The Scheduling Based on Machine Learning for Heterogeneous CPU / GPU Systems

D.A. Shulga[1], A.A. Kapustin[2], A.A. Kozlov[3], A.A. Kozyrev[4], M.M. Rovnyagin[5]

National Research Nuclear University MEPhI (Moscow Engineering Physics Institute)
Moscow, Russian Federation

[1]uinos@yandex.ru, [2]aakapustin2@gmail.com, [3]andrewkozlov@icloud.com,
[4]skyth724@gmail.com, [5]m.rovnyagin.2015@ieee.org

***Abstract* — Efficient use all of the available computing devices is an important issue for heterogeneous computing systems. The ability to choose a CPU or GPU processor for a specific task has a positive impact on the performance of GPGPU-systems. It helps to reduce the total processing time and to achieve the uniform system utilization. In this paper, we propose a scheduler that selects the executing device after prior training, based on the size of the input data. The article also contains the plots and time characteristics that demonstrate improvement in overall execution time, depending on the input data. The program modules were developed in C++ using CUDA libraries.**

***Index Terms* — GPGPU, Graphics processing units, NVIDIA CUDA, Heterogeneous System, Scheduling, Simple vector machine.**

I. Introduction

Until recent times, the CPU was the main component of high-performance systems, however the situation has changed upon development of the GPU. The CPU is advantageous in sequential tasks with relatively small amounts of processed information, while the GPU should be used to run tasks with a large size of data and a high degree of the processing concurrency [1,2,3]. To combine advantages of these two architectures, a separate architecture of heterogeneous computing was created. A heterogeneous computing system is a set of coprocessors – hereinafter referred to as "computing units" (CUs) – combined under a common control system. In the present Paper, CUs shall mean CPUs and GPUs, while the main purpose of using the heterogeneous computations is to achieve the best possible performance and use all CUs for their intended purpose, and that in its turn depends on operation of the scheduler.

In such systems, the CPU is quite often used to run and control cores on the GPU. However, in this case the CPU may be left unutilized, while the GPU would be on the peak of its use. That may affect execution of other tasks running at the moment. Meanwhile, some of cores run better on the GPU, and others – on the CPU. The main characteristic of a core – its runtime – depends on many parameters: current usage of system components, size and nature of input data, speed of data copying, and many others.

Static methods of scheduling the computations use assignment of a particular core to CUs without taking into account the above parameters, and the preference is often given to the GPU as the most productive device. In the present Paper, static scheduling means assignment of all tasks either completely to the CPU or completely to the GPU. Generally, it is on the GPU that GPU-optimized cores run faster, however in some cases this task could be assigned for execution by an under-utilized CPU. As a result, the task may be completed before its turn for execution on the GPU comes. In such systems, the same code can be executed repeatedly on different sizes of input data, and it is absolutely clear that the CPU is more efficient if used with a small size of input data, and this fact is completely ignored in static scheduling, thus leading to an increase in the time of processing the task queue. The present Paper proposes an algorithm for dynamic scheduling of computations, based on the size of input data, by using machine learning.

II. Related Work

It has been proposed to schedule computations by using different algorithms for predicting the runtime. In Paper [4], the choice of CUs between the CPU and the GPU took place according to a predicted runtime by using the history of test runs, while it was possible to generate a code for a particular CU. Paper [5] implements permanent usage of all CUs by assigning a task to the first free device, since it is aimed at balancing the usage of CUs. In Article [6], the scheduling is also carried out by using the method of runtime prediction, which is based on a function of parameters of input data and CUs. In Article [7], in order to predict the runtime, the least square method was used, where the size of input data and the time from the history of previous runs were used as parameters, and the current usage of queues to CUs was taken into account. Paper [8] considers various approaches to scheduling the computations between the CPU and GPU. One of them is based on selection of a free CU; another one uses time parameters of previous runs to calculate a coefficient, depending on which a task is given for execution by the CPU or GPU, with taking into account the usage of CUs. Final scheduling scheme takes into account the usage of CUs and calculates approximate time of executing a task, based on parameters from previous runs

III. Proposed Scheduling Scheme

The proposed scheduling scheme of computations is shown in Figure 1.

978-1-5090-0445-4/16/$31.00 ©2016 IEEE

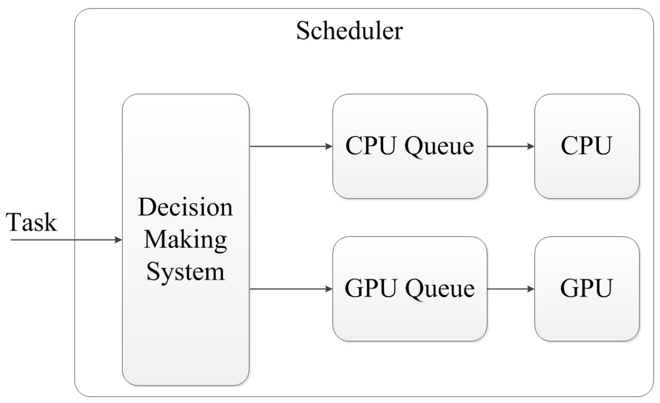

Fig. 1. Proposed scheduling scheme

Queue usage coefficient can be calculated by using the formula:

$$\rho = \lambda \times \overline{T}_w , \tag{1}$$

where λ means task delivery intensity, and $\overline{T}_w$ means average time of task maintenance [9]. It shall be ensured that queue usage coefficient is large enough, and yet, tasks should not accumulate in a queue, and then maximum performance of the scheduler may be achieved. In the present Paper, two queues are used, one of which keeps tasks for CPU, and another one – for GPU.

A new task comes to the scheduler, where static or dynamic scheduling is carried out. Dynamic scheduling of computations is performed by using algorithms of machine learning. As an algorithm for the system responsible for selection of CUs for execution, support vector method (SVM) [10] is used. Since the efficiency of SVM depends on core selection, its parameters, and soft margin value (C), a core in the form of a radial basis function [11] was selected as the most appropriate for the task of classification between the CPU and GPU. C value is selected during the learning, in order to achieve the highest possible accuracy. To implement support vector method in our case, it is necessary to solve the task:

$$\min_{\alpha} \frac{1}{2}\alpha^T Q\alpha - e^T \alpha , \tag{2}$$

where $e = [1,...,1]^T$ and $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, provided that for all $i = 1,...,l$, conditions are met as follows:

$$\begin{aligned} y^T \alpha &= 0 \\ 0 \le \alpha_i &\le C \end{aligned} . \tag{3}$$

Radial basis function looks as follows:

$$K(x_i, x_j) \equiv \exp(-\gamma \left\| x_i - x_j \right\|^2) . \tag{4}$$

It is worth noting that an incoming task should include implementations for the CPU and GPU to be processed correctly in the scheduler. Due to this, the scheduler can also be used both in normal systems with one type of CUs and in heterogeneous ones. A core for GPU should be written by using the CUDA technology [12]. If no task with this core has been executed before, it is necessary to configure the scheduler to a boundary value of input data size, for which the core should change CUs for execution. To this end, it is proposed to make several test runs of this task on all CUs with a particular set of input data defined by the user. The scheduler gathers information about input data size from each task and measures the time of copying the data from a host to a device and vice versa, and execution time of a task core. Simultaneously, at the time of starting the execution of a computing core for a task, current usage of all CUs is determined. Execution statistics depends not only on the size of input data, but also on current configuration and status of the system. Therefore, to fine tune to the right boundary, the learning process should be carried out for each individual system. Thus, the history of runs is stored for each task core. This statistics of task runs is shown to the user after each execution of a core during the learning, and it makes a decision about running on the CPU or GPU, based on this data. Once the scheduler is trained on this core, it is possible to choose CUs for execution of a task in the automatic mode. For execution of a task, a computing core is extracted from it for a particular CU and sent for execution in a separate system thread. Upon completion of this thread, the next queued task is called for execution, and thus the processing of the whole queue is carried out.

The scheduler was written in C++ as a stand-alone application by using the Qt framework. As a proposed implementation of SVM, the LIBSVM library is used [13] written in C++. Queues are implemented by using an asynchronous mechanism of signals and slots of the Qt framework [14], and its standard containers [15] are used for storing and processing the tasks.

IV. Experimental Results

For carrying out the experiment, a Histogram core from CUDA Samples [16] was used, because it has a fairly pronounced boundary, upon which it is necessary to switch CUs from CPU to GPU in order to achieve efficient usage. The configuration of test environment is shown in Table 1 below.

TABLE I. TESTING SYSTEM

Module	Model	Specifications
Central Processing Unit	Intel Core i3-2350M	2.30 GHz
Graphics Processing Unit	GeForce 630M	Compute capability 2.0
Random Memory Access	4 Gb	DDR3

For a start, static scheduling was used for the CPU, and then – for the GPU. The core ran on 20 sets of input data, starting from 125 Kb and finishing with 2.5 Mb at an increment of 125 Kb. Each set was filled with pseudo-random numbers. The algorithm of conducting a series of experiments, which has been used while carrying out the experiment, can be set up as follows:

1) *Preliminary Measurement Series:* A series of preliminary measurements is made in an amount greater than

100, and then the average of values is determined by using the formula:

$$\overline{x}_V = \frac{1}{n}\sum_{i=1}^{n} x_i\,, \tag{5}$$

where n means the number of measurements, and then standard deviation is calculated by using the formula:

$$s_V = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \overline{x}_V)^2}\,. \tag{6}$$

2) *Confidence Probability:* The probability of confidence is set, and the value of t parameter is determined by using the formula:

$$P\left\{\overline{x}_V - \delta < x < \overline{x}_V + \delta\right\} \approx 2\Phi(t) = p\,. \tag{7}$$

3) *Measurement Amount Determination:* The accuracy is set, and necessary number of experiments is determined by using the formula:

$$n \geq \left(\frac{ts_V}{\delta}\right)^2. \tag{8}$$

The resulting dependence of the core runtime on the size of input data is represented in Figure 2, where CPU Time shows results of running the CPU of a core, and GPU Time – GPU of a core respectively. The runtime includes the time of copying the data and runtime of a computing core. The time parameters for CPU version of a core were measured by capturing timestamps using the means of the operating system. From the inside of the GPU, the time was measured by using the event mechanism from CUDA API. As seen from the diagram, this core with input data of up to 1 Mb in size is more efficient if running on the CPU than on the GPU, and with data of a larger size – clearly more efficient on the GPU. This is caused by significant time of copying the data into the GPU memory from the CPU and back, while the runtime of a computing core of the GPU version is a lot less than the time of the CPU version.

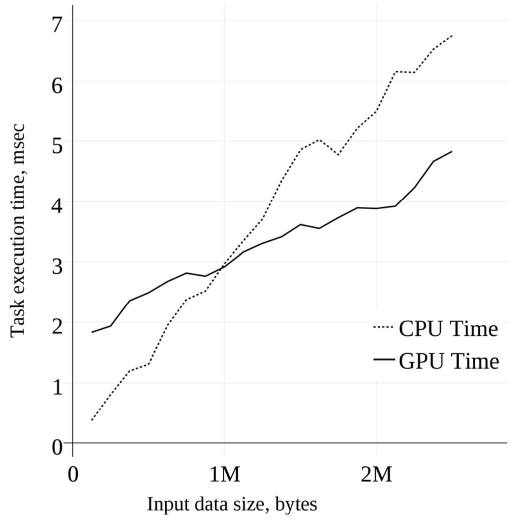

Fig. 2. Dependence of core runtime on input data size for static scheduling

For efficient operation of the scheduler in the dynamic scheduling mode, it is necessary to configure it by training the solver and specifying that for data less than 1 Mb in size. All tasks with this core should be sent for execution to the CPU, and if this boundary is exceeded – to the GPU. The results of operation of the scheduler in the dynamic mode after learning on a test set are shown in Figure 3.

You can see that when using dynamic planning with a trained system the total time of processing the queues is reduced to the largest time of processing one of the queues. In this experiment, the GPU queue completed its work later than the CPU queue, therefore the total time of executing all sets is equal to the time of executing the tasks on the GPU. Figure 4 shows the time of executing a task consisting of 20 consecutive executions of computations on the above datasets with a Histogram core in static scheduling on the CPU, GPU, and dynamic scheduling with the trained scheduler. One can see that the use of dynamic scheduling showed an improvement by 35% compared to the execution of all tasks on the CPU.

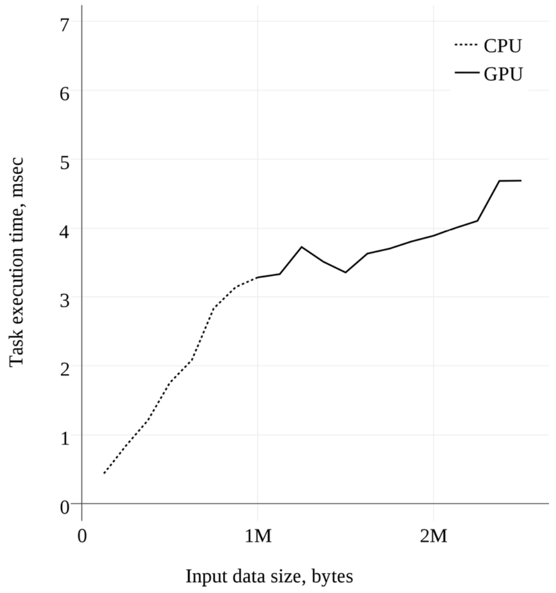

Fig. 3. Dependence of core runtime on input data size for dynamic scheduling

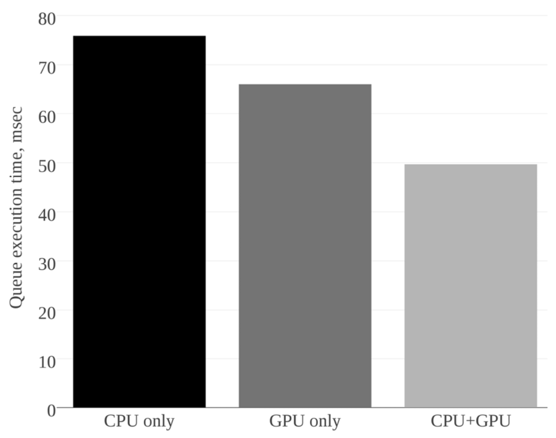

Fig. 4. Dependence of queue execution time on scheduling type

V. Conclusions

The present Paper considers the method of scheduling the computations in heterogeneous computing systems by using a machine-learning algorithm. The proposed algorithm is based on the support vector method, which allows carrying out classification of tasks between CPU and GPU efficiently after learning on a small amount of runs of computing cores.

This method allows a system administrator to configure the scheduler with high accuracy to the boundary, starting from which the tasks should be executed on one or another CU, and that leads to efficient use of system resources. The support vector method is well scalable for several input parameters, and that is planned to be used in future research. It is planned to consider more parameters in classification, such as the number of multiprocessors in CUs, the speed of copying the data, the ability to copy data in parallel with execution of a computing core, and the size of memory on a device. The experimental results of applying this algorithm show a significant reduction in the time of processing a set of tasks, and a high accuracy of classification, while the use of own queues of execution leads to achievement of efficient concurrency in executing the tasks.

Acknowledgment

The authors would like to express their gratitude to the Head of Computer Systems and Technologies Department of the National Research Nuclear University “MEPhI”, Professor M. A. Ivanov for his assistance and support during the research, as well as to the Head of Inter-Facility Hybrid Computing Systems Laboratory, Assistant Professor N. P. Vasilyev for the provided equipment and consultations.

References

[1] Kato, S.; Lakshmanan, K.; Kumar, A.; Kelkar, M.; Ishikawa, Y.; Rajkumar, R., "RGEM: A Responsive GPGPU Execution Model for Runtime Engines," in *Real-Time Systems Symposium (RTSS), 2011 IEEE 32nd* , vol., no., pp.57-66, Nov. 29 2011-Dec. 2 2011

[2] Hybrid reconfigurable computing system layout and high-accuracy calculations based on it. [Online]. Available: http://2013.nscf.ru/TesisAll/Plenar/06_1140_LacisAO_P06.pdf

[3] Dyumin, A.A.; Kuznetsov, A.A.; Rovnyagin, M.M., "Evaluation of statistical properties of a modified Bloom filter for heterogeneous GPGPU-systems," in *Young Researchers in Electrical and Electronic Engineering Conference (EIConRusNW), 2015 IEEE NW Russia* , vol., no., pp.71-74, 2-4 Feb. 2015

[4] J.-F. Dollinger and V. Loechner, “CPU+GPU Load Balance Guided by Execution Time Prediction” in IMPACT 2015, 2015.

[5] S. Zhang, C. Zhu, J. K. O. Sin, and P. K. T. Mok, “A novel ultrathin elevated channel low-temperature poly-Si TFT,” *IEEE Electron Device Lett.*, vol. 20, pp. 569–571, Nov. 1999.

[6] M. A. Iverson, F. O□zgu□ner, and L. C. Potter, "Statistical prediction of task execution times through analytic benchmarking for scheduling in a heterogeneous environment," IEEE Transactions on Computers, vol. 48, no. 12, pp. 1374- 1379, 1999.

[7] C. Gregg, M. Boyer, K. Hazelwood, and K. Skadron, "Dynamic Heterogeneous Scheduling Decisions Using Historical Runtime Data" in ISCAW 2011 (A4MMC), 2011.

[8] H. J. Choi, D. O. Son, S. G. Kang, J. M. Kim, H.-H. Lee and C. H. Kim "An efficient scheduling scheme using estimated execution time for heterogeneous computing systems", *J. Supercomput.*, vol. 65, no. 2, pp.886 -902 2013

[9] M. Zukerman, "An introduction to queueing theory and stochastic teletraffic models," Information available online at http://www.ee.cityu.edu.hk/~zukerman/classnotes.pdf, 2015.

[10] C. Cortes and V. Vapnik "Support-vector networks", *Mach. Learn.*, vol. 20, pp.273 -297 1995

[11] Y.-W. Chang, C.-J. Hsieh, K.-W. Chang, M. Ringgaard and C.-J. Lin "Training and testing low-degree polynomial data mappings via linear SVM", *J. Mach. Learn. Res.*, vol. 11, pp.1471 -1490 2011

[12] NVIDIA CUDA C Programming Guide 7.5 [Online]. Available: https://docs.nvidia.com/cuda

[13] C. Chang and C. Lin "LIBSVM: a library for support vector machines", *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 2, no. 3, pp.27 2011

[14] The Qt Framework, http://www.qt.io/qt-framework/, April, 2015

[15] Mark Summerfield “*Advanced QT Programming: Creating Great Software with C++ and QT4”*, 2010 :prentice Hall, printed

[16] Victor Podlozhnyuk, “Histogram calculation in CUDA”, NVIDIA CUDA Sample Documentation, 2007.