 DOI 10.1109/ISPASS55109.2022.00013

XFEATUR: Hardware Feature Extraction for DNN Auto-tuning

Jorge Sierra Acosta
Universitat Politècnica de Catalunya
Barcelona, Spain
jorge.sierra@upc.edu

Andreas Diavastos
Universitat Politècnica de Catalunya
Barcelona, Spain
andreas.diavastos@upc.edu

Antonio Gonzalez
Universitat Politècnica de Catalunya
Barcelona, Spain
antonio@ac.upc.edu

***Abstract*—In this work, we extend the auto-tuning process of the state-of-the-art TVM framework with XFEATUR; a tool that extracts new meaningful hardware-related features that improve the quality of the representation of the search space and consequently improve the accuracy of its prediction algorithm. These new features provide information about the amount of thread-level parallelism, shared memory usage, register usage, dynamic instruction count and memory access dependencies. Optimizing ResNet-18 with the proposed features improves the quality of the search space representation by 63% on average and a maximum of $2\times$ for certain tasks, while it reduces the tuning time by 9% (approximately 1.1 hours) and produces configurations that have equal or better performance (up to 92.7%) than the baseline.**

I. INTRODUCTION

Recent advancements in Deep Neural Networks (DNNs) are driven by important improvements in algorithms and new compute capabilities in modern hardware. New algorithms are being developed constantly to efficiently and accurately solve challenging applications such as autonomous driving, natural language processing, etc. However, the vast number of features and parameters that need to be explored to improve the performance of such algorithms on different hardware devices poses a new optimization challenge. Auto-tuning frameworks are being used to automatically search the parameter space and find the best software optimizations for a given application. The result of such a search highly depends on the quality of the representation of the optimization (search) space.

With this work, we introduce XFEATUR, a software tool that is integrated within the auto-tuning process of TVM (AutoTVM) to take advantage of the knowledge we have over the underlying hardware and include certain hardware metrics (e.g. shared memory allocation, register usage, number of threads) and hardware specs (e.g. number of processing units, memory hierarchy) in the description of the search space. In this work, we propose a new set of hardware-related features that improve the quality of the search space representation of different CNN tasks for a GPU platform and consequently improve the outcome of the search space exploration.

The main contributions of this work are:

- A pre-processing tool, called XFEATUR, that analyzes the target application source code and extracts several hardware-related features to be used by the auto-tuner;
- A new set of hardware-related features that combined with the current software-based features from AutoTVM improve the quality of the search space representation by 63% on average, with a maximum of $2\times$, according to the Pearson Coefficient. Thus, reducing the tuning time of an application by 9%, that translates to 1.1 hours; besides, the solutions found by XFEATUR achieve equal or better performance (up to 92.7% in our benchmarks) than the ones encountered by the baseline AutoTVM.

II. MOTIVATION

Most popular Deep Learning (DL) frameworks such as TensorFlow [1], MXNet [2], Caffe [3] and PyTorch [4] rely on a computational graph intermediate representation that requires manual tuning and can perform only high-level optimizations unless they make use of special device instructions or hand-tuned optimizations. New optimization frameworks offer tools to automatically fine-tune (auto-tune) specific nodes in the computational graph to increase performance. Older techniques [5], [6] make use of analytical methodologies for automatic optimization, while more modern solutions employ evolutionary algorithms to perform an optimization search for the best DNN-to-hardware mapping, considering tiling, computation order and exploiting multiple levels of parallelism [7]–[9]. Recently, several solutions turned to more intelligent, machine learning, techniques such as supervised learning and reinforcement learning to refine the search space exploration and improve the performance of the search [10]–[14]. One such popular framework is TVM [15], an automated end-to-end compiler and auto-tuner with different sets of features to generate a search space that is explored with Simulated Annealing (SA) [16] and finally predicts the performance of various configurations with XGBoost [17].

Previous works are mostly focused on improving the search heuristics using software- and algorithm-specific features to represent the search space. We observe that there is a lack of hardware-related information being used in auto-tuning tools. The growing diversity of the available hardware devices increases the number of parameters that can be explored, but at the same time, it offers new features that can be correlated with performance metrics to help the auto-tuning process. We argue that a search space that is not represented with meaningful hardware features makes the search space

explorer likely to miss-interpret the performance difference among different data points, resulting in a slow and potentially inaccurate auto-tuning process. The target of this work is to provide a tool that can seamlessly and effortlessly extract hardware-related features from the application source code and the target hardware, in an effort to improve the already existing application feature set of TVM. Our goal is to improve the overall accuracy and performance of the AutoTVM tuning process by improving the search space representation and allowing for better prediction accuracy using the same cost model.

III. Hardware-related Optimization Features

We examine a new set of hardware-related features that extends the AutoTVM feature set to improve the quality of the search space representation. We separate the proposed feature set into two categories: the *Explicit* features that have a direct relation to measurable hardware values and the *Implicit* features that are hardware-related and can be inferred with reasonable approximation. To automate the process of extracting these new features, we developed a pre-processing tool, called XFeatur, that integrates with AutoTVM and extracts the new features at the pre-compile stage.

A. Explicit Features

1) Number of threads and number of threadblocks: On a GPU these are directly correlated with workload balancing and memory access latency by leveraging parallelism and warp scheduling. These two features can be easily obtained as they are defined by the kernel that will be executed. Analysis reveals that too few threads result in under-utilization of the system and too many threads saturate shared resources (*e.g.* too many registers per thread, shared memory overuse).

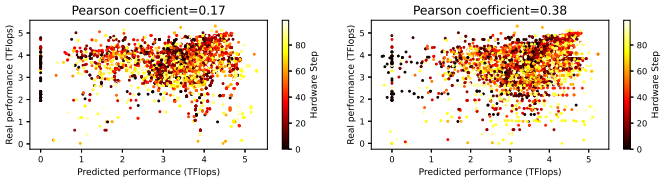
2) Shared memory usage: The shared memory offers one of the lowest memory access latency on a GPU. For this reason, it is commonly used for storing data that will be used and shared many times during the computation between threads in the same threadblock. Alternatively, using the caches that have a higher latency. Therefore, under-utilization of the shared memory when executing an application with high data reuse can be an early indication of low application performance.

3) Register usage: The number of registers used in each threadblock is also a good indicator of performance. Compiler-generated code that has more live variables than the GPU has registers to allocate, register spilling occurs and live variables must use the local memory, leading to longer access latency. We can approximate the number of registers with reasonable precision by calculating the number of local variable allocations it will perform and use this feature to represent the probability of register spilling during the execution.

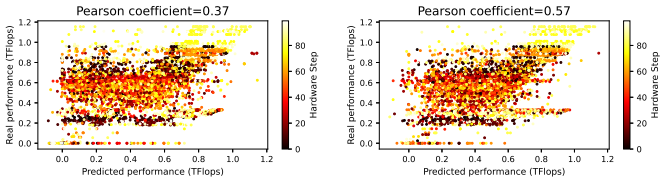
B. Implicit Features

1) Dynamic instruction count: The number of dynamic instructions provides important information on the ratio of various types of instructions that could stall the execution compared to useful progress (compute instructions). In this work, we use an approximation method to extract the dynamic number of instructions from the source code with static analysis and interpretation of the loops and conditional branches and categorize the instructions based on their type.

2) Number of cycles between data-dependent instructions: The number of instructions and the number of cycles spent between memory accesses and their dependent instructions are good indicators of how well a given application configuration can hide the latency of costly memory accesses. Using the categorization from the previous feature we build a histogram of the number of working cycles between a memory access instruction and its first dependent consumer.



(a) Task 4: AutoTVM (left), XFeatur (right)



(b) Task 13: AutoTVM (left), XFeatur (right)

Fig. 1: The search space representation for various tasks of ResNet-18 with plan size 8, expressed according to the Pearson Correlation Coefficient [20]. The color map represents the iterations of the auto-tuning process (measured in hardware steps). Note that the predicted performance is not bounded by the minimum (0 TFlops) or maximum possible performance.

IV. Experimental Evaluation

XFeatur is developed in Python and integrated into TVM v0.8.dev0. ResNet-18 [18] was used for the evaluation because of it's diverse set of compute- and memory-intensive convolutional layers, and it's fast enough to allow us to run multiple tests in a reasonable amount of time. All tests were evaluated on an Nvidia GeForce RTX 2070 SUPER GPU that implements the Turing architecture with 40 Streaming Multiprocessors [19].

The purpose of this work is to improve the quality of the space representation that will allow us to converge to optimal solutions faster during the auto-tuning process. To evaluate the quality of the search space we use the Pearson Correlation Coefficient [20]. Figure 1 shows the search space expressed according to the Pearson Correlation Coefficient with predicted performance on the x-axis and real performance achieved on the y-axis. The color map represents the iterations of the auto-tuning process (measured in hardware steps). The higher the correlation coefficient, the better the space is represented. In this paper, we show results only for plan size 8 and Tasks 4,

and 13, however, results for other tasks and with plan size 64 (AutoTVM default) show the same behavior. The improvement in the correlation coefficient when using the hardware-related features from Section III, extracted with XFEATUR, for Tasks 4, 13 is 2× and 54% respectively, compared to the baseline AutoTVM implementation that uses only its default feature sets. For all the tasks evaluated in ResNet-18 we achieve an average improvement of 63%.

The improvement of the quality of the search space translates into performance gains during the auto-tuning process. Our evaluation showed a reduction in the tuning time of an application by 9%, that translates to 1.1 hours. In addition, the solutions found by XFEATUR achieve equal or better performance (up to 92.7% in our benchmarks) than the ones encountered by the baseline AutoTVM.

ACKNOWLEDGMENT

This work has been supported by the CoCoUnit ERC Advanced Grant of the EU's Horizon 2020 program (grant No 833057), the Spanish State Research Agency (MCIN/AEI) under grant PID2020-113172RB-I00, and the ICREA Academia program and the FPU grant 2019-FPU-998758.

REFERENCES

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[2] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang, "Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems," *CoRR*, vol. abs/1512.01274, 2015. [Online]. Available: http://arxiv.org/abs/1512.01274

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[5] R. Whaley and J. Dongarra, "Automatically tuned linear algebra software," in *SC '98: Proceedings of the 1998 ACM/IEEE Conference on Supercomputing*, 1998, pp. 38–38.

[6] M. Frigo and S. Johnson, "Fftw: an adaptive software architecture for the fft," in *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP '98 (Cat. No.98CH36181)*, vol. 3, 1998, pp. 1381–1384 vol.3.

[7] S.-C. Kao and T. Krishna, "Gamma: Automating the hw mapping of dnn models on accelerators via genetic algorithm," in *Proceedings of the 39th International Conference on Computer-Aided Design*, ser. ICCAD '20. New York, NY, USA: Association for Computing Machinery, 2020. [Online]. Available: https://doi.org/10.1145/3400302.3415639

[8] N. Vasilache, O. Zinenko, T. Theodoridis, P. Goyal, Z. DeVito, W. S. Moses, S. Verdoolaege, A. Adams, and A. Cohen, "Tensor comprehensions: Framework-agnostic high-performance machine learning abstractions," *CoRR*, vol. abs/1802.04730, 2018. [Online]. Available: http://arxiv.org/abs/1802.04730

[9] J. Mu, M. Wang, L. Li, J. Yang, W. Lin, and W. Zhang, "A history-based auto-tuning framework for fast and high-performance dnn design on gpu," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[10] I. Baldini, S. J. Fink, and E. Altman, "Predicting gpu performance from cpu runs using machine learning," in *2014 IEEE 26th International Symposium on Computer Architecture and High Performance Computing*, 2014, pp. 254–261.

[11] S. Zheng, Y. Liang, S. Wang, R. Chen, and K. Sheng, "Flextensor: An automatic schedule exploration and optimization framework for tensor computation on heterogeneous system," in *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 859–873. [Online]. Available: https://doi.org/10.1145/3373376.3378508

[12] B. H. Ahn, P. Pilligundla, and H. Esmaeilzadeh, "Reinforcement learning and adaptive sampling for optimized DNN compilation," *CoRR*, vol. abs/1905.12799, 2019. [Online]. Available: http://arxiv.org/abs/1905.12799

[13] N. Ardalani, C. Lestourgeon, K. Sankaralingam, and X. Zhu, "Cross-architecture performance prediction (xapp) using cpu code to predict gpu performance," in *Proceedings of the 48th International Symposium on Microarchitecture*, ser. MICRO-48. New York, NY, USA: Association for Computing Machinery, 2015, p. 725–737. [Online]. Available: https://doi.org/10.1145/2830772.2830780

[14] K. Hegde, P.-A. Tsai, S. Huang, V. Chandra, A. Parashar, and C. W. Fletcher, "Mind mappings: Enabling efficient algorithm-accelerator mapping space search," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 943–958. [Online]. Available: https://doi.org/10.1145/3445814.3446762

[15] T. Chen, T. Moreau, Z. Jiang, L. Zheng, E. Yan, H. Shen, M. Cowan, L. Wang, Y. Hu, L. Ceze, C. Guestrin, and A. Krishnamurthy, "TVM: An automated end-to-end optimizing compiler for deep learning," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. Carlsbad, CA: USENIX Association, Oct. 2018, pp. 578–594. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/chen

[16] S. Kirkpatrick, C. Gelatt, and M. Vecchi, "A heuristic algorithm and simulation approach to relative location of facilities," *Optim. Simulated Annealing*, vol. 220, pp. 671–680, 1983.

[17] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," *CoRR*, vol. abs/1603.02754, 2016. [Online]. Available: http://arxiv.org/abs/1603.02754

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[19] "Nvidia ampere ga102 gpu architecture whitepaper," 2020. [Online]. Available: https://www.nvidia.com/content/PDF/nvidia-ampere-ga-102-gpu-architecture-whitepaper-v2.pdf

[20] D. Freedman, R. Pisani, and R. Purves, "Statistics (international student edition)," *Pisani, R. Purves, 4th edn. WW Norton & Company, New York*, 2007.