Mapping Estimator for OpenCL Heterogeneous Accelerators

André Bannwart Perina and Vanderlei Bonato
Institute of Mathematics and Computer Sciences, University of São Paulo
São Carlos - SP - Brazil
Email: abperina@usp.br, vbonato@usp.br

***Abstract*—To increase computing performance while keeping energy consumption to an acceptable budget, heterogeneous systems are currently investigated. By using dedicated compute units as accelerators to speedup specific parts of an application, hardware resources are better utilised resulting in a more energy efficient computing system. However, the task of performing such application mapping to accelerators is still a challenge, requiring knowledge beyond software domain in order to understand which part of the code fits better to the capability of the hardware available. Currently, there are tools supporting unified frontends and languages to simplify the programming of such heterogeneous systems, however there is still a high dependency of the user to manually perform the final mapping process. This work exposes a machine learning framework used to automatically infer the most suitable accelerator (between FPGA and GPU) for a given code by statically estimating energy efficiency. This framework can be used to assist the developer in deciding the best mapping for its application with an average hit-rate of 85 percent.**

I. Introduction

Since the traditional von Neumann architecture proposal in 1945, improvements were mainly dictated by modifications such as increasing the operational frequency. This trend was maintained until the first years of the new millenium where a power limitation was reached [1]. Trends since then directed to increasing the parallel capability of the available architectures while maintaining the operational frequency and power budget to feasible levels. In this scenario, alternate architectures such as the Graphics Processing Unit (GPU) and Field-Programmable Gate Array (FPGA) became popular as they can be used to speedup specific portions of an application. However, it is not trivial to efficiently partition a sequential software to a parallel model suitable to GPUs. Even worse, FPGAs are mainly programmed using Register-Transfer Level (RTL) languages where applications are mapped in terms of combinational circuits and registers. To alleviate the programming burden of FPGAs, High-Level Synthesis (HLS) tools convert high level software codes to RTL codes.

To assist the programming of heterogeneous systems, OpenCL was created [2] as a framework composed by a unified frontend where kernel functions (with resemblance to C) are programmed in different accelerators with vendor-specific backends. In the FPGA case, HLS tools were created to enable OpenCL support converting from kernel code to RTL, such as Xilinx SDAccel or Intel FPGA (former Altera) SDK for OpenCL. However, the task of choosing where to execute a kernel is still left to the developer. Also, accelerator-specific optimisations are necessary to obtain speedup or energy efficiency [3] and compilation for FPGAs requires circuit synthesis and place/route, which is a process that may take several hours. Therefore, manually exploring all different types of optimisations in FPGAs is unfeasible.

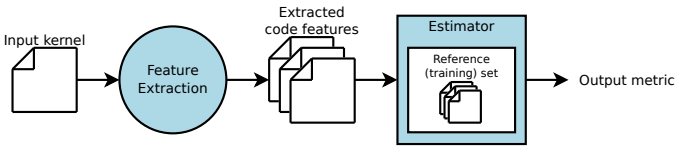

Fig. 1. Proposed framework flow.

This paper exposes a purely-static machine learning framework used to automatically decide the most suitable platform for an input OpenCL kernel among GPU and FPGA with suitability based on energy efficiency. Such analysis can give a quick mapping estimation avoiding the time-consuming synthesis. The main contribution is a framework capable of deciding the most suitable accelerator without actually synthesising the code or profiling it (i.e. no input data is needed). Results have shown that even though for some estimation metrics error is still significant, this framework was able to infer the most suitable accelerator for 85% of the cases in average.

The remainder of the paper is organised as follows: Section II formulates the problem and proposes the methodology. Experimental results are exposed and discussed in Section III. Related works are presented in Section IV. At last, the paper is concluded in Section V.

II. Formulation and Methodology

The main objective is to infer the most suitable OpenCL accelerator between FPGA and GPU considering energy efficiency for a given kernel. Kernels are usually coded with a specific platform in mind, requiring several structural modifications for proper platform migration, which is out of scope for this paper. Instead, this work focuses on giving a fast estimation on how well existing OpenCL codes would perform on each platform. Static analysis is performed to extract numerical values that describe several properties of the code, which are fed to a trained machine learning tool for estimation of metrics. Figure 1 presents the developed framework flow.

TABLE I
EXTRACTED CODE FEATURES

Feature code	Feature name	Longest path criterion
`lp`	Longest path	# of instructions
`noi`	Naive operational intensity	# of instructions
`nmi`	Naive memory intensity	# of bytes transferred
`fpops`	Floating-point operations	# of FP ops
`bars`	Number of barriers	# of barriers
`tc`	Maximum trip count	N/A
`ldep`	Deepest loop depth	N/A

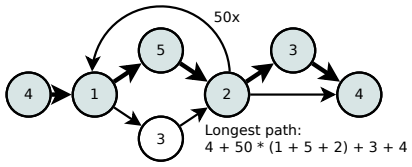

Fig. 2. Example of control-flow graph and its longest path in thicker edges. The weight of a node describes the amount of contained instructions. As an example, it is assumed a loop trip count of 50.

A. Features Extraction

The OpenCL kernel code features are extracted using OpCount[1], an analysis pass implemented using the LLVM compiler framework [4] based on its Intermediate Representation (IR) language. Table I presents the extractable code features.

The `lp` counts the number of IR instructions along the longest path of the control-flow graph (CFG) for the OpenCL code. Since finding the longest path for cyclic graphs is NP-hard, the loop back-edges are removed from the CFG making it acyclic. To compensate such removal, the number of instructions of all nodes inside loops are multiplied by the loops trip counts (i.e. how many times a loop executes). If loops trip counts are not statically inferrable, a default trip count is used (currently an arbitrary fixed value). Figure 2 presents an example of a CFG and its longest path. This metric can be interpreted as the worst case of execution, where loops are fully executed and the longest blocks are always taken from conditionals.

Both `noi` and `nmi` metrics are composed by counting the number of bytes transferred by loads and stores instructions and dividing by the longest path of the CFG. In the first metric, bytes are counted in the same path as the `lp` metric while the latter considers the longest path where the most amount of bytes has been transferred. Both metrics are based on the operational intensity concept used in the roofline model [5]. The naive characteristic comes from the fact that no input data is used to infer the code's execution path. For the description of the other code features, please refer to the OpCount repository.

Although static analysis may eventually not capture the precise execution profile for a code, it has the advantage of not needing any input data or profiling/simulation, nor any analysis on how well such execution reflects real-world usage of the code.

[1]This LLVM pass is available at https://github.com/comododragon/opcount

B. OpenCL Kernel Set

To train the machine learning tool, a set of 30 OpenCL kernels[2] was collected and adapted from three OpenCL benchmarks: Rodinia [6], SHOC [7] and CHO [8]. There were also 5 personal kernels added based on a Reed-Solomon Decoder implementation of Agarwal [9]. The known execution times, energy consumption and extracted code features were used to form the training database.

The following accelerators were used for execution time and energy consumption measurements:

- **GPU:** NVIDIA Quadro K620 (TDP: 41W);
- **FPGA:** Bittware S5PH-Q with an Intel FPGA Stratix V (TDP: 25W).

C. Output Metrics

The following metrics are proposed as outputs for the machine learning tool:

- **Energy Consumption:** the amount of energy consumed by a kernel. Currently this value is acquired by multiplying the execution time by the architecture's Thermal Design Power (TDP);
- **Class:** after calculating energy consumption, the kernel can be assigned to a class (i.e. FPGA or GPU) by considering the smallest consumption.

D. Machine Learning Tool

Since OpenCL performance estimation is non-trivial due to several subtleties such as memory access patterns or the HLS capability of extracting parallelism, it is suitable for machine learning exploration. Using MathWorks MATLAB R2015a, the following neural networks were used: learning vector quantisation (`lvq`), multi-layer perceptron (`mlp`) and radial basis function (`rbf`). In each network, several parameters were varied in order to explore different topologies and their performances. A single combination of such parameters is henceforth called setup.

To acquire a perspective on how well different combinations of input variables can contribute to the class assignment, `lvq` was explored before `mlp` or `rbf` since it has a faster training. However, the `lvq` network only estimates class assignment due to its discrete-only nature.

The cross-validation method with random subset sampling was used on all networks, where 100 trainings were performed for each possible setup. Each training/validation phase produces results that are analysed by a performance metric. For the continuous energy consumption, Root-Mean-Squared Error (RMSE) was used based on the Mean-Squared Error (MSE), which can be interpreted as the average squared difference between expected and estimated outputs (less is better). For the discrete class assignment metric, hit-rate was used, being a value between 0 and 1 (more is better). For example, a network

[2]**Rodinia:** kmeans, nn, nw1, nw2, srad, backprop1, backprop2, lud1, lud2, leukocyte1, leukocyte2, hotspot3D, hybridsort1, hybridsort2, hybridsort3, streamcluster, cfd. **SHOC:** md, md5hash, reduction. **CHO:** aes_enc, aes_dec, gsm, adpcm, mips. **Personal:** rsd1, rsd2, rsd3, rsd4, rsdfull.

TABLE II
SETUP PARAMETERS

Parameter	Possible values
LVQ	
No. of neurons	2, 4 and 8
Input variables sets	All features, (lp), (lp, noi), (lp, nmi), (lp, fpops, bars), (lp, noi, nmi, fpops, bars), (lp, noi, nmi, fpops, bars, ldep), (noi, nmi, fpops, bars, ldep), (fpops, bars)
Output metric	Class assignment
MLP	
Number of hidden layers	1, 2 and 3
Hidden layers topology	(5), (10), (50), (5, 5), (5, 10), (5, 50), (10, 10), (10, 50), (50, 50), (5, 5, 5), (5, 5, 10), (5, 5, 50), (5, 10, 10), (5, 10, 50), (5, 50, 50), (10, 10, 10), (10, 10, 50), (10, 50, 50), (50, 50, 50)
Input variables sets	All features and also the most accurate combinations from lvq
Output metric	Energy consumption and class assignment
RBF	
Spread	0.02, 0.03, 0.04, 0.05, 0.06 and 0.07
Input variables sets	All features and also the most accurate combinations from lvq
Output metric	Energy consumption and class assignment

with hit-rate of 0.9 implies that it was able to correctly infer the most suitable accelerator for 90% of the validation subset.

After all 100 trainings, the best, worst and average performance metrics were calculated for each possible setup. All input variables and output metrics were normalised prior to training and validation.

Table II presents the explored setup parameters for all networks.

III. EXPERIMENTAL RESULTS

Table III presents performance results for all networks. In lvq, results for all extracted features and two other setups with the best average performance are presented, while for mlp and rbf only the best setup is presented.

For lvq, the best average performance was for the setup (lp, nmi) with 4 neurons, reaching almost 85%. For mlp, the estimation error for energy is significant as pointed by the RMSE: the best setup has an average error of 16563.9 and 5362.9 for GPU and FPGA respectively for an unnormalised interval of $[2.0; 62507.4]$ for GPU and $[1.7; 20239.1]$ for FPGA, all in kilojoules (kJ). For rbf, interestingly the best performance was found when using all input variables, though several other setups for this network had almost the same performance. Considering energy consumption, not only did rbf perform slightly better (average error of $14376.2kJ$ and $4654.6kJ$ for GPU and FPGA respectively) but the worst RMSE was also smaller.

TABLE III
PERFORMANCE RESULTS

LVQ					
			Hit-rate		
Output	Input variables	Neurons	Worst	Best	Avg.
Class	All	4	0.400	1.000	0.764
	(lp, nmi)	4	0.400	1.000	0.849
	(lp, noi, nmi, fpops, bars)	8	0.400	1.000	0.826
MLP					
			RMSE / Hit-rate		
Output	Input variables	Neurons	Worst	Best	Avg.
Energy	(lp, nmi)	(5, 5)	0.664	0.063	0.265
Class	(lp, noi, nmi, fpops, bars)	(5)	0.200	1.000	0.798
RBF					
			RMSE / Hit-rate		
Output	Input variables	Spread	Worst	Best	Avg.
Energy	All	0.03	0.385	0.032	0.230
Class	(lp, noi, nmi, fpops, bars)	0.07	0.400	1.000	0.716

For both lvq and mlp, the input variables combination (lp, nmi) was present in almost all best results. For class assignment in mlp and rbf, the combination (lp, noi, nmi, fpops, bars) performed better.

An likely cause for energy estimation underperformance is the size of training and cross-validation subsets, not having sufficient coverage for an ideal estimation. However, the lvq network was able to correctly infer the most suitable platform for an average of almost 85%, outperforming all other approaches. Such result may be even further improved by increasing the number of samples for training and validation.

IV. RELATED WORKS

There are many works where kernel mapping is performed at runtime [10][11][12]. This is not suitable to FPGAs due to its time-consuming compilation (synthesis). Therefore, only works with compile-time decision making are presented.

Grewe [13] presents a framework to map OpenCL workloads to CPU-GPU heterogeneous environment, where code features are statically extracted from the kernel source and analysed by a machine learning technique. Their estimator was able to infer the correct workload mapping for 52% of the cases. It must be noted however that their estimator not only infers the best accelerator but also a workload proportion in each.

In Aladdin [14], estimation is inferred from high-level representations of sequential codes such as the Dynamic Data Dependence Graphs (DDDGs). With these graphs, they can generate more accurate estimations than the HLS tools, since a less conservative data dependency analysis is made. Average errors on power, performance and area when compared to actual synthesis are 0.9%, 4.9% and 6.5% respectively. As a limitation, estimations are totally dependent on input data,

therefore data which may excite all parts of the code must be provided.

Similar to Aladdin, Lin-Analyzer [15] also uses DDDGs. Since no HLS tool is used on each iteration of the DSE, estimations are ready in seconds. Differently from Aladdin, only part of the input code traces are analysed in the DDDG, reducing DSE time. Estimation error for 10 applications is below 6%.

FlexCL [16] is an analytical model is used to estimate performance of an OpenCL kernel on FPGAs through a computation model coupled to a global memory model with different data access patterns, having an accuracy of more than 90%.

Choi [17] presents HLScope+, where performance is estimated on codes for HLS tools such as Vivado or SDAccel. They present an analytical model to analyse memory latency, including resource contention when multiple functional units are requesting data. Results show that estimation errors are 1.1% and 5.0% for compute-bound and memory-bound applications respectively.

A. Comparison

The work here presented follows a similar tactic as proposed by Grewe, where code features are used as inputs for a machine learning framework. Although their work performs more complex partitioning by considering workload proportions, it focuses only on GPU and CPU. Adding FPGA as an accelerator would potentially require further analysis on how code features and FPGA performance are related.

Although Aladdin and Lin-analyzer provide good estimations, they are dependent of input data for generating the DDDGs, differently from the work here presented where no input data is required. Moreover, both are focused in C-based HLS, requiring adaptations for OpenCL estimation.

FlexCL and HLScope+ estimate only for FPGA. Furthermore, these works are focused on estimating the cycle count using a fixed frequency to estimate execution time and energy consumption, while the work here presented assumes that the operational frequency is optimised by the HLS tool.

V. Conclusion

This paper presented a machine learning framework composed of a neural network for statically inferring consumed energy and the most suitable accelerator among FPGA and GPU for an OpenCL kernel code using numerical features extracted from the code.

Several parameters for different neural networks were explored and validated through cross-validation. It was noted that although the estimated energy consumption have significant errors for all continuous networks, the `lvq` network was able to correctly infer the most suitable accelerator in almost 85% of the cases in average. It was also noted that the longest path and naive memory intensity code features were the most present in the best explorations. This work's result may be improved by using larger training and cross-validation sets, which could also contribute on reducing the estimation errors for the continuous metrics. Future work includes adding more kernels to the training set, exploring other code features and also adding estimation for FPGA resources.

Acknowledgment

The authors would like to thank FAPESP (Sao Paulo Research Foundation, grant no. 2016/18937-7) for the financial support given to this research project.

References

[1] A. R. Brodtkorb, C. Dyken, T. R. Hagen, J. M. Hjelmervik, and O. O. Storaasli, "State-of-the-art in Heterogeneous Computing," *Scientific Programming*, vol. 18, no. 1, pp. 1–33, 2010.

[2] Khronos Group, "OpenCL - The open standard for parallel programming of heterogeneous systems," 2018, available at https://www.khronos.org/opencl/, accessed 9th feb. 2018.

[3] H. R. Zohouri, N. Maruyama, A. Smith, M. Matsuda, and S. Matsuoka, "Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE Press, 2016, p. 35.

[4] C. Lattner, "The LLVM Compiler Infrastructure," 2018, available at http://llvm.org/, accessed 15th feb. 2018.

[5] S. Williams, A. Waterman, and D. Patterson, "Roofline: An Insightful Visual Performance Model for Multicore Architectures," *Communications of the ACM*, vol. 52, no. 4, pp. 65–76, 2009.

[6] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, "Rodinia: A Benchmark Suite for Heterogeneous Computing," in *Workload Characterization, 2009. IISWC 2009. IEEE International Symposium on*. IEEE, 2009, pp. 44–54.

[7] A. Danalis, G. Marin, C. McCurdy, J. S. Meredith, P. C. Roth, K. Spafford, V. Tipparaju, and J. S. Vetter, "The Scalable Heterogeneous Computing (SHOC) Benchmark Suite," in *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units*. ACM, 2010, pp. 63–74.

[8] G. Ndu, J. Navaridas, and M. Luján, "CHO: towards a benchmark suite for OpenCL FPGA accelerators," in *Proceedings of the 3rd International Workshop on OpenCL*. ACM, 2015, p. 10.

[9] A. Agarwal, M. C. Ng *et al.*, "A Comparative Evaluation of High-Level Hardware Synthesis Using Reed–Solomon Decoder," *IEEE Embedded Systems Letters*, vol. 2, no. 3, pp. 72–76, 2010.

[10] O. Souissi, R. B. Atitallah, D. Duvivier, and A. Artiba, "Optimization Of Matching and Scheduling On Heterogeneous CPU/FPGA Architectures," *IFAC Proceedings Volumes*, vol. 46, no. 9, pp. 1678–1683, 2013.

[11] A. M. Aji, A. J. Pena, P. Balaji, and W.-c. Feng, "Automatic command queue scheduling for task-parallel workloads in opencl," in *Cluster Computing (CLUSTER), 2015 IEEE International Conference on*. IEEE, 2015, pp. 42–51.

[12] E. Paone, F. Robino, G. Palermo, V. Zaccaria, I. Sander, and C. Silvano, "Customization of opencl applications for efficient task mapping under heterogeneous platform constraints," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2015*. IEEE, 2015, pp. 736–741.

[13] D. Grewe and M. F. O'Boyle, "A Static Task Partitioning Approach for Heterogeneous Systems using OpenCL," in *International Conference on Compiler Construction*. Springer, 2011, pp. 286–305.

[14] Y. S. Shao, B. Reagen, G.-Y. Wei, and D. Brooks, "Aladdin: A pre-rtl, power-performance accelerator simulator enabling large design space exploration of customized architectures," in *ACM SIGARCH Computer Architecture News*, vol. 42, no. 3. IEEE Press, 2014, pp. 97–108.

[15] G. Zhong, A. Prakash, Y. Liang, T. Mitra, and S. Niar, "Lin-analyzer: a high-level performance analysis tool for FPGA-based accelerators," in *Proceedings of the 53rd Annual Design Automation Conference*. ACM, 2016, p. 136.

[16] S. Wang, Y. Liang, and W. Zhang, "Flexcl: An analytical performance model for opencl workloads on flexible fpgas," in *Design Automation Conference (DAC), 2017 54th ACM/EDAC/IEEE*. IEEE, 2017, pp. 1–6.

[17] Y.-k. Choi, P. Zhang, P. Li, and J. Cong, "HLScope+: Fast and accurate performance estimation for FPGA HLS," in *Computer-Aided Design (ICCAD), 2017 IEEE/ACM International Conference on*. IEEE, 2017, pp. 691–698.