write their programs, in these circumstances, programmers also have to handle this non-trivial additional tasks.

Unfortunately, existing software frameworks for HCS have less features for assisting programmers to determine the most suitable processing element for their workloads. Due to this mapping problem inherits NP-complete problem, an approach that can be used to overcome the problem is by predicting the best-fit processor for a given workload. Classification algorithms could be used to accomplish this task. This paper introduces an approach to map workloads to processors by using K-Nearest Neighbors (KNN) as the classification algorithm. The purpose of the approach is to select the best processor, either CPU or GPU, by prediction. In this research, KNN algorithm is used in conjunction with static profiling to collect the characteristic of workloads, and feature-selection algorithms to select high-correlated features.

This paper is organized as follows. Section 2 provides motivations for this research. Some related works in workload mapping are explained in Section 3. Section 4 covers some fundamental concepts about OpenCL and KNN algorithm. Proposed methodology in this research is described in section 5. Results of experiments in this research are presented and discussed in section 6. Section 7 summarizes the contributions of this research. Conclusions and possible future works of this research are elaborated in section 8.

2. MOTIVATION

This research is motivated by some findings that different mapping of workloads onto processors could bring into significantly different performance of workloads execution on HCS. In this section, six OpenCL kernels are examined by executing them on two different types of processor, namely CPU, and GPU. Figure 1 depicts the speedups of six OpenCL kernels, each of which is executed on CPU, and GPU.

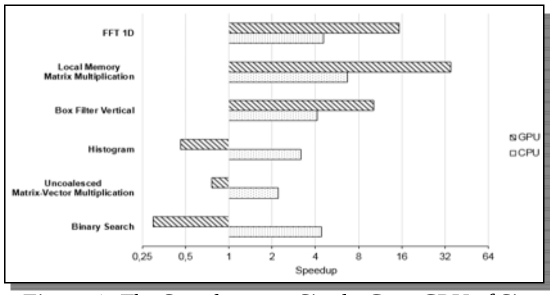

Figure 1: The Speedup over Single-Core CPU of Six OpenCL Kernels Running on Multi-Core CPU and GPU.

As depicted in Figure 1, there are two groups of workload that demonstrate different behavior regarding speedups they achieve both on CPU as well as GPU. The first group consists of FFT 1D, local-memory matrix multiplication, and box filter vertical. The second group consists of histogram, uncoalesced matrix-vector multiplication, and binary search.

The first group achieve the best performance when they run on CPU, whereas the last one achieve the best performance when they run on GPU. If the first group is assumed to run on GPU, then it will lose, on average, about three-fourth of their best speedup. On the other hand, if the second group run otherwise on GPU, it will lead to, on average, two times slower than its best performance.

Therefore, this clearly shows that different workload needs different mapping in order to achieve their best performance. If only a single mapping rule applied, for instance always run workloads on GPU or always run workloads on CPU, then, as previously described, it is probably will end up either in lower speed-up, or turn into slow-down.

3. RELATED WORKS

Some previous works were conducted to alleviate the problem of determining processors to execute workloads. Early research in the field of heterogeneous computing, like in [8], used interconnected single-core processors with different speed as their model. Some of them also studied mapping problems in simulation environments where the performance of all tasks on all processors is assumed known in advance.

Braun et al. [9] was comparing 11 heuristics used for static mapping of tasks on mixed-machine heterogeneous systems. Those heuristics were applied with estimated execution time (ETC) matrices to find the best heuristic for independent-tasks mapping on heterogeneous systems. A row in an ETC contains the estimated execution times for a given task in some machines, which also represents machine heterogeneity. The research concluded that genetic algorithm (GA) is the best heuristic in the heterogeneous mapping simulation. Other research in static task matching for heterogeneous systems also addressed dependent-task model, which is modeled using directed acyclic graph (DAG), like in [10] that also implements GA as its heuristic.

Wrensing and Greg [11] proposed an approach called elastic computing that introduced multi-implementation of workloads on various processor types, which was combined with linear-regression generated from workload profiles, in order to find the best processor type to execute workloads in question. Sandrieser et al. [12] initiated an external specification, called platform description language (PDL), as an abstraction for multiple different workloads implementations and multiple different hardware platforms, as the basis for the proposed approach for finding the most suitable processor to execute the workloads. Approaches in [11,12] share the common property, i.e. providing multiple different workload implementations is mandatory, which is limiting them to be widely applied.

Another methodology in workload mapping, proposed by Albyarak et al. [13], using three different main algorithms, i.e. greedy algorithm (GA), improved version of greedy-algorithm (IA), and mixed-integer programming (MIP), which all of them required input in the form of characteristics of kernels (workload). In this approach, the characterization of a kernel is accomplished by executing each kernel on each processors. This characterization technique is known as dynamic profiling, which has relatively higher cost than its static counterpart. Grewe and O'Boyle [6] and Tarakji et al. [7] introduced the use of machine learning to decide the most proper processor to execute a workload. Both of them used support vector machine (SVM), applied static-profiling to collect features from workloads, and used principal component analysis (PCA) to reduce the feature size. If it is necessary to know what features affecting mapping decisions, then the use of PCA cannot answer the question as it will transform features into new feature space.

4. FUNDAMENTAL CONCEPTS

Some basic theories about the OpenCL programming framework, and the KNN classification algorithm, employed in this research, will be elaborated briefly in this section.

4.1. The OpenCL Programming Framework

OpenCL standard specifies four models in the programming framework, namely platform model, execution model, memory model, and programming model. The platform model presents hierarchical organization abstracting the hardware in OpenCL. A platform consists of a host connected to one or more OpenCL devices. An OpenCL device is constituted by one or more compute units (CUs), which are further divided into one or more processing elements (PEs) [14]. The platform model also specifies that there is one processor coordinating execution (the host) and one or more processors capable of executing OpenCL C code (the devices) [3].

The execution model defines two terms for two different units of execution, namely kernels that run on one or more OpenCL devices, and a host program that runs on the host processor. The kernel term in OpenCL refers to where the main task, which is usually associated with computation, is placed. At run-time, kernels would be in the form of work-items that executes in groups, termed as workgroups. Relating to the platform model, these work-items would be run on OpenCL devices that are determined by the host program. A context is used to manage environment within which kernels run. A command-queue associated with a certain device is also set up in the context. To run a kernel on a certain device, a command is submitted to command-queue, in the form of N-dimensional index space, termed as NDRange. In a device, work-items in a workgroup would be further grouped into wavefronts that are executed in lockstep in a compute-unit [15].

The memory model defines organizational aspects of memory that control the interaction of values in memory as they are accessed from units of execution. It would help programmers to verify the correctness of their program. The model classifies memory into host memory and device memory, based on its direct-availability in a device or in a host. This model also specifies four regions of memory in an OpenCL device, namely global memory and constant memory that have scope in the device, local memory that is only accessible by work-items in the same workgroup, and private memory that is valid only for a work-item. Besides the specification of memory regions, the memory model of OpenCL also specifies about memory objects, shared virtual memory, and consistency model.

The last model in OpenCL is the programming model. It defines how the concurrency model is mapped to physical hardware. It abstracts a host processor and one or more OpenCL devices as a single heterogeneous parallel computer system. This model determines that an implementation of the OpenCL framework should consist of OpenCL platform layer, OpenCL runtime, and OpenCL compiler.

4.2. K-Nearest Neighbor (KNN) Classification

In the field of data classification, KNN algorithm falls into statistical learning class which its classification process is based on instances of data [16]. This algorithm is a lazy-learning algorithm because it delays the induction or generalization process until classification occurs. KNN is using the principle that instances in a dataset with similar properties are in general closer each other than with the others with different properties. In a supervised-classification case, which is data has a label associated with each of its instances, to find a label for an unclassified instance, KNN algorithm will discover k nearest neighbors and assign the single majority label to the instance. The general steps in KNN is described in Figure 2.

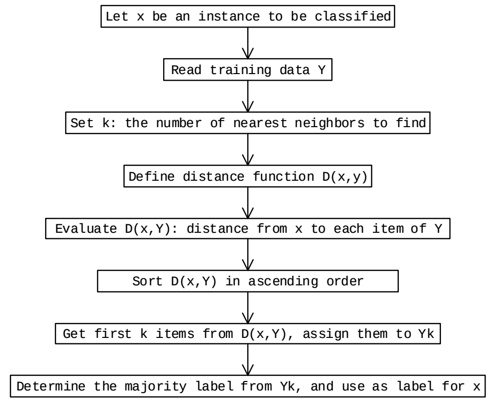
Let x be an instance to be classified
Read training data Y
Set k: the number of nearest neighbors to find
Define distance function D(x,y)
Evaluate D(x,Y): distance from x to each item of Y
Sort D(x,Y) in ascending order
Get first k items from D(x,Y), assign them to Yk
Determine the majority label from Yk, and use as label for x

Figure 2: Steps to Classify an Instance of Data Using KNN.

In the context of supervised classification, instances can be considered as points in an n-dimensional space. Proximity or distance between instances hence can be measured using some distance-functions [16] as presented in Table 1.

Table 1: Some Functions to Measure Distance.

Distance Function
Euclidian: $D(x,y)=\left(\sum_{i=1}^{m}\lvert x_i-y_i\rvert^2\right)^{1/2}$
Minkowski: $D(x,y)=\left(\sum_{i=1}^{m}\lvert x_i-y_i\rvert^r\right)^{1/r}$
Manhattan: $D(x,y)=\sum_{i=1}^{m}\lvert x_i-y_i\rvert$
Chebychev: $D(x,y)=\max_{i=1}^{m}\lvert x_i-y_i\rvert$
Camberra: $D(x,y)=\sum_{i=1}^{m}\frac{\lvert x_i-y_i\rvert}{\lvert x_i+y_i\rvert}$

5. METHODOLOGY

In this research, there are two main processes carried out to map a workload to two type of processors, namely a GPU and a CPU. Those processes are workload characterization and workload mapping, with the sequence of them is depicted in Figure 3.

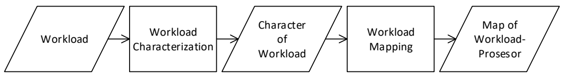

Figure 3: The sequence of stages in the proposed workload mapping approach.

The workload characterization process takes a workload in the form of OpenCL kernel as its input, determines the workload characters, and issues the character of the workload as the output. The character of a workload then passed to the workload-mapping process that will predict the most suitable processor for the workload. The output of the latest process is a pair of workload-processor that represents the processor in which the workload to run.

In this research, the number of involved workloads and processors in a single mapping step are assumed to be one workload and two processors with distinct types. This assumption is taken to observe the performance of this methodology in a simple mapping case. Further development and generalization for more complex mapping scenarios could be done based on the result of this research.

5.1. Workload Characterization

In this research, a characteristic of a workload is a collection of features related to a workload. To obtain values of these features, this research uses both static and dynamic profiling techniques. The static profiling approach is responsible for computing the frequency of occurrence of operations in a workload, whereas the dynamic profiling is responsible for collecting some dynamic aspects of workloads like input size and memory-transfer size.

Candidates of static features come from three types of operation: computation, memory, and control. As no support for I/O operation in GPU, the I/O operation type is not involved in this research. The rationale to involve those types of operations are all computations are commonly composed of them. This research uses input size and memory size (in bytes) for dynamic features for the reason they will determine the total size of instructions.

The workload features to use in this research are listed in Table 2. The static profiling approach uses an analysis-pass based on Clang/LLVM compiler framework. This analysis pass reads an OpenCL kernel that has been compiled into LLVM intermediate-representation (IR) and computes the frequency of certain operations in the IR.

In this research, the operations are classified into ordinary operation and complex operation. The frequency of occurrence for ordinary operations, like scalar integer operations, floating-point operations, etc., can be computed using a simple algorithm. The algorithm uses a counter variable for each operation under observation. The algorithm will increase the counter variables when it encounters the associated operations.

Table 2: Features of A Workload Acquired Using Static and Dynamic Profiling Techniques.

Operation Type	No.	Feature Name	Feature Type
Arithmetic	1	Scalar Integer Operations	Static
	2	Floating-point Operations	
	3	Vector Integer Operations	
	4	Vector Floating-point Operations	
Memory	5	Scalar Local Memory Access	
	6	Vector Local Memory Access	
	7	Scalar Global Memory Access	
	8	Vector Global Memory Access	
	9	Coalesced Memory Access	
Control	10	Barriers	
	11	Conditional Branch	
	12	Divergent Branch	
	13	Conditional Branch Cost	
	14	Unconditional Branch	
N/A	15	Input Size	Dynamic
	16	Memory Size	

A complex operation is an operation with special meaning. The complex operations are coalesced-memory-access operations and divergent-branch operations. A coalesced-memory-access operation allows adjacent threads to concurrently access values residing in adjacent memory locations. A coalesced-memory-access operation is actually an ordinary memory-access operation, but some special treatments take place when a processor executes it. One of the indications whether or not a memory access is a coalesced one is whether or not the memory access uses thread-index or workgroup-index. A divergent-branch instruction is a conditional-branch instruction which makes threads follow a different path of executions. A conditional branch is possibly divergent when it includes thread-index or workgroup-index in its conditional part.

In this situation, to determine whether or not a memory operation is a coalesced-memory instruction and whether or not a conditional-branch instruction is a divergent-branch instruction, some additional computations should be carried out. The computations consider the indication that both operations use thread-index or workgroup-index.

The flow of workload characterization is started with compiling an OpenCL source file into an IR and storing it in a text file. The next step is running the analysis pass against the text file containing the IR. The analysis-pass could be run using the opt tool in the LLVM framework. The analysis-pass runs the workload characterization process using the algorithm described before. The output of the analysis-pass is a file containing the values of the features of the workload.

5.2. Workload Mapping

The objective of this step is to decide which processor to run a workload. This task is accomplished using the KNN classification algorithm. The input for this step is a file containing features of workloads. This step consists of three sub-steps, namely normalization, feature selection, and classification.

5.2.1. Normalization

Due to the limited number of available workloads, and the need of a large enough sample size for data classification, to increase the sample size of the workload, all of the workloads are run with varied input size. This strategy gives identical feature values for groups of workloads, except for the input size feature and the memory access size feature.

To distinguish between two sets of feature values from the same workload with different input sizes, the total number of operations in the workload can be used. By multiplying size of operations in workload code with the number of work-items, the total number of operations in the workload execution could be estimated. To inform the classification algorithm about the relationship of sets of feature values that come from the same program but with different input size, the total number of operations is then divided by the memory access size. This sequence of computations is called normalization, introduced by Grewe & O'Boyle [6], and formulated in Eq. (1).

$$\text{normalized value} = \text{operations size} \times \frac{\text{number of workitems}}{\text{data transfer size}} \quad (1)$$

5.2.2. Feature Selection

To increase the performance of KNN classification, a sequence of steps, called feature selection, is carried out to select only important features of OpenCL kernels. The important features have high relevance to the class label assignment in the workload data. In this case, the important features are very decisive whether a workload is better to run on CPU or GPU. The feature selection only run once as the initialization process of the classifier, and it does not run in the classification step.

This research uses two feature selection models. The first model is filter model, which is independent of the classification algorithm. The second one is wrapper model, which uses a classification algorithm to produce selected features. The sequence of filter model and wrapper model is depicted in Figure 4.

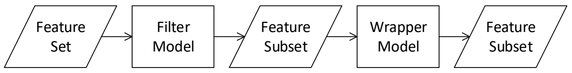

Figure 4: The Sequence of Filter Model and Wrapper Model in Feature Selection Process.

Filter model will compute ranks for each feature of OpenCL workloads. A dataset containing records of OpenCL kernels with their feature values is the input for this process. In this research, filter model uses two metrics, Pearson Correlation Rank [17] and Fisher Score [18], as the rankings for features. The rationale for the use of both metrics is to give better confidence about the order of the features produced by the filter model. In this research, the filter model is also set to feed the wrapper model, the next step of feature selection, with reduced feature size that will decrease computing cost of it. Features that have been ranked by filter model are then pruned by selecting *m* best features. The *m* best features can be selected by setting a threshold either objectively or subjectively. Eq. (2) and Eq. (3) formulate Pearson Correlation Rank and Fisher Score respectively.

$$R(i) = \frac{\sum_{k=1}^{m}(x_{k,i} - \bar{x}_i)(y_k - \bar{y})}{\sqrt{\sum_{k=1}^{m}(x_{k,i} - \bar{x}_i)^2 \sum_{k=1}^{m}(y_k - \bar{y})^2}} \quad (2)$$

$$F(x^j) = \frac{\sum_{k=1}^{c} n_k(\mu_k^j - \mu^j)^2}{\sum_{k=1}^{c} n_k(\sigma_k^j)^2} \quad (3)$$

Wrapper model will further select the features produced by filter model, by evaluating them against a classification algorithm. The rationale for using wrapper model is to select features that fit with the classifier in the classification phase. In this research, wrapper model employs KNN as its classification algorithm. Wrapper model works iteratively, which in each iteration the wrapper model will select a feature subset, evaluate it, and decide whether to continue to the next iteration or to terminate the iteration based on a stopping condition. The stopping condition used in this research is if the entire feature-subsets have been evaluated. The feature-subset with the best accuracy in KNN classification then will be selected for the output of wrapper model. Figure 5 depicts the flow of wrapper model for feature selection.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

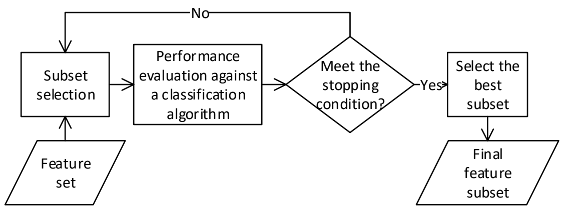
No
Subset selection
Performance evaluation against a classification algorithm
Meet the stopping condition?
Yes
Select the best subset
Feature set
Final feature subset

Figure 5: The Flow of Feature Selection with Wrapper Model.

5.3. Classifier Evaluation

The standard cross-validation technique is used to evaluate the performance of KNN classification for workload mapping. Among variants of cross-validation technique, this research uses n-fold cross-validation technique. In the n-fold cross-validation, the data is randomized and partitioned into *n* equally sized subsets. In an iterative way, in the *i*th iteration, the *i*th subset is selected as the test set and the rest *n*-1 subsets are merged to form the training set. In each iteration, the performance of KNN classifier is measured, usually using loss or accuracy as the metric, and all of the performance values are averaged at the end of entire iterations. It is important to note that none of the data in the test set is used for training the model.

5.4. Experimental Setup

In this research, experiments were conducted with the objective to evaluate the performance of workload mapping using KNN classification. All experiments were carried out on a heterogeneous computer with hardware configuration described in Table 3.

Table 3: Hardware Configuration for All Experiments.

	CPU	GPU
Type	Intel Core i7-2600k	NVIDIA Tesla C2075
Core clock	3.4 GHz	1.15 GHz
Number of cores	4 (8 with HyperThreading)	448
Memory sizes	8 GB	6 GB
Memory transfer speed	21 GB/s	144 GB/s
OpenCL platform	Intel® OpenCL	NVIDIA CUDA

The workload samples were selected from existing software packages: AMD APP SDK and CUDA SDK. There were 18 OpenCL kernels that each of them executed with four different inputs, producing 72 kernel-executions. Each of 72 kernel configurations was executed 20 times on CPU and GPU, resulting averaged workload execution times on CPU and GPU. Either CPU or GPU, which gave the best performance in executing a workload, was set as the label for the workload in the data set. The static feature values of the kernel are also collected to form the dataset along with the predefined label.

The *K* in the KNN classifier was set to 1 and 3. These values were based on initial experiments that evaluated the *K* values of 1, 3, 5, 7, and 9 and showed that the best KNN performance, in the workload mapping case, is given by K-values of 1 and 3. To evaluate the KNN classifier performance during feature selection process using wrapper model, n-fold cross-validation was used with *n* was set to 10. The final assessment of the KNN classifier for workload mapping also utilized n-fold cross-validation with the *n* was set to 4 and 6, considering the amount of test data and training data. In addition to the original order of the data set, permutations were also applied to the data set. This was done to observe the effect of permutations as the consequence of the reuse of workloads to construct the data set.

6. RESULTS AND DISCUSSION

This section shows and discusses the results of feature selection process and final assessment of the workload mapping process.

6.1. Feature Selection

It has been found from experiments, if the entire 14 features were used for classification using KNN, the average accuracy was 88.96%. The accuracy could be increased by applying feature selection process.

As mentioned in previous chapter, two models of feature selection were involved, filter model and wrapper model. The results of filter model that computed Pearson coefficient and Fisher score for the 14 features are depicted in Figure 6. From the results, by specifying the threshold for Pearson coefficient and Fisher score by 0.2 and 0.1

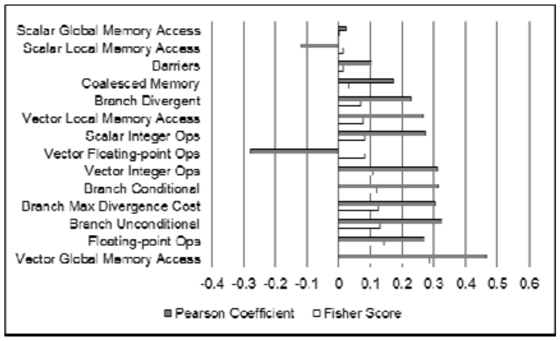

Figure 6: Pearson Coefficient and Fisher Score for 14 Initial Features.

respectively, it was obtained six selected features as listed in Table 4. These thresholds were set subjectively by observing the values and estimating the number of features should be selected.

Table 4 also indicates the numbering of the features for the purpose of referencing. The selected features were further selected by the wrapper model that used KNN classifier with K=1 and K=3. Figure 7 and Figure 8 describe the performance of KNN classification in wrapper model against varying features and feature sizes. Figure 7 and Figure 8 were obtained from KNN classifier with K=1 and K=2 respectively. For each subset of each possible subset size from the entire available feature subsets, only two best combinations of features are presented in both figures. From the figures, it is clear that the use of two features, either floating-point operations and vector integer operations (feature 4 and 5), or floating-point operations and vector global memory access (feature 4 and 6), in KNN classifier give best performances with the accuracy is more than 96%. These facts could also be a recommendation for programmers to notice these three deterministic features in their program: floating-point operations, vector integer operations, and vector global memory access.

Table 4: Features selected using filter method.

Feature Number	Feature Name
1	Branch Conditional
2	Branch Max Divergence Cost
3	Branch Unconditional
4	Floating-point Operations
5	Vector Integer Operations
6	Vector Global Memory Access

6.2. Workload Mapping

The results of experiments that evaluate the KNN classifier that performed workload mapping task are summarized in Table 5. The results show that the accuracy is ranging from 93.1% to 100%. The lowest accuracy was obtained when permutation was not applied to the dataset. It was occurred due to data items obtained from the same workload were in contiguous location. This circumstance made selected testing dataset possibly had no representative dataset in the training dataset, therefore misclassifications were likely occurred.

The highest accuracy was achieved with the permuted dataset. The permutation distributed data items to the testing dataset and the training dataset. The distribution would make data items come from each workload more probable to reside in the testing dataset and the training dataset. Hence, evaluation with these datasets ended up with accurate classification.

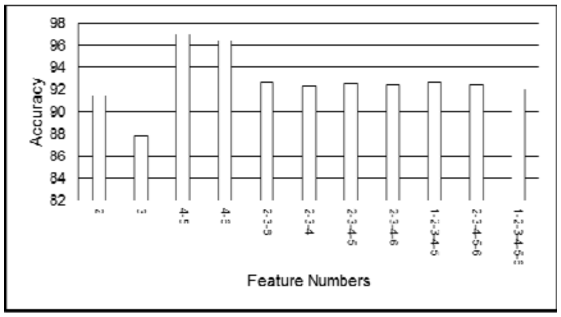

Figure 7: Performance of KNN Classifier in Wrapper Model with K=3.

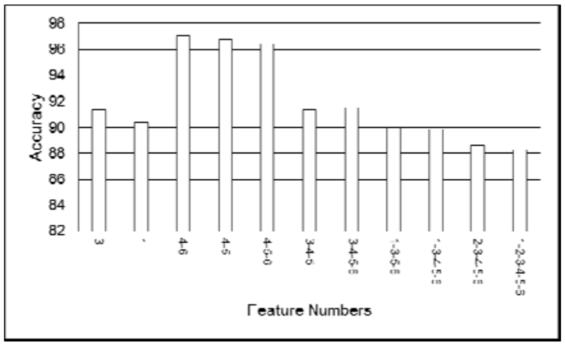

Figure 8: Performance of KNN Classifier in Wrapper Model with K=3.

Table 5: Summary of Evaluation of the KNN Classifier

Testing Scenarios	Data Permutation	Testing Parameters			Accuracy (%)
		Training Dataset Size	Testing Dataset Size	K in KNN	
Manually Splitting of Training-Testing Dataset	Yes	60	12	3	100
		54	18		100
	No	60	12		98.23
		54	18		98.90
K-Fold Cross Validation		K(K-Fold Cross Validation)			
	Yes	4			97.22
		6			97.22
	No	4			93.06
		6			94.44

The results of experiments using the testing dataset with the size of 12 and 18, without permutation, are presented in Table 6 and Table 7. Normalized speedup is used to compare speedups of each kernel when it was executed on CPU, GPU, and the processor that was selected by the KNN classifier against its best speedup. KNN classification that used both of testing datasets failed to classify HorizontalSAT0 kernel with the input size of 1024. However, although the KNN classifier failed to classify the kernel, the performance of the misclassified kernel is so close to the optimal one. When the kernel was executed on the processor that was selected by the KNN classifier, it gained the speedup of 0.999 relative to its best speedup. This result of experiments indicates that the KNN classifier can be used to determine the most suitable processor to execute a kernel with high classification accuracy.

Table 6: Performance Achieved by 12 Kernels When They Were Mapped Using Always-CPU, Always-GPU, and KNN-Classifier.

No.	Kernel	Normalized Speedup		
		CPU	GPU	Mapping
1	DCT_2048_2048	1.000	0.997	1.000
2	MatrixMultiplication_4096	0.074	1.000	1.000
3	MatrixMultiplicationLocal_2048	0.164	1.000	1.000
4	OclMatVecMulCoalesced1_2048	0.077	1.000	1.000
5	OclMatVecMulCoalesced2_1024	0.091	1.000	1.000
6	HorizontalSAT0_2048	0.900	1.000	1.000
7	VerticalSAT_2048	0.694	1.000	1.000
8	HorizontalSAT0_1024	1.000	0.999	0.999
9	box_filter_vertical_1024	0.396	1.000	1.000
10	box_filter_horizontal_1024	0.398	1.000	1.000
11	histogram_256_1024	1.000	0.959	1.000
12	histogram_256_2048	1.000	0.267	1.000

Table 7: Performance Achieved by 18 Kernels When They Were Mapped Using Always-CPU, Always-GPU, and KNN-Classifier.

No.	Kernel	Normalized Speedup		
		CPU	GPU	Mapping
1	DCT_2048_2048	1.000	0.997	1.000
2	BinarySearch_67108864	1.000	0.010	1.000
3	MatrixMultiplication_4096	0.074	1.000	1.000
4	MatrixMultiplicationLocal_2048	0.164	1.000	1.000
5	OclMatVecMulUncoalesced0_8192	1.000	0.392	1.000
6	OclMatVecMulCoalesced0_8192	0.174	1.000	1.000
7	OclMatVecMulCoalesced1_1024	0.076	1.000	1.000
8	OclMatVecMulCoalesced1_2048	0.077	1.000	1.000
9	OclMatVecMulCoalesced2_1024	0.091	1.000	1.000
10	HorizontalSAT0_2048	0.900	1.000	1.000
11	VerticalSAT_2048	0.694	1.000	1.000
12	HorizontalSAT0_1024	1.000	0.999	0.999
13	box_filter_vertical_1024	0.396	1.000	1.000
14	box_filter_horizontal_1024	0.398	1.000	1.000
15	histogram_256_1024	1.000	0.959	1.000
16	histogram_256_2048	1.000	0.267	1.000
17	histogram_256_8192	1.000	0.146	1.000
18	AESEncrypt_4096	0.025	1.000	1.000

7. SUMMARY OF CONTRIBUTIONS

In contrast to the related works [6] and [7] explained in Chapter 3, this research proposes a mapping approach that can describe the contribution of features of workloads in the mapping process, which is accomplished through the use of feature selection. The use of KNN classifier makes the mapping approach potentially more scalable as the KNN algorithm is inherently a multiclass classifier.

8. CONCLUSION AND FUTURE WORKS

The heterogeneity of OpenCL workloads and processors in heterogeneous computer systems makes workload mapping task to select the most suitable processor for a certain workload very crucial for the performance of the workload execution. The contribution of this research is a new solution for the workload mapping problem by utilizing KNN algorithm that was improved using feature selection. This approach is able to help programmers to determine processors to execute their workloads optimally, which is not a trivial task to perform manually.

This mapping approach used static features of workloads for the classification process. There were initially 14 static features which were reduced to a lower size using feature selection process. The feature selection in this approach employed two models, namely filter model and wrapper model. These models were invoked in sequence and have refined the accuracy of the KNN classification up to 12%. The combined features of floating-point operations and vector integer operations, or floating-point operations and vector global memory access are empirically gain the highest accuracy in the workload-processor mapping.

The experiments that were conducted using 18 workloads, picked from standard benchmark packages, have shown the accuracy of classification ranging from 93.1% to 100%. This range of value indicates lower and upper bound for the accuracy of the workload mapping approach. This range of value also suggests that the KNN classifier accurately map workloads to two heterogeneous processors. Although this result was obtained from mapping with two processors target, it is possible to increase the number of the processor to more than two processors.

This work has not yet addressed the problem of scheduling a set of workloads into a set of processors. Thus this scheduling case is left for the future works. There are also some open problems related to this field of research. One to consider is about involving other performance parameters in the mapping approach such as power consumptions, and load balancing. Another mapping scheme, like dynamic mapping, also has possible advantage over current approach, so it is worth to explore. The proposed mapping approach is also possible to extend to handle multi-kernel scheduling problem.

REFERENCES:

[1] Khokhar A.A., Prasanna V.K., Shaaban, M.E., & Wang, C.L., "Heterogeneous computing: Challenges and opportunities", *Computer* 26(6), June 1993, pp. 18-27.

[2] Kumar R., Tullsen D.M., Jouppi N.P., & Ranganathan P., "Heterogeneous chip

multiprocessors", *Computer* 38(11), Nov. 2005, pp. 32-38.

[3] Gaster B., Howes L., Kaeli D.R., Mistry P.,, & Schaa D., "Heterogeneous Computing with OpenCL: Revised OpenCL 1.2", Newnes, 2012.

[4] Flynn M.J., "Some computer organizations and their effectiveness", *IEEE transactions on computers* 100(9), Sept. 1972, pp. 948-960.

[5] Hennessy, J.L., & Patterson D.A., "Computer architecture: a quantitative approach", Elsevier, 2011.

[6] Grewe D., & O'Boyle M.A., "Static task partitioning approach for heterogeneous systems using OpenCL", *Compiler Construction*, Springer Berlin/Heidelberg, 2011, pp. 286-305.

[7] Tarakji A., Salscheider N.O., Alt S., & Heiducoff J., "Feature-based device selection in heterogeneous computing systems", *Proceedings of the 11th ACM Conference on Computing Frontiers*, ACM, May 2014, pp. 9.

[8] Ibarra O.H., & Kim C.E., "Heuristic algorithms for scheduling independent tasks on nonidentical processors", *Journal of the ACM (JACM)* 24(2), April 1997, pp. 280-289.

[9] Braun T.D., Siegel H.J., Beck N., Bölöni L.L., Maheswaran M., Reuther A.I., Robertson J.P., Theys M.D., Bin Y., Hensgen D., & Freund R.F., "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems", *Journal of Parallel and Distributed computing* 61(6), Elsevier, June 2001, pp. 810-837.

[10] Wang L., Siegel H.J., Roychowdhury V.P., & Maciejewski A.A., "Task matching and scheduling in heterogeneous computing environments using a genetic-algorithm-based approach", *Journal of parallel and distributed computing* 47(1), Nov. 1997, pp. 8-22.

[11] Wernsing J.R., & Stitt G., "Elastic computing: a framework for transparent, portable, and adaptive multi-core heterogeneous computing", *Proceedings of the ACM SIGPLAN/SIGBED 2010 conference on Languages, compilers, and tools for embedded systems*, April 2010, pp. 115-124.

[12] Sandrieser M., Benkner S., & Pllana S., "Using explicit platform descriptions to support programming of heterogeneous many-core systems", *Parallel Computing* 38(1), Jan. 2012, pp. 52-65, Elsevier.

[13] Albayrak O.E., Akturk I., & Ozturk O., "Improving application behavior on heterogeneous manycore systems through kernel mapping", *Parallel Computing* 39(12), Elsevier, Sep. 2013, pp. 867-878.

[14] Khronos OpenCL Working Group, "The OpenCL Specification version 2.0", July 2015.

[15] AMD, "AMD Accelerated Parallel Processing OpenCL Programming Guide", Nov. 2013.

[16] Kotsiantis S.B., "Supervised machine learning: A review of classification techniques", *Informatica* 31(3), Nov. 2007, pp. 249-268.

[17] Guyon I., & Elisseeff A., "An introduction to variable and feature selection", *Journal of machine learning research* 3(3), Mar. 2003., pp. 1157-1182.

[18] Tang, J., Alelyani S., & Liu H., "Feature selection for classification: A review", *Data Classification: Algorithms and Applications*, CRC, 2014, pp. 44.