Improving Scheduling Techniques in Heterogeneous Systems with Dynamic, On-Line Optimisations

Marcin Bogdański
School of Computer Science
University of Birmingham
Birmingham, UK
Email: mxb039@cs.bham.ac.uk

Peter R. Lewis
School of Computer Science
University of Birmingham
Birmingham, UK
Email: p.r.lewis@cs.bham.ac.uk

Tobias Becker
Department of Computing
Imperial College London
London, UK
Email: tobias.becker04@imperial.ac.uk

Xin Yao
School of Computer Science
University of Birmingham
Birmingham, UK
Email: x.yao@cs.bham.ac.uk

***Abstract*—Computational performance increasingly depends on parallelism, and many systems rely on heterogeneous resources such as GPUs and FPGAs to accelerate computationally intensive applications. However, implementations for such heterogeneous systems are often hand-crafted and optimised to one computation scenario, and it can be challenging to maintain high performance when application parameters change. In this paper, we demonstrate that machine learning can help to dynamically choose parameters for task scheduling and load-balancing based on changing characteristics of the incoming workload. We use a financial option pricing application as a case study. We propose a simulation of processing financial tasks on a heterogeneous system with GPUs and FPGAs, and show how dynamic, on-line optimisations could improve such a system. We compare on-line and batch processing algorithms, and we also consider cases with no dynamic optimisations.**

***Keywords*-Scheduling, Heterogeneous System, Genetic Algorithm, Artificial Neural Network, On-Line Learning, Dynamic Optimisation, FPGA, GPU**

I. Introduction

Graphics Processing Units (GPUs) and Field-Programmable Gate Arrays (FPGAs) have become popular for solving large-scale complex problems from science, engineering, finance, etc. These devices can effectively be used as accelerators in high-performance computing applications which exhibit large amounts of parallelism. One example may be financial option pricing [Tse et al., 2010b], [Tse et al., 2010a], which we use as a case study for introducing dynamic scheduling optimisations based on an on-line learning approach.

Despite the rising popularity of GPUs and FPGAs, task scheduling and resource allocation remain still challenging problems in such heterogeneous, dynamic systems, where evaluating real performance of a particular CPU/GPU/FPGA is very difficult and will vary depending on the types of task scheduled.

Such systems are commonly investigated using traditional computational models and resolution methods, but scheduling policies based on static performance evaluations can be inefficient when computation parameters vary dynamically. The other problem one has to consider is that there is a significant *initialisation overhead* when sending tasks to GPU/FPGA nodes, which is not accounted for in most scheduling algorithms. On the other hand, if one can evaluate performance of the underlying resources and the *initialisation overhead* for a specific architecture and computational problem, traditional scheduling methods will be able to return near-optimal solutions.

Using machine learning in scheduling has been considered in previous research. [Aytug et al., 1994] and [Priore et al., 2001] summarise different approaches including the use of neural networks. In more recent work [Mahajan et al., 2008] investigates use of a machine learning for branch prediction in processor scheduling and [Vladusic et al., 2009] and [Zhao et al.] apply machine learning strictly to grid scheduling problem. However, these do not consider the case when performance of the underlying resources is not known a-priori, which is the focus of this paper.

The aim of this paper is to present a supervised on-line learning approach for evaluating the performance of the underlying hardware architecture for a specific computational problem. In our model, the performance of the underlying resources is measured in application throughput τ which is not known a-priori. τ is the amount of data processed in a certain time window. It can also change with time depending on how a specific hardware architecture of the targeted computational resource (GPU/FPGA) and the deployed computational kernel is suitable for the currently computed tasks.

The research leading to these results has received funding from the European Union Seventh Framework Programme under grant agreement n° 257906.

Figure 1. Pilot vs non-pilot scheduling. Top: estimation of τ accurate; Middle: estimation not accurate; Bottom: running small batch as pilot program.

To deal with this problem, we introduce a Learning Module (LM), which monitors the system in discrete time steps. The LM can periodically schedule additional system checks to re-evaluate performance in GPU/FPGA nodes for the currently computed problem. These additional checks introduce a time overhead, but they can deliver a better estimations of τ in these nodes, which in turn can have positive effects on the scheduling algorithm. The task of the LM is to learn when it is beneficial to schedule an additional system check, and when to rely on partial information. We call these checks *pilot programs*.

The main research questions we pose here are:

1. Can an online-learning approach improve overall performance of a scheduler, if it is operating in uncertain conditions, where the exact hardware performance for a specific task is volatile and not known a-priori

2. How does such an online-learning approach compare with manually tuned periodical system checks and the case when no additional checks are made

3. How do the approaches that are specified in 2. compare when the underlying system is static in nature, or when some system parameters change slowly

We evaluate the proposed model under the heterogeneity of an CPU/GPU/FPGA system using the extended version of the HyperSim-G simulator [Xhafa et al., 2007b]. We integrated the main HyperSim-G framework with our online-learning module, where an artificial neural network (ANN) is used to learn and decide on additional system checks. We test our model in multiple cases, including:

a) a scheduler based on traditional genetic algorithm and Struggle Genetic Algorithm [Xhafa et al., 2008a], [Xhafa et al., 2007], [Xhafa et al., 2008]

b) small/large batches of tasks

c) the use of online learning versus hand-crafted solutions

d) an environment that is static, slowly changing, or quickly changing.

The rest of the paper is organized as follows: In Section II we define the simulation environment, the scheduler and introduce the learning algorithm. In Section III we explain the details of learning algorithm, we define an error feedback function and draw a schematic of the ANN. In Section IV we present our method and the results obtained in a dynamic, uncertain environment. We end the paper in Section V with conclusions and plans for future work.

II. Preliminaries

In this work we consider the independent job scheduling problem [Xhafa et al., 2010], [Kołodziej et al., 2009], [Kołodziej et al., 2010]. In the independent job scheduling problem a number of tasks n is processed in the batch mode. The scheduling module requires the number of machines available and their estimated processing throughput τ as an input. It also requires the workload for individual tasks. With this information the scheduling module is able to generate a matrix with the expected time to compute [Ali et al., 2000], which is used to create a near-optimal schedule for this particular problem.

To adapt the independent job scheduling model to our heterogeneous hardware architecture, we propose three changes to this model. Firstly, we assume that the computational parameters of the workload stay uniform over certain periods of time. This is true for some financial applications like financial option pricing methods that are optimised to some real time market parameters. Hence, the estimated τ of a machine will be only valid for a limited period of time.

Secondly, we assume that the data throughput τ for the available machines is not known. The GPU and FPGA may be loaded with different kernels optimised for different types of tasks, and often it is not known how well a kernel will perform for a new type of task. τ is also specific to a particular application implementation and usually not known a-priori. In this situation τ must be estimated based on previous experiences.

Thirdly, contrary to a standard CPU, accessing GPU/FPGA resources must follow strict procedures. In a traditional cluster, information about the state of the system, currently executing tasks, timing of task execution etc. is readily available. In such a case, estimating τ based on the execution time of an individual tasks would be trivial. In the case of a GPU/FPGA it is more difficult to measure execution times for individual tasks. As mentioned in [Tse et al., 2010a], distributing tasks to the GPU/FPGA one at a time results in large amount of message passing overhead and latency. Because of this it is not practical to continuously dispatch smaller workloads to the FPGA in order to measure task performance. Instead, it is more efficient to load a larger workload that corresponds to a batch of tasks. τ is then estimated based on the execution time for this task batch. To model the overhead for loading new tasks to a resource, we introduce an *initialisation overhead*, which is a time penalty added every time a task or a batch of tasks is distributed to the GPU or FPGA.

At any point the scheduler may decide to split out a small number of tasks from the batch and run a *pilot program*. The purpose of pilot program is to determine τ for the current batch of tasks before scheduling the whole batch. Scheduling a pilot program yields an additional *initialisation overhead* penalty, but because of better τ estimation, the scheduler should be able to distribute tasks more efficiently. Note that the scheduler has access to the timing of the previous batch, and it is able to update its estimations of τ with or without running a pilot program. However without a pilot program its estimations may be less accurate. Pilot vs non-pilot execution is visualized in Fig. 1. As we can see in

Figure 2. Scheduling multiple batches of tasks. During second batch type of tasks change. a) With good τ' estimation scheduler works effectively. b) change happen in the system, new type of tasks run slow on machine nb 2, real completion time is longer than estimation. c) pilot program is run to accurately re-evaluate τ'.

figure 1, a pilot program may improva task allocation, but the overall effect may be worse than without a pilot program. This is the main question we pose in this paper: In which situations it is beneficial to run a pilot program?

The main problem that we are considering, is if the LM can learn from the stream of multiple batches of tasks in which situations it is worth running an additional pilot program. We want to balance the overhead of the pilot program with the potential speed-ups that can be achieved by using learning. For example if the system computes only one type of tasks, then running many pilot programs will have a negative impact on performance. On the other hand, if multiple types of tasks are loaded over time, periodical pilot programs will be necessary to avoid miscalculations of schedules. This is ilustrated in Fig. 2

A. Input stream definition

We define an input stream of n task batches $IS = [tb_1, \ldots, tb_n]$, where each task batch consists of m tasks $tb = [u_1, \ldots, u_m]$. Each task batch arrives at time t_n. After task batch arrival the scheduler is initialized to distribute tasks between machines. Each batch is also one learning example. The batch size is generated by a Gaussian distribution which is limited to the $-3\sigma^2$ to $3\sigma^2$ range in order to avoid negative batch sizes. We experimented with other distributions as well, but we found that this did not have a significant impact on the results of our experiments.

$$P(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \tag{1}$$

We assume that the next batch is pooled from an external source as soon as the current one finishes. Time span between two scheduling events is defined as $\Delta t_{n-1} = t_n - t_{n-1}$.

The system consists of k computational nodes, and each node is characterised be τ_k parameter, which is the computational performance of node k. Because τ_k may change as a result of tasks changing over time, we define $\tau_k = f_k(t)$, the value of τ_k is updated continuously during the simulation, not only at time when new batch of tasks arrives. Note, that the task scheduler is not aware of the real value of τ_k, it operates based on an estimation τ'_k. We also define a *load overhead* lo, which is global and constant during one simulation.

In particular we use sinusoidal (Eq. 3) and step (Eq. 2,4) functions to model the dynamic nature of τ_k.It is important to note that this is just simple, preliminary model. It would be interesting to see how thel LM behaves when more complex functions are used.

$$f_k(t) = \sum_{i=0}^{n} \alpha_i \chi_{A_i}(t) \tag{2}$$

$$f_k(t) = A_k \sin(\omega_k t + \varphi_k) \tag{3}$$

where:

$$\chi_{A_i}(t) = \begin{cases} 1 \text{ if } t \in A \\ 0 \text{ if } t \notin A \end{cases} \tag{4}$$

Where A_i is function interval.

B. HyperSim-G

To simulate the scheduling we use the HyperSim-G framework [Xhafa et al., 2007b]. The HyperSim-G simulator is based on a discrete event model; the behaviour of the system is simulated by discrete events occurring in the system. The sequence of events and the changes in the state of the system capture the dynamic behaviour of the system. The simulator provides a full simulation trace by simply indicating a parameter for trace generation. This functionality is useful for an easy implementation of the Learning Module. The main concept of how the HyperSim-G simulator and the Learning Module are connected to each other is shown in Fig. 4. In a comparison with the HyperSim-G framework there is a clear division between real and estimated resources, which must be additionally generated to define a scheduling event. The Learning Module is designed for better estimation of τ and in this way it supports the resolution methods used in the scheduler class of the simulator. The output of the scheduler is a sub-optimal schedule.

C. Scheduling problem definition

In this work, each learning step we consider is an sndependent job scheduling problem, in which tasks are processed in the batch mode [Xhafa et al., 2010]. A total number of m tasks are scheduled in each learning step n.

A *schedule* of the batch of tasks at the Grid site is defined as a vector $x = [x_1, \ldots, x_n]^T$, in which $x_j \in [1, k]$ indicates the number of the machine, to which task j is assigned ($j = 1, \ldots, m$) and k is number of machines.

The problem formulation in this approach is based on the expected time to compute matrix model [Ali et al., 2000], in which an instance is defined by:

(a) the computational loads of the tasks;

(b) the computing capacities of machines;

(c) the estimation of the prior load of each available machine and

(d) the *ETC* matrix, which elements define estimations of the time needed for task completion on machines in the system.

In our problem task characteristics change from time to time, and we take account of that by updating τ. Because we handle changing task characteristics in τ, the workload of each task is constant and does not change. By workload we mean the amount of work any single task requires to compute. It does not make sense to update workload, as different machines may react differently to changes in tasks characteristics. In our case we assume the workload is constant and τ changes.

III. Learning Module

Our scheduling problem is defined on per-batch level, and we now concentrate on the question: If it is beneficial to execute a *pilot program* for a specific batch or not? The LM is executed for each batch-scheduling event tb_n, and its task is to make the binary decision, if the pilot program should be run (and initialization overhead penalty should be taken), or not. To make this decision the LM has to evaluate how good the current estimation of τ'_{nk} is. As an input, we define two variables: the time that passed since last task batch $\Delta t_{n-1} = tb_n - tb_{n-1}$, and the error that was made in last task batch. See Fig. 2.

$$\tau'_{nk} = \frac{\sum workload}{completion_{nk}} \tag{5}$$

The error in learning step n for machine k is defined as the difference between estimation and actual execution time.

$$error_{nk} = abs(\frac{completion_{nk} - \Delta t}{\Delta t}) \tag{6}$$

If the error in learning step $n-1$ is high and spanned over a long period of time Δt, then some change happened in the system and it would be beneficial to run the pilot program. If the error is small, and the time period since scheduling last batch is short, then the change was probably insignificant and the pilot program would probably take too much initialization overhead to be beneficial. The task of LM is to learn most efficient decision boundary.

But how do we know if LM decision was right or wrong? How do we know if we should reinforce positively or negatively? We cannot evaluate the LM right at the moment it makes decision. At learning step n we can only evaluate the past decision at learning step $n-1$. To do this we simulate *what if a different decision was made* situation. Knowing estimated $\tau'_{k,n-1}$, updated $\tau'_{k,n}$ and the estimated completion time $completion_{k,n-1}$ we can simulate the scenario of executing task batch tb_{n-1} where pilot program was or was not run. Of course we only need to simulate scenario that did not happen. Depending on if the simulation yields better performance or not, at learning step n we can decide if the LM made a good decision at learning step $n-1$. See Fig. 3

Figure 3. Comparison of real and simulated runs a) Simulating *what if pilot program were run* b) Simulating what if pilot program were not run

Figure 5. Comparison of three scheduling algorithms, while resource changes according to $sine$ function. Vertical axis shows makespan, lower is better; a) initialization overhead = 200, k = 8 b) initialization overhead = 400, k = 8, c) initialization overhead = 200, k = 16, d) initialization overhead = 400, k = 16

IV. Experimental Analysis

In this section we present an experimental evaluation of the Learning Module using our extended version of HyperSim-G. In particular, we are considering two cases: in the first case the dynamic nature of τ_k is defined as sinusoidal, see Eq. 3. In second case we define τ_k as a step function, see Eq. 2. We compare performance of three approaches: without the pilot program (DISABLED), with the learning module (LEARN) and with pilot program executed in each iteration (ALWAYS ON). We consider DISABLED and ALWAYS-ON to be *hand-crafted* solutions. By this we mean it is a pre-programmed solution by the user. Note that the user may be right or wrong in choosing which solution to use, which may result in well- or badly-crafted solution to particular problem. On the other hand the LEARN approach adapts to underlying system automatically. The computation in this experiment is representative of an option pricing application with slowly changing market parameters.

In addition we compare clusters of 8 and 16 computational nodes. It may seem to be a small number of nodes, but usage of GPU/FPGA accelerators are meant to decrease cluster size. We also consider values of *initialization overhead* of 200ms and 400ms. All results are averaged over 30 runs.

Comparing ALWAYS-ON, LEARNING and DISABLED we see that the LM approach achieved the best result in 3 out of 8 cases. It is more important to note that in none of the test cases it was the slowest. It is true that in 5 cases one of hand-crafted solution was faster, but it is not easy to predict in advance which solution will be better for particular test case.

If we look closer at relative performance of different methods, we can see that in the worst case (fig. 5.4) the LM is about 20% slower than hand-crafted solution. On the other hand figure 6.3 shows that a badly-crafted solution can be twice as slow as the optimal solution (in this case LM).

Furthermore, we can compare plots 1-2 and 3-4 on figure 6. As we see, changing the initialisation overhead has significant impact on ALWAYS-ON policy, but the LM does not suffer much in performance (it actually improves). Although the LM is not always the best, it can achieve solutions close to optimal.

V. Conclusions and Future Work

In this paper we proposed an online-learning approach to boost scheduler performance in cases when underlying com-

Figure 4. Structure of simulator and learning module, re-simulating *what if* situations not included

Figure 6. Comparison of three scheduling algorithms, τ_k is defined by $step$ function. Vertical axis shows makespan, lower is better; a) initialization overhead = 200, k = 8 b) initialization overhead = 400, k = 8, c) initialization overhead = 200, k = 16, d) initialization overhead = 400, k = 16

putational resources cannot be easily evaluated and changes dynamically. Experimental analysis show that learning can improve scheduler performance.

An important insight is that in all cases the LM approach performs better than badly hand-tuned solutions and in many cases better than what we considered good solutions before running the LM. In the situation where the dynamics of underlying system are not known, the LM approach may therefore be proposed as a suitable alternative to pre-programmed solutions, in cases when they are not feasible. We showed that the LM will adapt automatically to underlying system and yield near-optimal solution even though dynamics of the system are not known.

The other point to note is that in our research we only change τ as function of time. What if initialisation overhead and number of machines change with time as well? It is very common for computational nodes to be attached or detached from the cluster.

In current and future work we plan to introduce more input variables to the Learning Module to make it more aware of its environment, incoming tasks, hardware etc. We also intend to investigate how our approach can scale up to larger and more complex systems. We plan on extending LM, so it doesn't only make binary decision about a pilot program, it should gradually tune multiple parameters of the scheduler.

REFERENCES

[Ali et al., 2000] Ali, S., Siegel, H.J., Maheswaran, M., and Hensgen, D.: "Task execution time modeling for heterogeneous computing systems", *Proceedings of Heterogeneous Computing Workshop*, pp. 185–199, 2000.

[Kołodziej et al., 2009] Kołodziej, J., Xhafa, F., Kolanko, Ł.: "Hierarchic Genetic Scheduler of Independent Jobs in Computational Grid Environment", *Proc. of 23rd ECMS, Madrid, 9-12.06.2009, in J. Otamendi, A. Bargieła, J.L. Montes and L.M. Doncel Pedrera eds.*, IEEE Press, Dudweiler, Germany, 2009, pp. 108–115.

[Kołodziej et al., 2010] Kołodziej, J. and Xhafa, F.: "A Game-Theoretic and Hybrid Genetic meta-heuristic Model for Security-Assured Scheduling of Independent Jobs in Computational Grids", *Proc. of CISIS 2010*, IEEE Press, , USA, 2010, pp. 93–100.

[Michalewicz, 1992] Michalewicz, Z.: *"Genetic Algorithms + Data Structures = Evolution Programs"*, Springer, 1992.

[Xhafa et al., 2007] Xhafa, F., Barolli, L. and Durresi, A.: "Batch Mode Schedulers for Grid Systems", *International Journal of Web and Grid Services*, 3(1): 19–37, 2007.

[Xhafa et al., 2008] Xhafa, F., Barolli, L.,and Durresi, A.: "An Experimental Study On Genetic Algorithms for Resource Allocation On Grid Systems", *Journal of Interconnection Networks.* 8(4): 427–443, 2008.

[Xhafa et al., 2008a] Xhafa, F., Duran, B., Abraham, A., and Dahal, K. P.: "Tuning Struggle Strategy in Genetic Algorithms for Scheduling in Computational Grids", *Neural Network World*, 18(3): pp. 209–225, 2008.

[Xhafa et al., 2010] Xhafa, F., Abraham, A.: "Computational models and heuristic methods for Grid scheduling problems", *Future Generation Computer Systems*, 26 (2010), pp. 608–621.

[Xhafa et al., 2007b] F. Xhafa, J. Carretero, L. Barolli and A. Durresi: "Requirements for an Event-Based Simulation Package for Grid Systems", *Journal of Interconnection Networks* vol.8, No.2, 163178, 2007, World Scientific Pub.

[Tse et al., 2010a] Tse, A. H. T., Thomas, D. B., Tsoi, K., and Luk, W. (n.d.): "Dynamic Scheduling Monte-Carlo Framework for Multi-Accelerator Heterogeneous Clusters." *IEEE Symposium on Field-Programmable Technology (FPT)*

[Tse et al., 2010b] Anson H.T. Tse, David B. Thomas, K.H. Tsoi, Wayne Luk: "Efficient Reconfigurable Design for Pricing Asian Options" *in Proceedings of the International Workshop on Highly Efficient Accelerators and Reconfigurable Technologies (HEART)*

[Haohuan et al., 2009] Haohuan Fu, Oskar Mencer, Wayne Luk: "FPGA Designs with Optimized Logarithmic Arithmetic" *Submitted to IEEE Transactions on CAS-I*

[Aytug et al., 1994] Haldun Aytug, Siddhartha Bahattacharyya, Garry J. Koehler, Jane L. Snowdon: "A Review of Machine Learning in Scheduling" *IEEE Transactions on Engineering Management*, vol. 41 (1994), no. 2

[Priore et al., 2001] Paolo Priore, David Dela Fuente, Alberto Gomez, Javier Puente: "A Review of Machine Learning in Dynamic Scheduling of Flexible Manufacturing Systems" *AI EDAM*, 15 (2001), pp 251-263

[Mahajan et al., 2008] Anjali Mahajan, Ms Ali, Mamta Patil: "Instruction Scheduling using Evolutionary Programming" *Applied Computing Conference* 2008

[Vladusic et al., 2009] Dainel Vladusic, Ales Cernivec, Bostjan Slivnik: "Improving Job Schduling in GRID Environments with Use of Simple Machine Learning Methods" *Sixth International Conference on Information Technology: New Generations* (2009) pp.177-182

[Zhao et al.] Guopeng Zhao, Zhiqi Shen, Chunyan Miao: "ELM-Based Intelligent Resource Selection for Grid Scheduling" *Machine Learning and Applications* (2009) pp.398-403