Concurrency and Interference Analysis of Kernels on GPUs

Pablo Carvalho[1], Lucia Maria de A.Drummond[1](advisor), Cristiana Bentes[2](co-advisor)

[1]Instituto de Computação
Universidade Federal Fluminense (UFF), Rio de Janeiro, Brasil

[2]Faculdade de Engenharia
Universidade do Estado do Rio de Janeiro (UERJ), Rio de Janeiro, Brasil

pablocarvalho@id.uff.br,lucia@ic.uff.br,cris@eng.uerj.br

Abstract. *Heterogeneous systems employing CPUs and GPUs are becoming increasingly popular in large-scale data centers and cloud environments. In these platforms, sharing a GPU across different applications is an important feature to improve hardware utilization and system throughput. However, under scenarios where GPUs are competitively shared, some challenges arise. The decision on the simultaneous execution of different kernels is made by the hardware and depends on the kernels resource requirements. Besides that, it is very difficult to understand all the hardware variables involved in the simultaneous execution decisions, in order to describe a formal allocation method. In this work, we studied the impact that kernel resource requirements have in concurrent execution and used machine learning (ML) techniques to infer the interference caused by the concurrent execution, and to classify the slowdown that results from this interference. The ML techniques were analyzed over the GPU benchmark suites, Rodinia, Parboil and SHOC. Our results showed that, from the features selected in the analysis, the number of blocks per grid, number of threads per block, and number of registers are the resource consumption features that most affect the performance of the concurrent execution.*

1. Problem Characterization and Motivation

Graphics Processing Units (GPUs) have proven to be a powerful and efficient platform to accelerate a substantial class of compute-intensive applications. For this reason, many large-scale data centers are based on heterogeneous architecture comprising multicore CPUs and GPUs to meet the requirements of high performance and data throughput. GPUs are also being used in computational clouds. Exploring GPUs in clouds through GPU virtualization allows physical devices to be logically decoupled from a computational node and shared by any application, resulting in monetary cost reduction, energy savings and more flexibility.

In this scenario, efficiently sharing a GPU across different applications is an indispensable feature. Recent GPUs introduced the concept of concurrent kernel execution that enables different kernels to run simultaneously on the same GPU, sharing the GPU hardware resources. Concurrent kernel execution facilitates GPU virtualization and can improve hardware utilization and system throughput. The blocks of the concurrent kernels are dispatched to run on the streaming multiprocessor (SMs) and the warp scheduler

arranges the order at which each warp will execute, with near-zero context-switch overhead.

However, one key difficulty for concurrent kernel execution is that, in the GPU, the low-level sharing decisions are proprietary and strictly closed by GPU vendors. Consequently, GPU virtualization software has no control over the actual resource sharing. Our first work in this area, published in [Carvalho et al. 2020b], showed the impact that kernel resource requirements have in concurrent execution for the kernels of the most important GPU benchmarks. The results showed that resource-hungry kernels on one resource might prevent concurrent execution. This study, however, did not explain how the resource requirements of the kernels have an effect on the performance of the concurrent execution.

We performed an extensive analysis of the concurrent execution of the kernels from Rodinia [Che et al. 2009], Parboil [Stratton et al. 2012], and SHOC [Danalis et al. 2010] benchmarks to assess how their performance is affected when they run simultaneously [Carvalho et al. 2020a]. We use four machine learning techniques [Michalski et al. 2013] to induce models capable of inferring if there is interference in the concurrent execution and also to classify the slowdown resulting from such interference. Furthermore, we rely on feature selection [Guyon and Elisseeff 2003] and feature importance [Zien et al. 2009] techniques to understand how the resource usage of the kernels impacts the possible interference and slowdown. We focus on the co-execution of two kernels to better understand their interference avoiding the explosive increase in the number of experiments.

2. Objective and Main Contributions

Our first study [Carvalho et al. 2020b] exposed that the relation between kernel pairs may not be simple to explain. The kernel co-executing experiments were performed on our kernel submission framework but using a reduced set of kernels.

Our second study [Carvalho et al. 2020a] provided a more in-depth analysis of the effects of concurrent execution. We analyzed the co-execution of 60 kernels, creating 3600 execution scenarios to be analyzed. The questions we wanted to answer were: Can a pair of kernels run concurrently? How their resource requirements have influenced their co-execution interference? To answer these questions, we used four machine learning classifiers (XGBoost[Chen and Guestrin 2016], Multilayer Perceptron, Logistic Regression and K-Nearest Neighbor) to determine if there is interference in the concurrent execution and to classify the slowdown. The classifiers used as features the resource requirement information requested before the kernel execution: numbers of threads, blocks, shared memory usage and numbers of registers. We evaluated the machine learning models considering not only accuracy but also some other important statistics (e.g. precision, recall and kappa — this last one provides a measure on how far from the expected classification the results are). Furthermore, we analyzed the feature importance in order to understand the relation between the features in each problem.

This dissertation has the following contributions:

- An extensive experimental analysis of the concurrent execution of pairs of kernels from the most important GPU benchmark suites;

- A machine-learning study on the concurrent execution results with four different techniques to unveil how the kernels interfere with each other in the concurrent execution;
- A comparison of the machine techniques, k-Nearest Neighbours, Logistic Regression, Multilayer Perceptron and XGBoost in their ability to infer if there is interference in the concurrent execution, given the resource requirements.
- A feature importance analysis to reveal the features that matter the most for the performance interference of the concurrent execution.

3. Experimental Results

Due to the lack of space, we show here only the results of the machine learning study. The experiments were performed on two GPUs: a Tesla P100-SXM2, and a RTX 2080.

3.1. Machine Learning Results for Determining Concurrency

We induced the four classifiers for distinguishing whether or not a pair of kernels can execute concurrently, using four features from each kernel on the processed dataset. Tables 1 and 2 present the accuracy, precision, recall and kappa results for P100 and RTX-2080, respectively. We can observe that XGBoost achieves the best results on almost all metrics. This is expected, as ensemble methods are known to reach better results than learning the models individually, and XGBoost is the current *defacto* choice of ensemble methods for classification tasks. One can also see that kappa index is consistently better for XGBoost than to the rest of the classifiers in both cases.

The only exception is the value of recall when the kernels run on RTX-2080. In this particular case, the logistic regression performs surprisingly well, with almost no positive test examples incorrectly classified as negative. In fact, with P100, some classifiers have a very disappointing performance: logistic regression performs extremely badly for the positive examples and only achieves an accuracy of 0.6014 because it correctly classifies the negative examples. Similarly, MLP incorrectly classifies several positive examples as negative, yielding a very low value of recall. In this way, MLP would say that two kernels cannot run concurrently when they actually can, yielding a very cautious classifier. KNN is consistent on the precision and recall metrics, but it still worse than XGBoost. On the other hand, when the kernels run on RTX-2080, all the classifiers achieve quite good performance. KNN, for example, has precision as high as XGBoost and, as said before, the recall of Logistic regression is even higher than the one achieved by XGBoost. The RTX-2080 presented more pairs of kernels with a high probability of executing concurrently. Therefore, the classifiers solve the concurrency problem in RTX-2080 easier than on P100.

	Accuracy	**Precision**	**Recall**	**Kappa**
MLP	0.7295	0.7299	0.1366	0.1162
K-NN	0.7295	0.6836	0.5887	0.4202
LR	0.6014	0.2533	0.0035	-0.0029
XGB	**0.8164**	**0.8113**	**0.7001**	**0.6068**

Table 1. Predictive results for the concurrency problem on P100.

	Accuracy	**Precision**	**Recall**	**Kappa**
MLP	0.7642	0.7832	0.9532	0.1630
K-NN	0.7663	**0.8260**	0.8759	0.3215
LR	0.7574	0.7599	**0.9933**	0.0243
XGB	**0.8008**	**0.8238**	0.9375	**0.3657**

Table 2. Predictive results for the concurrency problem on RTX-2080.

3.2. Machine Learning Results for Determining the Interference (Concurrency Effect)

Knowing that 1,427 and 2,546 kernel pairs have a high probability (at least 80%) of concurrent execution, on P100 and RTX-2080, respectively, the next experiment consisted in inducing a classifier for the interference problem, computed from the concurrency effect (CE). Given two kernels K_i and K_j, the CE is defined as the ratio between the sum of their standalone sequential execution times (when they are executed one after another without concurrency) and the time they take to execute concurrently.

$$CE = \frac{T_{K_i} + T_{K_j}}{T_{(K_i,K_j)}} \tag{1}$$

To that, for each pair (K_i, K_j), if CE < 1, the interference is such that the concurrent execution causes a slowdown in the execution of the kernel when compared to their standalone executions. This case is called here as a *negative* effect. On the other hand, if CE > 1, this means that there is no interference, which we can call a *positive* effect. Inducing a classifier for CE is a different problem from deciding whether or not a kernel pair would run concurrently, as the variables that could impact the CE could be others.

The predictive results are disposed in Tables 3 and 4, for P100 and RTX-2080, respectively. Once again, XGBoost reached the best values for accuracy, precision, recall and kappa. In comparison to the prior experiment, the CE classifier achieves lower metrics results since the concurrency effect is a more difficult task to find a pattern. Similar to the previous experiment the Logistic Regression recall value on RTX-20280 is also as high as XGBoost, with a slight small difference between them. Still, its kappa index is quite low, making it not a good classifier to this problem, even though it is good at not mistake the positive examples as negative ones (it has a low value of false-negative examples.)

	Accuracy	**Precision**	**Recall**	**Kappa**
MLP	0.5781	0.5370	0.3858	0.1213
K**-NN**	0.6279	0.5833	0.5703	0.2448
LR	0.5726	0.5324	0.3149	0.0976
XGB	**0.6889**	**0.6723**	**0.5876**	**0.3623**

Table 3. Predictive results of the Interference Problem (Task 2) on P100.

	Accuracy	**Precision**	**Recall**	**Kappa**
MLP	0.5746	0.6187	0.6339	0.1355
K**-NN**	0.6426	0.6645	0.7055	0.2732
LR	0.5491	0.5645	0.7813	0.0495
XGB	**0.7133**	**0.7165**	**0.7927**	**0.4140**

Table 4. Predictive results of the Interference Problem (Task 2) on RTX-2080.

3.3. Feature Importance Analysis

Regarding the concurrency, the number of blocks per grid of both kernels and the number of registers of the second kernel are the most relevant features on both GPUs. For kernels (K_1, K_2) to run concurrently from the beginning, K_1 blocks must not occupy all the SMs entirely, so the number of blocks of K_1 must be smaller than the maximum number of blocks that the hardware can allow being active in the SMs, which means that K_1 is leaving space for K_2 execution. The number of registers of K_2 is important to determine if there is space in the register file for the K_2 variables. For P100, blocks per grid are around 20% more important than the number of registers. For RTX-2080, this difference is more pronounced. Blocks per grid have almost double the importance value than the

number of registers. This occurs because RTX-2080 has a smaller number of SMs, so the kernels blocks will have less space to be allocated, increasing the importance of this feature.

About the interference problem, the results are somewhat different for the two GPUs. On P100, the method elicits the blocks per grid and the number of registers as the most important features, with almost the same importance. On RTX-2080, the method also elicits the blocks per grid as the most important feature, around 30% more important than the other features, but the importance of the number of threads per block is increased when compared to the P100 results. The importance of the blocks per grid on the interference results reinforces the leftover policy of NVIDIA. When the kernels execute concurrently, the first kernel allocates the GPU resources and the second kernel can run with the leftover resources. On RTX-2080, nonetheless, the importance of the number of threads per block increases since RTX-2080 has the same number of registers as P100, but its maximum number of threads per SM is half of the P100 maximum. This means that some warps of the second kernel have to have their execution postponed when the maximum number of threads is reached in the SM.

4. Main Publications

1. Carvalho, P., Clua, E., Paes, A., Bentes, C., Lopes, B., and Drummond, L. M. (2020a). Using machine learning techniques to analyze the performance of concurrent kernel execution on gpus. Future Generation Computer Systems, 113(1):528–540. **(Qualis A2)**
2. Cruz, R. A., Bentes, C., Breder, B., Vasconcellos, E., Clua, E., de Carvalho, P. M., and Drummond, L. M. (2018). Maximizing the gpu resource usage by reordering concurrent kernels submission. Concurrency and Computation: Practice and Experience (online), 1(1):1–12. **(Qualis A2)**
3. Carvalho, P., Drummond, L. M., Bentes, C., Clua, E., Cataldo, E., and Marzulo, L. A.(2020). Kernel concurrency opportunities based on gpu benchmarks characterization.Cluster Computing, 23(1):177–188. **(Qualis B1)**
4. Carvalho, P., Drummond, L., Bentes, C., Clua, E., Cataldo, E., and Marzulo, L. (2017). Analysis and characterization of gpu benchmarks for kernel concurrency efficiency. High Performance Computing. CARLA 2017. Communications in Computer and Information Science, 796. **(Qualis B4)**

5. Conclusions

Modern GPU architectures support concurrent sharing of the GPU resources among multiple kernels, which can unleash the power of the GPU for dynamic and highly virtualized environments such as large-scale heterogeneous clusters and cloud environments. This work presented an extensive analysis of the concurrent execution of the kernels from Rodinia, Parboil, and SHOC benchmarks on two different GPU architectures to assess how their performance is affected by the concurrent execution. We used machine learning techniques to understand and predict the execution interference of the kernels and which types of kernel can execute concurrently.

Our focus was to identify tricky relations among the resource requirements of the kernels and their concurrent execution. We used four machine learning techniques to

capture the hidden patterns that make a kernel interfere in the execution of another one. Our results showed that XGBoost, a state-of-the-art ensemble method, achieved the best quantitative results with statistical significance validated. The feature importance method showed the resource requirements that are the most relevant in the concurrent execution performance. By analyzing the variables chosen as the most important to induce the XGBoost model, we conclude that the number of blocks per grid is the most relevant feature to define if the kernels will execute concurrently and to influence the performance interference. The second most important feature depends on the GPU architecture. For the GPU with more resources, P100, the number of registers is key for the kernels interference, while for the GPU with less SM resources but the same amount of registers, RTX-2080, the number of threads per block is more relevant in the kernels interference. The results obtained in this work can be further used in the design of a scheduling strategy for GPUs, where the resource requirements of the kernels could help the scheduler in making wise decisions for concurrent execution.

References

Carvalho, P., Clua, E., Paes, A., Bentes, C., Lopes, B., and Drummond, L. M. (2020a). Using machine learning techniques to analyze the performance of concurrent kernel execution on gpus. *Future Generation Computer Systems*, 113(1):528–540.

Carvalho, P., Drummond, L. M., Bentes, C., Clua, E., Cataldo, E., and Marzulo, L. A. (2020b). Kernel concurrency opportunities based on gpu benchmarks characterization. *Cluster Computing*, 23(1):177–188.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., , and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. *In Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*, page 44:54.

Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794. ACM.

Danalis, A., Marin, G., McCurdy, C., Meredith, J. S., Roth, P. C., Spafford, K., Tipparaju, V., and Vetter, J. S. (2010). The scalable heterogeneous computing (SHOC) benchmark suite. *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*, page 63:74.

Guyon, I. and Elisseeff, A. (2003). An introduction to variable and feature selection. *Journal of machine learning research*, 3(Mar):1157–1182.

Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. (2013). *Machine learning: An artificial intelligence approach*. Springer Science & Business Media.

Stratton, J. A., Rodrigues, C., Sung, I.-J., Obeid, N., Chang, L.-W., Anssari, N., Liu, G. D., and mei W. Hwu, W. (2012). Parboil: A revised benchmark suite for scientific and commercial throughput computing.

Zien, A., Krämer, N., Sonnenburg, S., and Rätsch, G. (2009). The feature importance ranking measure. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 694–709. Springer.