Predicting the Execution Time of a kernel on a specific GPU using PTX code

Monika Dagar
Courant Institute of Mathematical Sciences
New York University
New York, US
md4676@nyu.edu

Jorge Roldan
Courant Institute of Mathematical Sciences
New York University
New York, US
jlr9718@nyu.edu

***Abstract*—During the last couple of decades, there has been an exponential growth in the amount of time and energy required to run workloads on high-performance computing systems, which nowadays rely heavily upon GPUs. In order to reduce the resources required by these systems, one clear approach is to avoid inefficient applications by using prediction models that could inform developers of the approximate execution time. In this work, we have trained models based on ensemble learning techniques such as Random Forest and Gradient Boosted Decision trees, as well as the deep learning architecture TabNet to predict the execution time of a specific kernel on a specific GPU architecture. We used data obtained using the CUDA-Flux profiler from the PTX code as input features. The best performing model in terms of the number of predictions with an error in the range of (0-10%) is CatBoost with 91.6%, Random Forests with 85.4%, and TabNet with 76.6%.**

I. Introduction

The use of modern high-performance computer systems (HPC), and their workloads have drastically increased during the last couple of decades. The wide adoption of graphical processing units (GPUs) as an essential part of these systems has played a key role in the computational capabilities of these systems as well as the scope of applications that are now possible in the areas of artificial intelligence, big data analytics, scientific computing, video games, and many others.

As the complexity and size of these workloads increase, the time and energy required to complete them have increased along [1]. The increase of the execution time and power required by these systems has had an evident negative financial, and environmental impact on any sector of the society and industry that relies on HPC systems, which is only expected to increase in the future.

One clear way to address this problem is to simply avoid or reduce the computations which will take excessive amounts of time and power. In order to do this, one approach is to use prediction models which could inform developers of the approximate execution time of these applications. During the last couple of years, there has been an increase in the interest regarding these prediction models of execution time and power of GPUs, with promising results. Nevertheless, there is still clear room for improvement.

II. Background

In this section, we briefly review different technologies and machine learning architectures related to our experiments.

A. PTX Code

Parallel Thread Execution (PTX) is a low-level programming model and instruction set architecture (ISA) for GPU's CUDA applications. PTX code is generated by high-level language compilers such as C/C++ and CUDA. It supports a wide range of GPU architectures and generations [18].

A PTX program is simply a text source file with a syntax similar to that of assembly language. PTX has its own defined syntax with specific keywords for directives, instruction statements, identifies, constants, state spaces, types, and variables. [18]

B. CUDA-Flux

CUDA-Flux is a profiler integrated to the LLVM framework which collects statistics about the control flow of a CUDA application and analyzes its PTX code [4]. The CUDA-Flux gives the user an output containing information such as the kernel name, grid and block dimensions, shared memory size, profiling mode, and details about the count of different instructions for each block.

C. Random Forests

Random forest is a machine learning technique used for both classification and regression tasks. This technique is considered an ensemble learning algorithm that aggregates different decision trees and averages over all of them to give a final result in the case of regression tasks. For classifications tasks, it chooses the class selected by the majority of the trees. [17].

D. CatBoost

CatBoost is an open-source framework that allows users to use gradient boosting on decision trees. Gradient-boosting on decision trees are common replacements to random forests, as they tend to give a better performance in different applications.

E. TabNet

TabNet is a deep learning architecture that specializes in efficiently learning from tabular data and yields interpretable results by using sequential attention to decide on each decision task what features to learn from. TabNet has been shown to perform better than any other decision tree based technique such as Random forests, Stochastic decision trees, and even Gradient boosted trees [3].

III. Related Work

There are three major categories of predicting models, namely, analytical, machine learning, and hybrid models. One of the most extensive reviews of these methods is presented in [5].

A. Analytical Models

Analytical models have been shown to be an important approach for predicting the execution time of a kernel using PTX code. One promising approach in [1] proposes a model as a function of two parameters, Delay Composition Algorithm (DCA) and GPU Simulation algorithm (GSA). To calculate DCA, the authors collected micro-benchmarking details for the instruction types and other hardware specifications, and they built a control flow graph for calculating the delay of the instructions obtaining an absolute error of 26.86 % in the predictions for execution time.

Authors in [10] used the number of parallel memory requests as the main input of their analytical model to calculate the execution time of GPU applications obtaining a geometric mean of absolute error of 13.3 %. Other works such as [11] used metrics such as the maximum number of memory requests that can be served concurrently and the number of warps that in a memory access period can finish one computational period to predict execution time.

B. Machine Learning Models

Machine learning (ML) methods for predicting the execution time of kernels have become the most common approach in the research community to tackle this problem. Some of the used techniques include k-nearest-neighbor [14], support vector machines [2], random forests [2][12] [16], artificial neural networks [21][13][19], and finally long short-term memory networks [9].

C. Hybrid Models

Hybrid models combine both analytical and ML models and have been explored in [22].

IV. Datasets and Benchmarks

A. Benchmarks

The dataset used for training and testing our Machine Learning models was created by the authors in [5]. The benchmarks used to obtain the data were Rodinia 3.1 [6], Parboil 2.5 [20], SHOC [7], and Polybench-GPU 1.0 [8]. A table of the applications used for each benchmarks is included [5].

These are some of the applications for each of the benchmarks:

- **Rodinia 3.1:** 3D, b+tree, bfs, backprop, dwt2d, euler3d, gaussian, heartwall, ludcuda, myocyte, needle, particlefilternaive, particlefilterfloat, scgpu.
- **Parboil 2.5:** cutcp, histo, lbm, mri-q, sgemm, stencil, tpacf.
- **SHOC:** BFS, FFT, MD5Hash, MaxFlops, Reduction, S3D, Scan, Sort, Stencil2D, Triad.
- **Polybench-GPU 1.0:** 2DConvolution, 2mm, 3DConvolution, 3mm, atax, bicg, correlation, covariance, fdtd2d, gemm, gesummv, gramschmidt, mvt, syr2k, syrk.

B. Datasets

The authors in [5] collected the data using the four benchmarks shown before using different NVIDIA GPUs. A complete overview of the GPU's specifications is shown in [5]. A summary of this information of some of the GPUs is shown below:

- K20: 3.5 TFLOP/s float perf, 208 GB/s Memory BW, 13 SMs, 2496 SPs
- Titan XP: 12.0 TFLOP/s float perf, 548 GB/s Memory BW, 30 SMs, 3840 SPs
- P100: 9.3 TFLOP/s float perf, 732 GB/s Memory BW, 56 SMs, 3884 SPs
- V100: 14.0 TFLOP/s float perf, 900 GB/s Memory BW, 80 SMs, 5120 SPs
- GTX 1650: 3.0 TFLOP/s float perf, 128 GB/s Memory BW, 14 SMs, 896 SPs

The authors used the CUDA-Flux profiler tool to obtain the features for each experiment. Furthermore, the authors repeated each experiment 10 times to avoid the likelihood of getting outliers.

V. Methodology

A. Prediction Models

Instead of training different models for different architectures, we combined data from different architectures, included GPU architectures in input features, and trained one generic model. Our model is generic and works with the following eight architectures: K20, Titan XP, P100, V100, GTX 1650, K80, M60, and T4.
We experimented with three machine learning algorithms: Random Forests, TabNet, and CatBoost. Ensemble learning algorithms such as Random Forests are among the best performing Machine Algorithms for structured tabular data. We choose Random Forests because of its simplicity, performance, popularity, and less training time. Gradient boosted decision trees are an improved version of Random Forests and beat the performance of random forest in many tabular data applications. Finally, we decided to use TabNet, because as explained in the background section, this deep learning architecture has

been shown to perform better than any other technique using Decision trees [3].

B. Tools and Frameworks

The data processing, model training, and testing were done using Python. For pre-processing, the main libraries used were Sklearn, Pandas, and Numpy. Sklearn is one of the main extensive, stable, and widely used libraries for Machine Learning and Data Science applications and played a significant role in this work.

We used PyTorch-tabnet for the TabNet's model, Sklearn's RandomForestRegressor for the Random Forest's model, and CatBoot's CatBoostRegressor for the Gradient boosted decision trees model.

C. Feature Engineering

We followed the same approach as [5] for feature selection and grouped different instructions into more general groups of instructions, namely, threads per CTA, CTAs, the total number of instructions, special operations, logic operations, control operations, arithmetic operations, sync operations, global memory volume, param memory volume, shared memory volume, arithmetic intensity. The ground truth value was the time to execute the kernel in different architectures. To add the CUDA architecture as an input feature we used one-hot encoding. Also, normalized the data using min-max normalization.

D. Error Metric

As explained in [5], absolute error-based metrics are not suitable for our dataset since they will give more weightage to errors in long-running kernels than short running kernels. Thus, a relative error measurement like Mean Absolute Percentage Error (MAPE) is more suitable. MAPE usually works well when the dataset has samples with a wide-scale [15], which is the case for the dataset used in this work. The MAPE equation is shown below, where y_{true} is the ground truth value, y_{pred} is the predicted value and n is the number of samples in the dataset.

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_{true} - y_{pred}}{y_{true}}\right| \qquad (1)$$

VI. Experimental Results

We trained CatBoost with default hyper-parameters except for the maximum depth of the three and the number of iterations. We used 16 for maximum depth and 200 for the number of iterations. For random forest as well, we used maximum depth 16 with other default hyperparameters. For TabNet, we used a 0.001 learning rate with default hyperparameters. We leave extensive hyperparameters tunning for future work.

The final results of our experiments for models TabNet, CatBoost, and Random Forests are shown in tables I and II. Tables I and II show the raw and normalized distribution of the execution time prediction errors in our testing dataset, respectively.

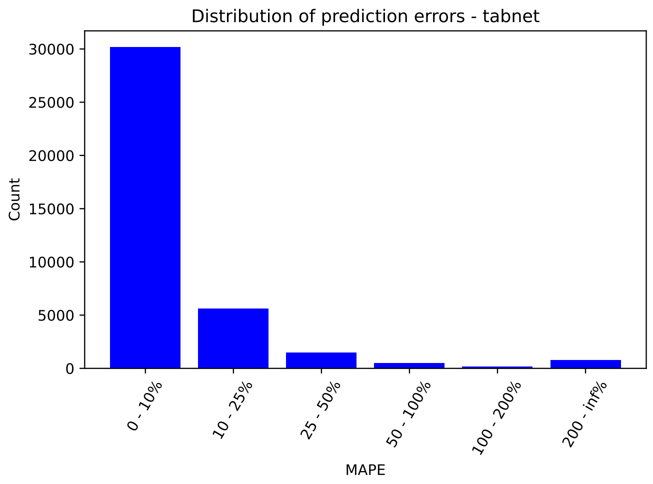

Fig. 1. Tabnet error Distribution

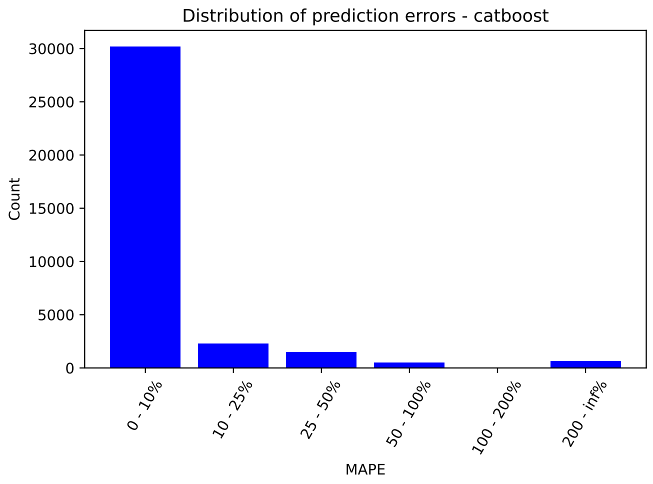

Fig. 2. Catboost error Distribution

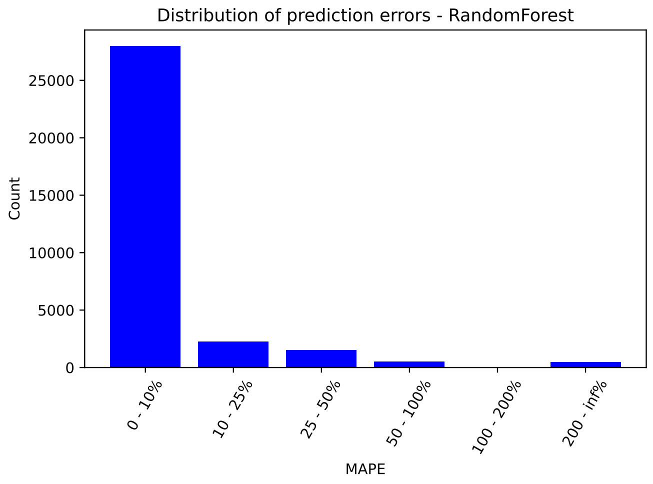

Fig. 3. Random Forest error Distribution

TABLE I
Distribution of Errors; Each cell represent the number of test data samples; for example, 25237 test points had MAPE error between 0 to 10 percent for TabNet

Distribution of Execution Time Prediction Error						
Model	**0-10%**	**10-25%**	**25-50%**	**50-100%**	**100-200%**	**200%-inf**
TabNet	25237	5610	796	368	149	788
CatBoost	30193	1820	166	123	28	618
Random Forest	27985	2258	1497	508	28	475

From the normalized distributions in table II, we can observe that the best performing model in terms of the number of predictions with an error in the range of (0-10%) is CatBoost with 91.6%, Random Forests with 85.4%, and finally TabNet with 76.6%. The performance of the CatBoost model and Random Forest model was expected to be good since our dataset is well structured and went through detailed feature engineering. We hypothesize that the performance of TabNet is impacted by the training data set size. TabNet architecture contains a neural network component that requires a large amount of data to perform well.

TABLE II
Normalized Distribution of Errors; Each cell represent the percentage of total test records; For example, 76.6 percent of test points had MAPE error between 0 to 10 percent for TabNet

Normalized Distribution of Execution Time Prediction Error						
Model	**0-10%**	**10-25%**	**25-50%**	**50-100%**	**100-200%**	**200%-inf**
TabNet	0.766	0.170	0.024	0.011	0.005	0.024
. **CatBoost**	0.916	0.055	0.005	0.004	0.001	0.019
Random Forest	0.854	0.069	0.046	0.016	0.001	0.015

Figures 1, 2, and 3 show the distribution of prediction errors corresponding to the table I for the models TabNet, CatBoost, and RandomForests, respectively. We can observe that the distributions of the errors look very similar for the three models, suggesting that either of these models would serve the purpose of predicting the execution time of kernels within a reasonable range of errors.

VII. Conclusion

We have shown that our models based on ensemble learning techniques such as Random Forest and Gradient Boosted Decision trees, as well as the deep learning architecture TabNet, gave a sensible performance at predicting the execution time of a specific kernel. We used data obtained using the CUDA-Flux profiler from the PTX code as input features.

We can conclude that the best performing model in terms of the number of predictions with an error in the range of (0-10%) is CatBoost with 91.6%, Random Forests with 85.4%, and finally TabNet with 76.6%.

Future Work

Future work can include more effort on data collection and feature engineering. Also, we believe extensive hyperparameter tuning would give better performance. We can also try combining some hardware-level features.

Acknowledgment

We would like to thank Dr. Mohamed Zahran for his support and guidance. We would also like to thank the staff at the Courant Institute of Mathematical Sciences for providing the computing resources to complete this work.

References

[1] Gargi Alavani, Kajal Varma, and Santonu Sarkar. “Predicting Execution Time of CUDA Kernel Using Static Analysis”. In: *2018 IEEE Intl Conf on Parallel Distributed Processing with Applications, Ubiquitous Computing Communications, Big Data Cloud Computing, Social Computing Networking, Sustainable Computing Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom)*. 2018, pp. 948–955. DOI: 10.1109/BDCloud.2018.00139.

[2] Marcos Amarís et al. “A comparison of GPU execution time prediction using machine learning and analytical modeling”. In: *2016 IEEE 15th International Symposium on Network Computing and Applications (NCA)*. 2016, pp. 326–333. DOI: 10.1109/NCA.2016.7778637.

[3] Sercan O. Arik and Tomas Pfister. *TabNet: Attentive Interpretable Tabular Learning*. 2020. arXiv: 1908.07442 [cs.LG].

[4] L. Braun and H. Froning. “CUDA Flux: A Lightweight Instruction Profiler for CUDA Applications”. In: *2019 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*. Los Alamitos, CA, USA: IEEE Computer Society, Nov. 2019, pp. 73–81. DOI: 10.1109/PMBS49563.2019.00014. URL: https://doi.ieeecomputersociety.org/10.1109/PMBS49563.2019.00014.

[5] Lorenz Braun et al. “A Simple Model for Portable and Fast Prediction of Execution Time and Power Consumption of GPU Kernels”. In: *CoRR* abs/2001.07104 (2020). arXiv: 2001.07104. URL: https://arxiv.org/abs/2001.07104.

[6] Shuai Che et al. “Rodinia: A benchmark suite for heterogeneous computing”. In: *2009 IEEE International Symposium on Workload Characterization (IISWC)*. 2009, pp. 44–54. DOI: 10.1109/IISWC.2009.5306797.

[7] Anthony Danalis et al. “The Scalable Heterogeneous Computing (SHOC) Benchmark Suite”. In: New York, NY, USA: Association for Computing Machinery, 2010. ISBN: 9781605589350. DOI: 10.1145/1735688.1735702. URL: https://doi.org/10.1145/1735688.1735702.

[8] Scott Grauer-Gray et al. "Auto-tuning a high-level language targeted to GPU codes". In: *2012 Innovative Parallel Computing (InPar)*. 2012, pp. 1–10. DOI: 10.1109/InPar.2012.6339595.

[9] João Guerreiro et al. "GPU Static Modeling Using PTX and Deep Structured Learning". In: *IEEE Access* 7 (2019), pp. 159150–159161. DOI: 10.1109/ACCESS.2019.2951218.

[10] Sunpyo Hong and Hyesoon Kim. "An Analytical Model for a GPU Architecture with Memory-Level and Thread-Level Parallelism Awareness". In: 37.3 (2009). ISSN: 0163-5964. DOI: 10.1145/1555815.1555775. URL: https://doi.org/10.1145/1555815.1555775.

[11] Sunpyo Hong and Hyesoon Kim. "An integrated GPU power and performance model". In: *ACM SIGARCH Comput. Archit. News 2010* (), pp. 280–289.

[12] Beau Johnston, Gregory Falzon, and Josh Milthorpe. "OpenCL Performance Prediction using Architecture-Independent Features". In: *2018 International Conference on High Performance Computing Simulation (HPCS)*. 2018, pp. 561–569. DOI: 10.1109/HPCS.2018.00095.

[13] Sajib Kundu et al. "Application performance modeling in a virtualized environment". In: Feb. 2010, pp. 1–10. DOI: 10.1109/HPCA.2010.5463058.

[14] Christoph Lehnert et al. "Performance Prediction and Ranking of SpMV Kernels on GPU Architectures". In: *Proceedings of the 22nd International Conference on Euro-Par 2016: Parallel Processing - Volume 9833*. Berlin, Heidelberg: Springer-Verlag, 2016, pp. 90–102. ISBN: 9783319436586. DOI: 10.1007/978-3-319-43659-3_7. URL: https://doi.org/10.1007/978-3-319-43659-3_7.

[15] Eryk Lewinson. *Choosing the correct error metric: Mape vs. Smape*. Nov. 2020. URL: https://towardsdatascience.com/choosing-the-correct-error-metric-mape-vs-smape-5328dec53fac.

[16] Abhinandan Majumdar et al. "Dynamic GPGPU Power Management Using Adaptive Model Predictive Control". In: *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2017, pp. 613–624. DOI: 10.1109/HPCA.2017.34.

[17] Rachel Meltzer. *What is Random Forest? [beginner's guide + examples]*. July 2021. URL: https://careerfoundry.com/en/blog/data-analytics/what-is-random-forest/.

[18] *Parallel thread execution Isa version 7.5*. URL: https://docs.nvidia.com/cuda/parallel-thread-execution/index.html.

[19] Shuaiwen Song et al. "A Simplified and Accurate Model of Power-Performance Efficiency on Emergent GPU Architectures". In: *2013 IEEE 27th International Symposium on Parallel and Distributed Processing*. 2013, pp. 673–686. DOI: 10.1109/IPDPS.2013.73.

[20] J.A. Stratton et al. "Parboil: A Revised Benchmark Suite for Scientific and Commercial Throughput Computing". In: *Center for Reliable and High-Performance Computing* (2012).

[21] Qiang Wang and Xiaowen Chu. "GPGPU Performance Estimation with Core and Memory Frequency Scaling". In: (Jan. 2017).

[22] Xiebing Wang et al. "A Hybrid Framework for Fast and Accurate GPU Performance Estimation through Source-Level Analysis and Trace-Based Simulation". In: *2019 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. 2019, pp. 506–518. DOI: 10.1109/HPCA.2019.00062.