A REVIEW OF SCHEDULING MECHANISMS FOR HETEROGENEOUS MULTI-CORE MACHINES

M. Pradeep¹, J. R. Maria Navin², Balaji Kannappan³
¹Dept. of EEE, ²Dept. of ISE
³Dept. of CSE
¹,²,³Sri Venkateshwara College of Engineering, Bangalore, Karnataka, INDIA.
kannappan.balaji@gmail.com

October 7, 2018

Abstract

In recent trends, the heterogeneous computing devices become one of the most efficient platforms for executing real-time applications. The heterogeneous devices perform computation based on CPUs and GPUs. Now a day, OpenCL is one of the engineering standards program for heterogeneous devices. The conventional methods are used allocating the jobs to CPUs and GPUs to get optimal performance in terms of load balance, execution time, and throughput. The allocation of multiple applications to a heterogeneous system will leads to inconsistency. In this review, we present an overview of the most in ential algorithms reported in the field of heterogeneous multicores, and highlighting their limitations or drawbacks. Moreover, we present taxonomy of these algorithms as well as a qualitative evaluation of these algorithms, about a set of features needed since a prac- tical
1 Introduction

In practice, a single processor does not perform high performance computing on real-time applications. In recent years, single processor is being modified to multiple processors such as from smart phones (Samsung Galaxy S9) to super computers [1]. The multiple processors consist of CPU and GPU as well. GPU has high performance computing rather than CPU [2]. OpenCL (Open Compute Language) has developed as an important implementation for GPU. The OpenCL applications can be executed on different types of accelerators including GPUs, CPUs etc. [3]. The OpenCL program is divided into two parts, namely, host program and kernel functions. Host program is executed on a CPU device, whereas kernel function is executed on either CPU or GPU. The application developer [4] manages buffer creation, kernel mapping to the devices and data transfer in an OpenCL program. The main objective of multiple processor machines is to employ a scheduling support for efficient utilization of computing resources and to reduce the time complexity of applications [5].

The key objective of this research can be organized as follows:
1. To provide an existing literature of various scheduling mechanisms
2. To categorize the significances of this domain and
3. To track the recent developments of research in this domain

The rest of this paper is organized as follows. The Scheduling Policies are described in Section 2, Evaluation Metrics in Section 3, and Scheduling Mechanisms in Section 4. Finally, Section 5 concludes the paper.
2 Scheduling Policies

For evaluation, there are five scheduling techniques:
1. All jobs are assigned to a CPU device for execution
2. All jobs are scheduled on a GPU device.
3. Lowest numbers of jobs are assigned to CPU, whereas highest numbers of jobs are assigned to GPU.
4. CPU starts execution of the job queue from top of the queue, whereas GPU starts execution from end of the queue.
5. The scheduling is based on Machine Learning based classification technique.

3 Evaluation Metrics

1. Execution time refers to the time consumed in the execution of all jobs of the job pool.
2. Throughput represents the number of jobs completed per unit time.
3. Average time of a job is defined as an average amount of time taken by a job to complete its execution.
4. CPU starts execution of the job queue from top of the queue, whereas GPU starts execution from end of the queue.
5. Load balance is a distribution of workload among CPU and GPU.

4 Scheduling Mechanisms

A Qilin [6] system has been presented for programming heterogeneous devices which divided the kernel into two parts and map them to CPU and GPU devices. The performance of these devices is stored on a database for further use. Afterwards, the Qilin system makes scheduling decision with respect to the information stored in the database. In [7], there three different types of approaches are considered for scheduling decisions. The algorithm makes scheduling decision based on First Free, First Come First Serve, and Execution History. But this method is complex in case of predicting the waiting time for a task on a particular proces-
sor. In [8], the method considers execution history as well as data transfer information for scheduling legacy kernels to heterogeneous machines. Then, the runtime system makes a function calls on kernel and schedules those tasks either to a CPU or a GPU. In [9], the scheduling mechanism considers historical runtime data, state of the system, and specification of the system for allocating jobs to the multi-core devices. In [10], OpenCL features are extracted from the operations of int, oat, barrier, work items, etc during compilation time. To make scheduling decisions, whether to map a kernel to CPU or GPU, the above features are passed to support vector machine. StarPU [11] is a simple method for scheduling tasks to heterogeneous machines. It is a runtime system which provides a unified approach for executing kernels on multi-core devices. Afterwards, a greedy algorithm is used for assigning the job to a processor as soon as StarPU becomes idle. In StarPu, a priority preference is followed for executing the task having high priority. In [12], historical runtime data is used for scheduling the devices. If the appropriate device is busy in computing, then slower device is allocated for computation. In [13], a scheduling mechanism is represented by (1) generalized loop reduction and (2) structured grid computation. The given application is divided into number of chunks. The chunks are further divided into chunk-lets and assigned to a CPU or a GPU. Using First Come First Serve basis, single chunk-let are allocated to a CPU and multiple chunk-lets are assigned to a GPU. The algorithm ensures better load balancing between devices. A scheduling method proposed in [14] is an online profiling method which has data dependencies and time complexity of OpenCL application. Afterwards, a greedy algorithm has been employed for making scheduling decisions. HDSS [15] scheduling mechanism improves the performance of a kernel by equally balancing the load between CPU and GPUs. Initially, it learns the computing time of each processor in a smaller number of iterations. Afterwards, it makes optimal usage of resources depend on their execution speed. Finally, it makes sure for providing a load-balanced execution for multi-core machines. In [15], scheduling decision is based on OpenCL data parallel tasks to a CPU or a GPU. In this method, a kernel is divided into several sub tasks. Afterwards, each processor execution time is recorded. The scheduling decision takes place in future depends on the stored performance of the proces-
The algorithm presented in [16], divides a task into chunks and scheduling those task to either a CPU or a GPU. Initially, the scheduler starts with allocating smaller chunks to each processor. The chunk size varies depend on previous execution. In such a way, high performance computing devices are loaded with larger chunks, whereas slower devices are loaded with smaller chunks. In [17], scheduling decision for a CPU or a GPU is made with the help of estimated execution time of an application. The execution time is recorded during the training period. When a new application arrives for execution, then the recorded history is verified to allocate the device which is capable of high performance computing. The total execution time of an application is estimated as application completion time. In [18], the algorithm dynamically allocates the task to a CPU or a GPU. Machine learning technique such as Artificial Neural Network model is used to predict the model for assigning the task. The algorithm depends on OpenCL built-in functions and data transfer buffer size. Afterwards, Principal Component Analysis is used for optimizing the task partition. In [19], Single Kernel Multiple Data scheduling mechanism is employed for partition the task between available devices. The n-dimensional (ND) range is calculated and each subset is allocated to each device for performing computation. The time complexity and data transfer rate are computed and recorded to perform load balancing across multiple devices. In [20], a scheduling mechanism called CAP is presented for multi-core machines. Initially, small portion of tasks are equally distributed to a CPU and a GPU. The time complexity of both the device is analysed. Now, the workload is increased twofold on the high performance computing device. This process continues until there is a variance between present and past executions becomes smaller than a predefined threshold value. In [21], a scheduling mechanism called FluidiCL is implemented which handles executing and distributing of OpenCL program to CPU and GPU devices. The system does not need any prior based training. It automatically handles aggregation of results and transfers the data without any help from the designer. The n-dimensional (ND) range is calculated and data is distributed over multi-core machines. If a kernel is executed on a GPU device, simultaneously its subtasks are executed on a CPU device. In [22], scheduling mechanism is di-
vided into two different strategies which assign the work to CPU and GPU. Initially, naive profiling phase is useful for assigning the small task to both the devices and analysing their performance. After that, depending upon their recorded performance, the job is allocated to the specific device. In [23], the scheduling algorithm is uses different programs for scheduling multiple kernels. The system reduces average turnaround time and improves the throughput. The algorithm considers number of instructions, load/store operations and run time features for recording the performance of devices. Due to high performance computing, large number of tasks is allocated to GPU, and smaller numbers of tasks are assigned to CPU. In [24], a Multi Kernel on Multi Devices (MKMD) scheduling algorithm is used for distributing multiple kernels of an application. Initially, kernel is assigned to GPU device which minimizes data transfer rate and execution time. After that, kernel is divided into a number of sub tasks which are rescheduled to CPU. This algorithm creates a regression model for mapping the devices. The scheduling mechanism presented in [10] is extended by [25], in which machine learning technique is used for scheduling decisions. The method works on the idea of branch divergence for scheduling the job to either CPU or GPU. The branch divergence becomes complex in case of executing the application. In [26], a scheduling mechanism based on machine learning technique is proposed which includes OpenCL code features such as blocks, math functions and instructions for determining appropriate device. Moreover, the features such as data size and branch ratio are also used to analyze the performance of the device. The comparison characteristics of various scheduling mechanism are described in Table 1.
5 CONCLUSION

In summary, most of the algorithms are either anxious with the single kernel based scheduling or inadequate to the scheduling of particular types of applications. Some of the algorithm requires either offline training or modification of code. To the best of knowledge, there is no perfect method which performs load balancing without any offline training or profiling.

References


[26] Wen, Yuan and O’Boyle, M.F., Merge or Separate? Multi-job
Scheduling for OpenCL Kernels on CPU/GPU Platforms, In:
Proceedings of the General Purpose GPUs. ACM, (2017),
22.31.