Feature-based Device Selection in Heterogeneous Computing Systems

Ayman Tarakji Niels Ole Salscheider Stephan Alt Jan Heiducoff
Faculty of Electrical Engineering and Information Technology, RWTH Aachen University
Aachen, Germany
{tarakji,salscheider,salt,heiducoff}@lfbs.rwth-aachen.de

ABSTRACT

With the advent of accelerator-based heterogeneous parallel systems, the need for a solution of the task-device matching problem is increasing. Due to the enormously growing diversity in existing computing architectures, optimal matching promises to deliver high performance at reduced energy costs. By means of OpenCL and particularly the LLVM compiler infrastructure, our approach makes the task-device matching decisions taking into account the characteristics and particularities of the different processing hardware. We evaluate our approach using a set of OpenCL based real-world applications and well established benchmarks, which are run on different hardware platforms and architectures. Our results indicate highly accurate predictions made by our model during the matching procedure.

Keywords

OpenCL, GPGPU, LLVM Compiler, static code analysis, heterogeneous parallel systems, GCN, Kepler

1. INTRODUCTION

In the field of GPU-based heterogeneous computing, investigating the load characteristics of applications on the one hand, and analyzing the properties of the processing devices on the other are both necessary to ensure highly efficient execution. For instance, the number of stream operations indicates on which device (CPU, GPU) a computation task will execute faster [6]. Thus, the need for a unified programming model is increasing, which can deal with the heterogeneity problem of existing computing architectures and achieve high performance and maximal utilization of existing resources. Besides being an open standard, the unified parallel programming model *OpenCL* provides a unique benefit due to its ability to target a variety of devices. A group of properties is provided in this context allowing the programmer to specify which types of devices they are interested in (`CPU`s, `GPU`s or `ACCELERATOR`s) [13].

CF'14, May 20 - 22 2014, Cagliari, Italy
Copyright 2014 ACM 978-1-4503-2870-8/14/05... $15.00.
http://dx.doi.org/10.1145/2597917.2597927
.

This paper presents a technique to automatically achieve the task-device mapping [8] in a heterogeneous computing environment. It uses a machine-learning method based on code features of OpenCL programs, which are extracted during compile time. At first, a static code analysis is performed on the intermediate representation of LLVM (Low Level Virtual Machine) [11], which is a modular compiler framework written in `C++` and deployed by the OpenCL driver. LLVM represents a collection of compiling and tool chain technologies, containing libraries, compilers, and code generators. Thus, LLVM's accessory suits our project's requirements and demands very well. In the following step, using a statistical model and based on the extracted code features, a machine-learning predictor determines the optimal device for a given task.

In order to give a realistic evaluation, we show that our model is able to predict the device's suitability of a set of real-world applications, even when only few programs are involved in the training. With outstanding accuracy, we demonstrate the ability of our predictor to classify a variety of programs as "CPU-only" or "GPU-only". In addition, the well established Parboil benchmark suite [14] is considered in our experiments.

To emphasize the portability of our approach, we run the same tests on different groups of programs deploying two different computing platforms and GPU architectures. On the one hand, the use of the different platforms should bring the evidence, that the machine-learning technique used by our predictor is adaptive to a variety of accelerator-based computing machines. On the other hand, the consideration of different groups of programs shows, that our predictor is not limited to a certain class of algorithms.

This paper is organized as follows: In the following, a review of the motivation and related work are provided. In section 4, we present the design and methodology of our feature-based predictor. After a summary of contribution in section 5, section 6 includes a variety of experiments achieved on our predictor, and section 7 concludes the paper.

2. MOTIVATION

In general, a great distinction among computing architectures exists. In order to always use available resources to the full potential, appropriate methods are required to accurately determine the optimal work distribution among the different execution units in a heterogeneous system. We believe, that due to the architectural diversity and the efficiency issue in such systems, the decision where to run a given task should be made a priori.

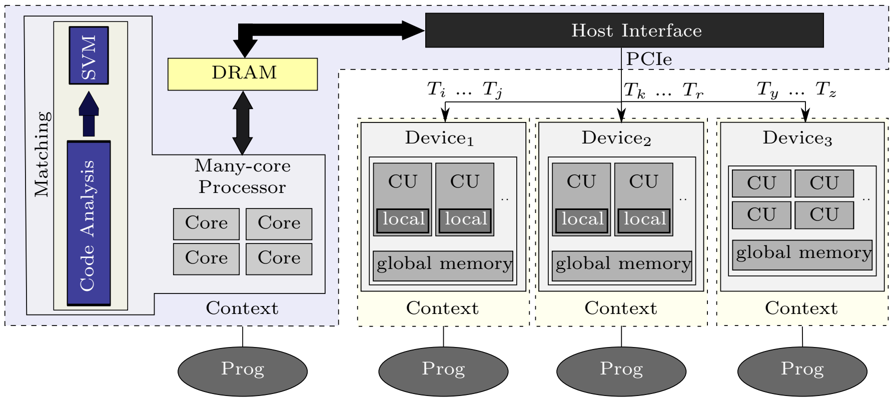


Figure 1: An exemplary OpenCL-based model including task-device mapping in a heterogeneous environment.

We introduce a predictor that performs the task-device matching in a highly diverse computing environment, providing for a broad definition of modern accelerator architectures. In fig. 1, an example of the matching procedure is illustrated in connection with such a heterogeneous system mapped to the OpenCL platform model.

In the context of this paper, we are currently in the process of discussing our predictor's usage with two GPU architectures: *NVIDIA's Kepler* and *AMD's Graphics Core Next* (GCN), both represent the latest innovation in the field of general purpose computing accelerators. However, although the most widely studied variant thus far has been a certain mixture of CPU+GPU system architecture (as will be seen from the following discussion in section 3), the long term goal of this approach is to extend the predicting method to existing accelerators other than GPUs. For instance, an interesting comparison of GPU architectures would be with the *Intel* Many Integrated Cores (MIC) architecture regarding the task-device mapping decisions. We have already performed several tests on the *Intel* Xeon Phi accelerator, however, we let this discussion for future work.

In a relevant context [16], we introduced a preemptive scheduling method *OCLSched* for the purpose of utilizing the computing resources of any existing OpenCL device in a heterogeneous system, as soon as they become available. Its major function is to manage the execution of multiple tasks on different OpenCL devices in a system-wide view, and to provide for multi-user by means of the well established server-client model. A combination of such a time-sharing scheduler with the predictor presented in this paper, would provide a comprehensive unit that runs in background and manages the distribution and execution of tasks centrally, exhausting the available computing units in any accelerator-based heterogeneous system. Such a combination will be discussed in future work.

3. RELATED WORK

In the CPU-GPU heterogeneous computing era, many efforts have been taken to provide robust and efficient computing environments. Several dynamic methods to predict the run-time behavior of a variety of applications have been proposed [8, 10, 18]. These were based on either analytic benchmarking or code profiling. The concept treats the execution time of a task as a random variable and makes a prediction based on past observations.

Similarly, a further approach to estimate execution time by means of a hybrid method was presented [8]. Using analytic benchmarking, performance discrepancies of multiple computing devices were characterized on the basis of a series of special benchmarks. Then, in combination with past observations, these characteristics were used in a statistical method to obtain the execution time estimates for each task. In that strategy, a lot of past observations were required in order to improve the accuracy of predictions, but at very high computation costs of the prediction algorithm. Otherwise, the prediction errors ranged around 50%, although the evaluation was restricted to a small set of applications. Another dynamic approach *Harmony* [4] explored the matching of a kernel program to a device based on their conformity. Using a profiling model to predict the performance of a program during runtime, the approach scheduled the entire task to a certain device.

In contrast to dynamic methods, run-time overhead is pull-forwarded to compile time if static techniques are used. In the related work, such an approach was proposed for classifying whether an OpenCL kernel should run on a CPU or a GPU [7]. Additionally, in case a task was suitable for both the CPU and the GPU, an estimate how to distribute the work between them was given. For this purpose, a machine-learning technique was used in order to predict the optimal partitioning $p = f(\boldsymbol{c}) \in [0, 10]$ based on a *feature vector* $\boldsymbol{c}$. The features in $\boldsymbol{c}$ were extracted from a kernel using a static code analysis on the compiler level.

In this paper, we similarly introduce a static approach based on predictive modeling of device suitability, but, in contrast to [7]:

- instead of the AST (Abstract Syntax Tree) generated by *CLang* (LLVM's front-end for C-like languages), we focus on strengthening the portability of our method, thus, our predictor uses the LLVM's IR (Intermediate Representation) to extract code features independently from any special front-end.
- since our focus is on providing for a wide range of existing computing devices and involving a variety of accelerator architectures, we use a single model approach building predictions for "CPU-only" or "GPU-only", in other words, we focus on matching the entire task to the appropriate device.
- our intention is to extend the feature-space and the

training data of our predictor corresponding to the increasing heterogeneity when considering multiple computing devices. However, the necessity for a third classification choice "mix " (as proposed in [7]) should be investigated only after achieving the intended extensions, because then such an investigation would be meaningful for our model.

- particularly in regard to the energy consumption issue, involving multiple devices to perform a certain computation task could result in less energy efficiency in some cases. Fur this purpose, we plan to develop different predicting modes, which will be added to our model supporting different policies: *maximal performance* and *minimal energy consumption*. This might be infeasible if tasks are split among multiple accelerators.

4. APPROACH

In the context of this work, we introduce a method for predicting the suitability of a computation task to be run on a selective device in a heterogeneous computing environment. In the prediction procedure, a set of properties are extracted from OpenCL kernel programs at compile time. The extracted features need to carry enough information about the behavior of programs, in order to classify them concerning the execution hardware. It is possible to quantify these features as a vector of numeric parameters $\boldsymbol{x}^T = [x_1 x_2 ... x_n]$. Thus, the optimal matching of a task can be modeled as a function g($\boldsymbol{x}$) of this parameter vector. By means of a machine-learning technique that uses a statistical model, predictions are systematically created. In this way, our predictor can adapt to any new hardware using such techniques, and thus our method becomes platform-independent.

Well established benchmarks are useful to understand the computer architecture on the one hand, and to evaluate and compare different innovative methods on the other. For this reason, an important part of our research would be conducting a thorough evaluation of our predictor on the basis of several benchmarks and applications. As the Parboil benchmark suite includes a variety of heterogeneous applications emphasizing throughput-oriented computing, we deploy these benchmarks besides a set of real-world applications in the evaluation part. Involving this suite in the feature-extraction procedure enriches our evaluation during the machine learning phase greatly.

In order to explore and verify the system portability of our approach, we run our tests on different machines deploying a variety of very recent GPUs from the two major vendors in the GPU market: *AMD* and *NVIDIA*. A performance comparison between the different GPUs would be an interesting point in such a device-task mapping model, especially when also considering other modern computing architectures (such as *Intel* Xeon Phi). But, that will not be treated in the context of this work due to the abstract analysis intended in the context of code feature extraction.

The need for a comprehensive practice with modern accelerator architectures treated by such a predictor has been the initial spark of the contribution of another study [15], in which we disclose architectural characteristics of different processing devices. An extensive evaluation of performance is introduced while running a variety of applications using OpenCL. This investigation should help us to further extend the functionality of our feature-based predictor, and to selectively extend its feature space in future work.

Table 1: *List of extracted source code features. Both further columns contain the level of impact on the decision-making with regard to the target device. The impact values are determined as either low or high.*

feature		CPU	GPU
f_1:	Mem. accesses per transmitted Bytes	low	high
f_2:	FLOPs	–	–
f_3:	FLOPs per transmitted Byte	low	high
f_4:	IOPs	–	–
f_5:	IOPs per transmitted Byte	low	high
f_6:	Work-items	low	high
f_7:	Basic blocks	high	low

4.1 Feature-based Predictor

Estimation-based techniques for task-device matching in heterogeneous systems use either profiling-based or statistical methods. While code-profiling methods use past observations obtained from executed instances of a program, feature-based statistical methods make use of extracted properties to give a statement about the run-time behavior. With the former method, a lot of past observations are usually required in order to improve the accuracy of predictions, and even worse, high computation costs of profiling are to be expected. Furthermore, accurate predictions are only possible for regular computation patterns, which is not the case for many real-world applications. For these reasons, we use a feature-based statistical model to give an estimate of the task's suitability, and hence a decision about where to run a given task can be made autonomously. This model is able to treat regular as well as irregular computation patterns at low overhead costs.

When using such a model, as the number of the extracted features increases, the estimation accuracy of the predictor improves. But at the same time, the more features are considered, the more sample programs are required during the learning phase of the predictor. However, according to our approach, we currently extract the most decisive features (f_1 – f_7) of computation tasks as shown in table 1, with the intention of extending the feature space of our model in future work.

Each of the extracted features has a different level of impact on the matching decision regarding the target device (as shown in the CPU and GPU columns of table 1). Depending on run-time aspects in a heterogeneous system, the extracted features can be classified into two categories: f_1 – f_6 describe the complexity of the computation problem and the amount of involved mathematical operations, the last feature f_7 is a measure for the control flow complexity in the kernel program. In the assessment of feature-based prediction, it is not just a question of the individual features but also their combinations that are required in order to provide all useful information they contain. In this context, we use a statistical model due to its advantage that no prior knowledge about the deployed hardware is required. It can compensate for many different parameters, without requiring a distinct model for each of the device architectures. This achieves the portability of our model across systems as well as implementations.

4.2 LLVM Compiler Infrastructure

The feature extraction of kernel programs takes place at compile time. We propose a modified version of the LLVM compiler which accomplishes this task efficiently. The LLVM infrastructure has many advantages. Besides its simplicity, it has a language-agnostic design resulted in a variety of front ends for different programming languages. Most commercial and free OpenCL implementations are based on both components, LLVM [1] and Clang [2].

We have developed a custom LLVM pass that extracts a set of code features from a given kernel program. In order to extract the number of memory accesses and floating point as well as integer operations, an analysis is performed on the application's *Control Flow Graph* (CFG) that is generated by LLVM. This analysis iterates over all basic blocks in the given code. The recorded parameters are then multiplied by the frequency of the corresponding basic blocks in order to get the total amount of instructions that will be executed.

Fortunately, different types of LLVM's analysis passes are provided in the compiler infrastructure. One important analysis with regard to our objectives estimates the frequency of a basic block, that is how often a certain basic block in a program's work-flow will be executed. LLVM provides two different passes that can be used for this analysis: The first is the `BlockFrequencyInfo` pass which relies on the `BranchProbabilityInfo` pass. The latter deploys heuristics (and profiling information when available) to determine the probability that a given execution path (branch) of the program will be taken. The `BlockFrequencyInfo` pass uses this information to estimate the frequency of a basic block. The second pass that can be used to analyze the frequency of a basic block is `ScalarEvolution`. This pass calculates closed form expressions for all scalar integer variables in a program code [2], including induction variables. Therefore, it can be used to determine loop bounds (when it is possible).

Since `BranchProbabilityInfo` does not analyze the value range of an induction variable to determine the exact loop count even when it is possible, we decided to use the `ScalarEvolution` pass. In contrast to `BlockFrequencyInfo`, the other pass `ScalarEvolution` does not provide any other estimations for the block frequency. Thus, as an intermediate workaround, we consider each basic block to be executed exactly once if no further information is available.

Although our predictor in the current state, delivers highly accurate decisions without using `BlockFrequencyInfo` as will be seen later in section 6.2, it would be very interesting to observe the improvement in the predictor's accuracy when deploying the information from the `BlockFrequencyInfo` pass. In other words, we are going to combine both `BlockFrequencyInfo` and `ScalarEvolution` to obtain more accurate information about the basic block's frequency. However, this belongs to the intended feature-space extension.

4.3 Machine Learning

Our predictor uses a machine-learning technique to determine the optimal task-device matching of an OpenCL program. Each considered program is characterized as a fixed vector of features, which are extracted using a static analysis method and fed into the predictor. A feature-map function is built upon the training data (from the considered programs), which consists of a set of pairs <code features,predictions> obtained from the other programs. The learnt function describes a binary classification problem. In order to solve this kind of problem, statistical classification methods are required. In the following, we explain the mathematical background of these methods based on previous work [9], and discuss the implementation of these methods in our approach in details.

4.3.1 Reduction of Dimensionality

In our model, when the feature extraction process has terminated, the dimensionality of the feature space should be reduced in order to make the features specific for each program. Then, the resulting vectors can be passed to the predictor. The statistical procedure *Principal Component Analysis* (PCA) can be used to get a reduced set of orthogonal linear projections from a random vector. This random vector can be represented as follows: $\boldsymbol{X} = (X_1, \ldots, X_r)^T$, while $\boldsymbol{X}$ consists of possibly correlated random variables [3]. In other words, PCA performs a linear transformation of input data into a new coordinate system. The linear projections given by PCA are ordered by decreasing variance. The basic idea is, that the variance is an indication of the information content of a variable. Thus, the first principal component generally carries a major part of information while the following components carry less. Since the resulting random variables are orthogonal (see fig. 2), PCA decorrelates $\boldsymbol{X}$ during the construction. PCA is mainly used as a dimensionality-reduction technique, but, it can also be used to discover features of a data set (e. g. by plotting the data set given by PCA).

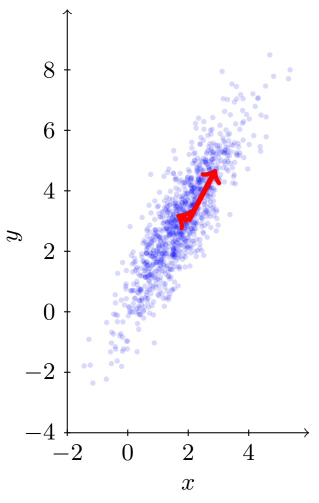


Figure 2: PCA of a multivariate Gaussian distribution, $\boldsymbol{\mu_X} = (2,3)^T$, $\boldsymbol{\Sigma_{XX}} = (1, 1.5; 1.5, 3)$. The vectors in the plot are the eigenvectors of the covariance matrix $\boldsymbol{\Sigma_{XX}}$, scaled by the square root of the corresponding eigenvalue.

Let $\boldsymbol{X} = (X_1, \ldots, X_r)^T$ a random vector with mean $\boldsymbol{\mu_X}$ and covariance matrix $\boldsymbol{\Sigma_{XX}}$.

PCA transforms the (possibly correlated) input variables X_i to a set of t uncorrelated linear projections $\xi_1, \ldots \xi_t$,

[1]http://www.llvm.org/
[2]http://clang.llvm.org/

where $t \leq r$:

$$\xi_j = \boldsymbol{b}_j^T \boldsymbol{X}$$

The total variance of the input variables can be computed as:

$$\sum_{j=1}^{r} \mathrm{var}(X_j) = \mathrm{tr}(\boldsymbol{\Sigma}_{\boldsymbol{XX}})$$

PCA only works well when the data lies in an approximately linear manifold $\mathcal{M}$, i. e. a linear subspace of input space $\mathbb{R}^r$. If it is likely that the data lies in a nonlinear manifold, further techniques such as *kernel PCA* are required. *kernel PCA* transforms the input data points $\boldsymbol{X}_i \in \mathbb{R}^r$ to points $\boldsymbol{\Phi}(\boldsymbol{X}_i) \in \mathcal{H}$. In this case, $\mathcal{H}$ is called *feature space*, the map $\boldsymbol{\Phi} : \mathbb{R}^r \to \mathcal{H}$ is called *feature map*. The dimensionality $N_{\mathcal{H}}$ of $\mathcal{H}$ is higher than the dimensionality r of the input space. Thus, after performing this transformation, a linear PCA can be carried out in the *feature space*.

We have already implemented PCA in our approach, however, in the current state of our feature-space, there is no need for it. In the following development of our model when the feature-space grows, we are going to reactivate PCA to get its performance advantages.

4.3.2 Binary Classification

There exist standard techniques for supervised learning models that allow binary classification of data. Due to its simplicity, *Support Vector Machine* (SVM) is one of the most used mathematical methods for solving such problems. In the case of data linearity, SVMs provide the optimal solution at high performance if compared to other mathematical algorithms like *Artificial Neural Network* (ANN) [17], which is better suited for modeling a non-linear space than SVMs. One example is the feature-based performance predictor [5] included in the related work.

An exemplary SVM model for a linearly separable data set is depicted in fig. 3. Given a set of training data, the idea behind such methods is to find a hyperplane in the vector space spanned by the features, so that the distance between the hyperplane and the closest training data point of each class becomes maximal. To give a quantified description of the SVM used in our model, let $\mathcal{L} = \{(\boldsymbol{x}_i, y_i) : i = 1, \ldots, n\}$ be a set of training data where $\boldsymbol{x}_i \in \mathbb{R}^r$ and $y_i \in \{-1, +1\}$. The classification of each program is taken within a binary space: Given $\mathcal{L}$, construct a function $g : \mathbb{R}^r \to \mathbb{R}$ so that

$$C(\boldsymbol{X}) = \mathrm{sign}(g(x))$$

is a classifier. That is, g partitions a test set $\mathcal{T}$ in two disjunctive classes, Π_+ and Π_-:

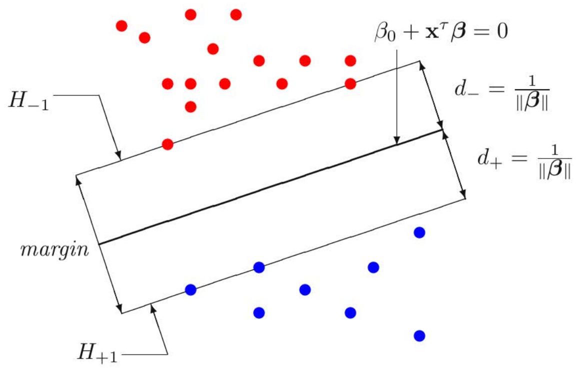
$$\Pi_+ = \{\boldsymbol{x} : C(\boldsymbol{x}) = +1\}, \quad \Pi_- = \{\boldsymbol{x} : C(\boldsymbol{x}) = -1\}$$

Consider the case where the data set $\mathcal{L}$ is linearly separable, i. e. by a hyperplane H:

$$H = \{\boldsymbol{x} : g(\boldsymbol{x}) = \beta_0 + \boldsymbol{x}^T \boldsymbol{\beta} = 0\}$$

$\boldsymbol{\beta}$ is called `weight vector` and β_0 is the `bias`.

The margin is defined as $d = d- + d+$ where $d-$ is the shortest distance from the hyperplane to the nearest negative data point and $d+$ is the shortest distance to the nearest positive data point (see fig. 3). The problem to find the



$\beta_0 + \mathbf{x}^{\tau}\boldsymbol{\beta} = 0$
H_{-1}
$d_- = \frac{1}{\|\boldsymbol{\beta}\|}$
$d_+ = \frac{1}{\|\boldsymbol{\beta}\|}$
margin
H_{+1}

Figure 3: A SVM exemplary model for a linearly separable data set $\mathcal{L}$. The red points belong to class Π_-, the blue points belong to class Π_+ (taken from [9]).

maximal margin classifier H can be formulated as:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|\boldsymbol{\beta}\|^2 \\ &\text{subject to} \quad y_i \cdot (\beta_0 + \boldsymbol{x}_i^T \boldsymbol{\beta}) \geq 1, \quad i = 1, \ldots, n \end{aligned}$$

This is a convex optimization problem that can be solved using Lagrangian multipliers. The dual problem can be written as follows:

$$\begin{aligned} \text{maximize} \quad & F_D(\boldsymbol{\alpha}) = \mathbf{1}_n^T \boldsymbol{\alpha} - \frac{1}{2}\boldsymbol{\alpha}^T \boldsymbol{H} \boldsymbol{\alpha} \\ \text{subject to} \quad & \boldsymbol{\alpha} \geq \mathbf{0} \\ & \boldsymbol{\alpha}^T \boldsymbol{y} = \mathbf{0} \end{aligned}$$

where $\boldsymbol{y} = (y_1, \ldots, y_n)^T$ and $\boldsymbol{H} = (H_{ij})$, $H_{ij} = y_i y_j (\boldsymbol{x}_i^T \boldsymbol{x}_j)$.

Let $S = \{i \in \mathbb{N} : y_i \cdot (\beta_0 + \boldsymbol{x}_i^T \boldsymbol{\beta}) = 1, \ 1 \leq i \leq n\}$. Then $\hat{\beta}_0$ and $\hat{\boldsymbol{\beta}}$ can be expressed as:

$$\hat{\boldsymbol{\beta}} = \sum_{i \in S} \hat{\alpha}_i y_i \boldsymbol{x}_i$$

$$\hat{\beta}_0 = \frac{1}{|S|} \sum_{i \in S} \left(\frac{1 - y_i \boldsymbol{x}_i^T \hat{\boldsymbol{\beta}}}{y_i} \right)$$

4.3.3 Training Data Separation

In general, the training data set $\mathcal{L}$ might be not linearly separable due to existing noise in the training data, so that the data can not be clearly classified, or that the data points are separable, but not in a linear fashion. In the first case, additional *slack variables* can be used to get a *soft-margin* solution. This allows that some data points can violate the constraint $y_i \cdot (\beta_0 + \boldsymbol{x}_i^T \boldsymbol{\beta}) \geq 1$, accounting for out-liners.

When the data is non-linearly separable, a technique similar to the idea used for kernel PCA can be employed. That is, a nonlinear transformation is applied to the training data points $\boldsymbol{x}_i \in \mathbb{R}^r$, so that a linear SVM can be constructed using samples $\{\boldsymbol{\Phi}(\boldsymbol{x}_i), y_i\}$.

In conclusion, a given OpenCL kernel can belong to one of two categories: “CPU-only” or “GPU-only”. The feature extraction of program code occurs at kernel’s compile time, however, the prediction itself has to be done at runtime of the application, because both the hardware platform and the program code are not defined a priori (due to the machine-learning approach). In the current state, the linear separation of feature data has achieved the objective of our predictor. However, methods that are better suited for modeling a non-linear space (such as ANN), might be valuated for further development in future work.

5. SUMMARY OF CONTRIBUTIONS

In contrast to the related work [7], we focus on matching entire computation tasks to the appropriate device in a heterogeneous system. Using a single model approach building predictions for "CPU-only" or "GPU-only", our predictor deploys the LLVM's IR (Intermediate Representation) for the purpose of code analysis. This has the advantage that we can extract code features of any kernel independently from any special front end. While the feature extraction of OpenCL kernel programs takes place at compile time, the prediction itself is performed at runtime.

The contributions of this paper can be described as follows:

- We provide a hardware-independent as well as compiler front-end independent predictor, which simplifies the process of adding new devices to the deployed computing machine. Thus, other accelerator architectures (i .e. *Intel Xeon Phi*) can profit the establishment of such a strategy.

- With the use of a feature-based matching model, highly accurate predictions are obtained for several applications and benchmarks (as will be seen in the evaluation), although no profiling is used by our predictor.

- From a programmer point of view, for applications on multiple processing the programmer should write a kernel once and it will be automatically mapped to the suitable computing device.

- These benefits may reduce power consumption in heterogeneous parallel systems and leverage the combined capabilities of multi-core CPU and many-core GPU architectures. In the case, if the wrong device (in terms of energy consumption) is repeatedly selected to perform a certain computation task in a heterogeneous system, this would result in less energy efficiency. A specially developed predicting mode "minimal energy consumption" would build the optimal decisions concerning the energy efficiency *performance per watt.* This predicting mode is currently in development as an alternative policy to the currently used one (maximal performance).

- While the focus of this paper is tending to avoid the waste of resources in a heterogeneous system by means of prediction, we believe, that these experiences might be advantageous for high performance computing systems' developers in the early stages of processor design.

- In future work, several measures will be taken to extend the feature space of our predictor, such as breaking the extracted FLOPs and IOPs down into different types of mathematical operations and combining the output data of both LLVM passes *BranchProbabilityInfo* and *ScalarEvolution.*

6. EXPERIMENTS

Experimental evaluations of our approach were performed using the Parboil benchmark suite as well as own implementations of several real-world applications written in OpenCL. These implement well-known algorithms in mathematics and physics. The experimental part is divided into two parts, each of them uses a different test platform and a different group of benchmarks and applications. In the first part, we consider applications, which primarily implement matrix algorithms.

In the second experiment, we extend the application range of the training data in order to demonstrate the improvement potential of the machine-learning technique used by our predictor, when several benchmarks with similar properties are taken into account. Further, the accomplishment of our tests on completely different hardware platforms is clear evidence of the predictor's adaptivity to a variety of accelerator-based computing machines.

6.1 Experiment – Part I.

The test set of applications selected for this experiment represent a wide range of algorithm behaviors and computation patterns. These include: Matrix-Matrix multiplication, Matrix-Vector multiplication, Laplace, Convolution, Electrical Field, MergeSort and Mandelbrot. For the purpose of extending the training data required in the machine-learning when considering transcendental functions, we also use a modified implementation of the Matrix-Matrix multiplication algorithm *Matrix-sqrt*, that additionally calculates the square root of each element in the resulting matrix deploying the *sqrt* function.

6.1.1 Platform I.

This experiment was carried out on a computing system with an *AMD's* FirePro S7000 GPU. This device is based on the *AMD* Graphic Core Next architecture [12] and consists of 20 compute units (64 processing elements for each) and 4 GB global memory. It delivers 2.4 TFLOPS of theoretical peak single precision floating point performance and 624 GFLOPS for double precision. It is connected through PCI Express 3.0 to a quad-core *Intel* CPU (i5-3550).

6.1.2 Evaluation

In this experiment, we demonstrate the ability of our predictor to classify given programs as "CPU-only" or "GPU-only". Since the execution time of an application is a function of the size and properties of input data, we performed a quantity of experimental tests on several applications by varying the workload. Prediction values are then represented each as an average of all those when executing the corresponding application with different input sizes. With the use of a standard cross-validation technique, the training data is partitioned into complementary subsets performing the statistical analysis on each subset and validating the analysis on the other subsets.

In order to solve the binary classification problem using SVM, the `weight vector` $\boldsymbol{\beta}$ and the `bias` β_0 in the equation above must be determined first. As explained previously, the extracted features taken from table 1 generate a vector $\boldsymbol{x}^T = [f_1, f_2, f_3, f_4, f_5, f_6, f_7]$. Generated by means of SVM, we measured the individual elements of the hyperplane H for the considered features. The `weight vector` ($\boldsymbol{\beta}$) as mea-

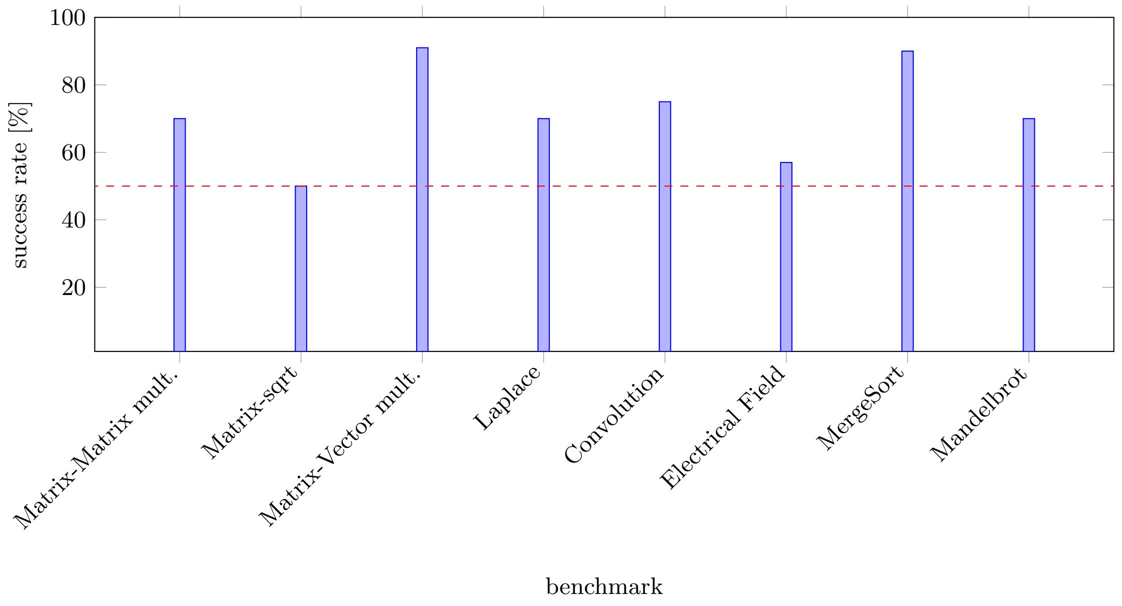


Figure 4: Predictor's results (presented in the form of success rates) of a set of applications. The test platform deploys an AMD's FirePro S7000 GPU connected through PCI Express 3.0 to a quad-core Intel CPU (i5-3550). The dotted line indicates the success rate of 50%.

sured using the experimental machine is:

$$\boldsymbol{\beta} = [0.2213, 0.0291, 0.2448, 0.0258, 0.1986, 0.0285, -0.0001]$$

and the measured *bias* is: β_0 = -0.7378.

The most expressive features having the highest weight values are marked with a different color.

In fig. 4, the predictions for several real-world applications are illustrated. The prediction of each application is expressed as a percentage between 0 - 100. In the most cases of our tests, a reasonably accurate prediction with a rate of 70% (error rate of 30%) or higher is achieved, except for Electrical Field[3] and Matrix-sqrt. Both algorithms involve a high amount of special transcendental functions, which are still not sufficiently supported by our predictor.

However, it is easily achievable to further increase the confidence of our machine-learning technique regarding this type of applications, by extending the feature space and the training data (considered algorithms) of our model. Especially in regard to the feature space, the memory access pattern in program code constitutes an indicative parameter when predicting matrix algorithms like Laplace and Matrix-Matrix multiplication. Similarly for the Laplace test, since this algorithm involves several calls of the same kernel, the slightly lower confidence (compared to the others) can be explained by the lack of training data. Considering more applications with similar computation pattern will surely lead to a further rise in its confidence. The extension of the training data and its impact on the predictions' accuracy will be explored in the second experiment section 6.2.

6.1.3 Validation of Results

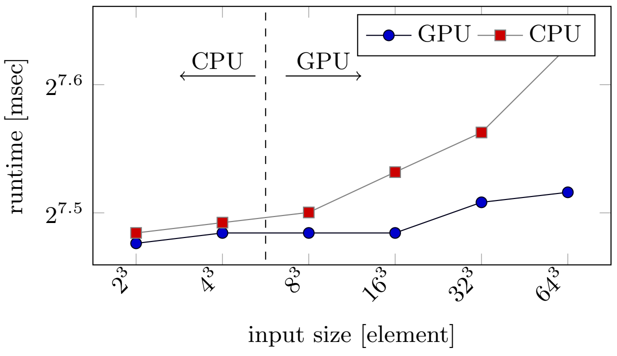
In order to establish the fidelity of our predictor and to know if the predicting matching is successful, we give a realistic evaluation of the predictions of selected applications. This is achieved through the observation of the actual execution times while using different input sizes. Each experiment was repeated 15 times and the average execution time was taken. During the validation, we focus on choosing input sizes that may have an impact on the matching procedure.

As shown in fig. 5, by increasing the workload (input sizes are shown on the x-axis), the device's suitability of both Matrix-Matrix multiplication and Electrical Field changes correspondingly. Since the overhead of transferring data to the GPU is not worthwhile for small workloads, the amount of computation per data item represents the main criteria in the matching decision. This is confirmed by the measured values of the weight vector, whereby the highest weight values correspond to features that focus on the computation's amount per transfer.

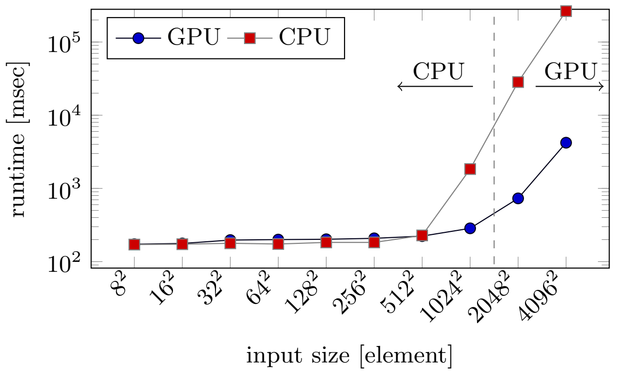
As depicted in fig. 5(a), our predictor classifies the Electrical Field up to input size $4 \times 4 \times 4$ as "CPU-only" and from than on as "GPU-only" (see the dashed line in the graphic). This estimate complies with the run-time measurements of both CPU and GPU, which change drastically from the input size $8 \times 8 \times 8$. When increasing the input size, the run time on the CPU increases continuously while that of the GPU changes only a little. The situation is similar for the Matrix-Matrix multiplication in fig. 5(b) with the only difference, that the run time of the CPU increases much more dramatically with the increased input size above 512×512. Our predictor gives its estimate for "GPU-only" one step later in the input scale (as shown on the dashed line). Since the execution time of Matrix-Matrix multiplication as many matrix algorithms depends on the size of the matrix, the computation amount describes a major factor in the decision regarding such applications. This factor is sufficiently represented in the feature space of our predictor.

To summarize, it may be surprising that even such a small feature space is sufficient to predict such a variety of applications, achieving a quite accurate predictor. Further, for

[3]http://physics.bu.edu/ duffy/py106/Electricfield.html



(a) Electrical Field



(b) Matrix-Matrix multiplication

Figure 5: The CPU and GPU run times of two applications under varying loads (log scale). The test platform deploys an AMD FirePro S7000 GPU connected through PCI Express 3.0 to a quad-core Intel CPU (i5-3550). The dotted lines indicate the threshold for the classification, which is defined by our predictor.

still not fully trained computation patterns we also see good opportunities in the long term development of our model, though highly accurate predictions can be made for such algorithms when extending both the feature space and the training data.

6.2 Experiment – Part II.

In this experiment, we extend our test benchmarks consisting of almost the same applications from the previous experiment by the Parboil benchmark suite and additional applications. Three real-world applications are included in the test set besides those from the first experiment. While the first application implements the *N-body* simulation [19] from astrophysics, the second one *Euclidean Distance* calculates the distances between two n-dimensional vectors, and the third one implements the *Jacobi* method used in the numerical linear algebra. Jacobi is an iterative algorithm for determining the solutions of a system of linear equations with largest absolute values in each row and column dominated by the diagonal element. During the first experiment, a special implementation of the Matrix-Matrix multiplication "Matrix-sqrt" was used for the purpose of extending the training data of the machine learning when considering transcendental functions. Since, we consider a wider range of applications in this experiment, "Matrix-sqrt" is not required anymore.

The *Parboil* benchmark suite includes several throughput computing applications, which are widely used to evaluate computer architectures, studying the performance of throughput computing devices and exploring new compilation and runtime techniques. A few of these benchmarks constitute common library routines with broad applicability, such as SGEMM or sparse-matrix, others represent specific analyses, such as the MRI sample gridding. Besides their simplicity, through its diversity and relevance the *Parboil* suite offers great opportunities for researchers to explore also accelerator-based computing systems.

6.2.1 Parboil Benchmarks

Since it fulfills the requirements of our research, we consider the baseline accelerated version of the *Parboil* suite based on [14]. The baseline suite version comprises decisions by which the algorithms are computationally parallelized and accelerated. Associated with the higher development costs, specially optimized versions of the benchmark suite also exist. However, in our exploration we focus on those versions, which are similar to what the average programmer would write. In the context of our investigation, the list of chosen benchmarks from the *Parboil* suite as illustrated in [14] includes:

- Sparse Matrix-Dense Vector Multiplication (SpMV): Sparse Matrix-Vector multiplication is the core of many iterative solvers. SpMV is memory-bandwidth bound when the matrix is large. Sparse matrix data can be stored or transformed into many data layout patterns (e .g. compressed sparse row (CSR) and Jagged Diagonal Storage (JDS)). Particularly, the JDS format is well-designed for parallel or vector processors, due to its better load balance characteristics. Thus, the chosen version in this benchmark is based on JDS format.

- Breadth-First Search (BFS): The Breadth-first Search algorithm is commonly used in graph problems such as finding the shortest path between two nodes. It is specifically optimized for a particular EDA application finding the shortest paths between a source node and every other node in a graph. Every node is taken in the current frontier and all unexplored neighbors are enqueued to the next frontier. This process iterates until all the nodes in the graph have been visited.

- Mri non-Cartesian Q matrix calculation (MRI-Q): This benchmark calculates in the GPU-based MRI reconstruction. An MRI image reconstruction is a conversion from sampled radio responses to magnetic field gradients. The Q matrix in an MRI image is a precomputable value based on the sampling trajectory, the plan of how points in k-space will be sampled. The algorithm examines a large set of input representing the intended MRI scanning trajectory and the points that will be sampled. MRI-Q is a fundamentally compute-bound application, as trigonometric functions are expensive and the regularity of the problem allows for easy management of bandwidth.

- Two-Point Angular Correlation Function (TPACF): TPACF is a measure of the distribution of massive bodies in space. The benchmark builds a histogram of

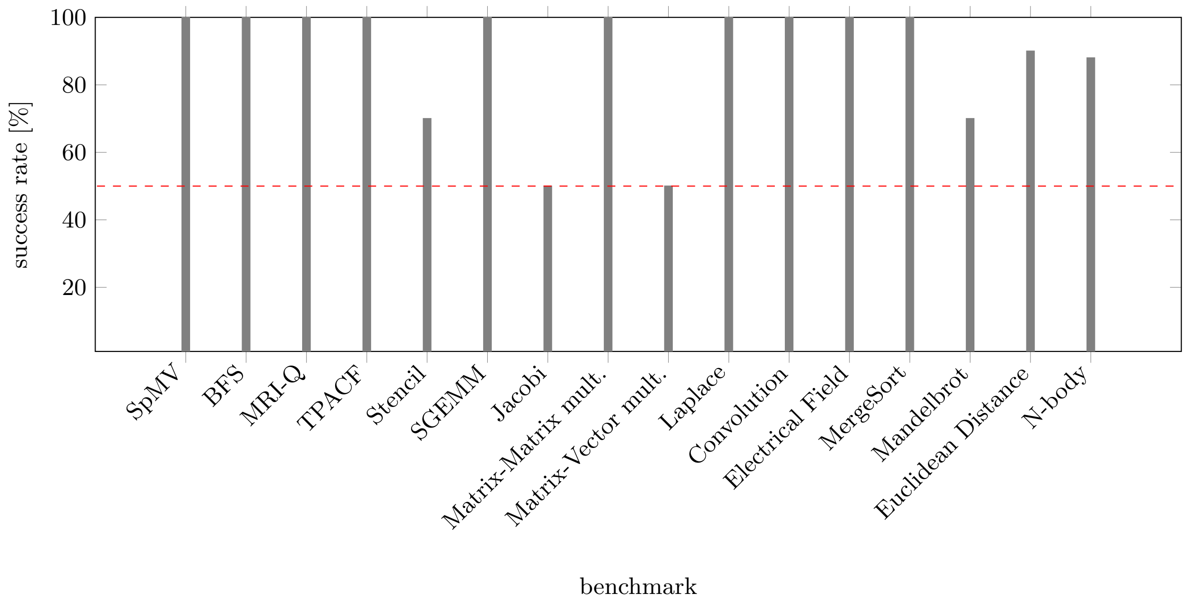


Figure 6: Predictor's results (presented in the form of success rates) of Parboil benchmarks and several applications. The test platform deploys an NVIDIA's *Tesla K20Xm* GPU connected through PCI Express 3.0 **to an eight-cores Intel CPU (Xeon E5-2650). The dotted line indicates the success rate of** 50%.

angular distances between all pairs of observed objects in space.

- Stencil: Due to the numerically and computationally-intensive nature of this class of applications, partial differential equations (PDE) solvers became an interesting candidate for accelerators. In this benchmark, the stencil code implements an iterative Jacobi solver of the heat equation on a $3D$ structured grid, which can also be used as a building block for more advanced multi-grid PDE solvers.
- SGEMM: The SGEMM dense matrix operation is an important building block in numerical linear algebra codes. Due to its comprehensibility, it is often the first studied and the most heavily tuned application on any new architecture.

6.2.2 *Platform II.*

The tests are run using exactly the same structure and methods as in the previous experiment. But in contrast to that, another hardware platform is used deploying the modern *NVIDIA's Tesla K20Xm* GPU. The GPU is connected through PCI Express 3.0 to an *Intel* CPU (Xeon E5-2650) with eight cores. *Tesla K20Xm* is based on the Kepler architecture [1], and consists of 14 compute units (192 processing elements for each) and 6 GB global memory. It delivers 3.52 Teraflops of theoretical peak single precision floating point performance and 1.7 Teraflops for double precision.

6.2.3 *Evaluation*

In order to further increase the confidence of our machine-learning technique regarding the different computation patterns of existing applications on the one side, and to verify the adaptivity of our approach to various computing machines on the other, we duplicate the first experiment from section 6.1 by extending the training data and deploying a completely different hardware platform in the context of this experiment. We demonstrate the potential improvement of our predictor while taking into account further real-world applications, these are: Jacobi, Euclidean Distance and N-body. Furthermore, the Parboil benchmark suite is involved in this experiment. By varying the input size, our tests carried out several hundreds in total. Each of the tests was repeated 20 times in order to increase the reliability of the results, whose averages are shown in fig. 6.

As shown with Laplace during the first experiment, the slightly lower confidence of our predictor when compared to the other applications can be explained by the iterative nature of this algorithm. Hence, the lack of training data for such computation patterns resulted in especially imprecise predictions. For this reason, we extend the application range during this experiment considering more applications with a similar computation pattern.

In this experiment, we show that the decisions' accuracy of our predictor improves reasonably when extending the application range. Running a variety of benchmarks from the Parboil suite next to the real-world applications rises the confidence of our predictor (see fig. 6). Predicting the considered matrix algorithms (i.e Laplace and Matrix-Matrix multiplication) has also improved drastically in terms of accuracy by the extension of the training data. Similarly, the machine-learning technique shows highly accurate results for most of the other considered real-world applications. Further, these tests show that our predictor is adaptive to a variety of hardware platforms. By using completely different experimental environments and GPU architectures, and repeating exactly the same tests on two different set of applications and benchmarks without any required change in its structure, the portability of our approach has been proven.

In contrast to the previous work presented by Grewe and F.P. O'Boyle [7], using a relatively smaller set of features was sufficient to obtain similarly high accurate predictions in whole. A realistic comparison with their results would be very interesting in the context of this paper, however, such a comparison is unfortunately not possible, because those results were obtained by means of a two-step approach. For such a comparison, only the results of their first-level predictor would be required.

7. CONCLUSION AND FUTURE WORK

The focus of this work lies on distributing several tasks among different devices in a heterogeneous computing system. We presented a feature-based static predictor that takes over the decision making in regard to where to run a certain task. During the evaluation, we have shown that our matching algorithm delivers highly accurate predictions for a variety of real-world applications and well established benchmarks. Future work will focus on further improving the accuracy of our predictor concerning algorithms with special computation patterns. This might be achieved through the extension of the feature space used by the machine-learning technique, for instance, breaking the extracted `FLOPs` and `IOPs` down into different types of mathematical operations.

We have performed a variety of tests on the *Intel Xeon Phi* accelerator, however, we let the presentation of these results for future work, when deploying multiple CPUs, GPUs and *Xeon Phis* in a single computing system. The simultaneous support of multiple devices will be especially attractive for many researchers and programmers, due to the increasing heterogeneity in modern systems. We believe that building a base for executing programs on heterogeneous devices autonomously could create an evolutionary path for the deployment of accelerators in the field of high performance computing research.

8. REFERENCES

[1] NVIDIAs Next Generation CUDA Compute Architecture: Kepler GK110, 2012. `www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper`.

[2] O. Bachmann, P. S. Wang, and E. V. Zima. Chains of recurrences–a method to expedite the evaluation of closed-form functions. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation*, ISSAC '94, pages 242–249. ACM, 1994.

[3] C. M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, 2005.

[4] G. F. Diamos and S. Yalamanchili. Harmony: an execution model and runtime for heterogeneous many core systems. In *Proceedings of the 17th International Symposium on High Performance Distributed Computing*, volume 4 of *HPDC '08*, pages 197–200, New York, NY, USA, 2008. ACM.

[5] C. Dubach, J. Cavazos, B. Franke, G. Fursin, M. F. O'Boyle, and O. Temam. Fast compiler optimization evaluation using code-feature based performance prediction. In *Proceedings of the 4th International Conference on Computing Frontiers*, volume 12 of *CF '07*, pages 131–142, New York, NY, USA, 2007. ACM.

[6] F. Feinbube, P. Tröger, and A. Polze. Joint Forces: From Multithreaded Programming to GPU Computing. *IEEE Software*, 28(1):51–57, 2011.

[7] D. Grewe and M. F. O'Boyle. A static task partitioning approach for heterogeneous systems using OpenCL. In *Proceedings of the 20th International Conference on Compiler Construction: Part of the Joint European Conferences on Theory and Practice of Software*, number 20 in CC'11/ETAPS'11, pages 286–305, Berlin, Heidelberg, 2011. Springer-Verlag.

[8] M. A. Iverson, F. Özgüner, and G. J. Follen. Run-time statistical estimation of task execution times for heterogeneous distributed computing. In *High Performance Distributed Computing, 1996., Proceedings of 5th IEEE International Symposium on*, pages 263–270, August 1996.

[9] A. J. Izenman. *Modern Multivariate Statistical Techniques*, chapter Support Vector Machines. Springer, 2008.

[10] J.-K. Kim, S. Shivle, H. J. Siegel, A. A. Maciejewski, T. D. Braun, M. Schneider, S. Tideman, R. Chitta, R. B. Dilmaghani, R. Joshi, A. Kaul, A. Sharma, S. Sripada, P. Vangari, and S. S. Yellampalli. Dynamic Mapping in a Heterogeneous Environment with Tasks Having Priorities and Multiple Deadlines. In *Parallel and Distributed Processing Symposium, 2003. Proceedings. International*, IPDPS.2003, pages 22–26, April 2003.

[11] C. Lattner. The LLVM Compiler Infrastructure. open source License, Jan 2014. `http://www.llvm.org/`.

[12] M. Mantor and M. Houston. AMD Graphics Core Next. `developer.amd.com/wordpress/media/2013/06/2620_final.pdf`.

[13] A. Munshi, B. R. Gaster, T. G. Mattson, J. Fung, and D. Ginsburg. *OpenCL Programming Guide.* Addison-Wesley Pearson Education, 2011.

[14] J. A. Stratton, C. Rodrigues, I.-J. Sung, N. Obeid, L.-W. Chang, N. Anssari, G. D. Liu, and W. mei W. Hwu. Parboil: A revised benchmark suite for scientific and commercial throughput computing. Technical report, IMPACT, March 2012.

[15] A. Tarakji and N. O. Salscheider. Runtime Behavior Comparison of Modern Accelerators and Coprocessors. In *2014 IEEE 28th International Parallel & Distributed Processing Symposium Workshops*, HCW 2014. IEEE Computer Society Press, 2014. (to appear).

[16] A. Tarakji, N. O. Salscheider, and D. Hebbeker. OS Support for Load Scheduling on Accelerator-based Heterogeneous Systems. *Procedia Computer Science, Proceedings of the 2014 International Conference on Computational Science*, 2014. (to appear).

[17] H. White. Learning in Artificial Neural Networks: A Statistical Perspective. *Neural Computation*, (1):425–464, 1989.

[18] V. Yarmolenko, J. Duato, D. K. Panda, and P. Sadayappan. Characterization and Enhancement of Dynamic Mapping Heuristics for Heterogeneous Systems. In *Parallel Processing, 2000. Proceedings. 2000 International Workshops on*, pages 437–444, 2000.

[19] S. P. Zwart, R. Belleman, and P. Geldof. High Performance Direct Gravitational N-body Simulations on Graphics Processing Unit I: An implementation in Cg. Technical report, Cornell University Library, 2007.