Autotuning OpenCL Workgroup Sizes

Chris Cummins*, Pavlos Petoumenos*,
Michel Steuwer*, Hugh Leather*

* *University of Edinburgh, UK*

ABSTRACT

Selecting appropriate workgroup sizes for OpenCL is critical for program performance, and requires knowledge of the underlying hardware, the data being operated on, and the kernel implementation. We propose the use of machine learning-enabled autotuning to automatically predict workgroup sizes for stencil patterns on CPUs and multi-GPUs, using the Algorithmic Skeleton library SkelCL. In an evaluation across 429 combinations of architecture, kernel, and dataset, we find that static tuning of workgroup size achieves only 26% of the optimal performance. Using machine learning and synthetically generated stencil programs, we achieve 92% of this maximum, demonstrating a median $3.79\times$ speedup over the best possible fixed workgroup size.

KEYWORDS: OpenCL; Autotuning; Stencil codes; GPUs; Machine learning; Synthetic benchmarking

1 Introduction

Efficient, tuned implementations of stencil codes are highly sought after, with a variety of applications from fluid dynamics to quantum mechanics. OpenCL enables the application of a range of heterogeneous devices to parallelise stencils; however, achieving portable performance is a hard task — OpenCL kernels are sensitive to properties of the underlying hardware, to the implementation, and even to the dataset that is operated upon. This forces developers to laboriously hand tune performance on a case-by-case basis, since simple heuristics fail to exploit the available performance. Iterative search approaches have been shown to select good values [1–3], but these processes are expensive and must be repeated for every new program or change to hardware and dataset. In this work we demonstrate how machine learning-enabled autotuning can be applied to Algorithmic Skeletons to achieve near optimal performance of unseen stencils operations, without requiring hand tuning or iterative searches.

[1]E-mail: c.cummins@ed.ac.uk, ppetoume@inf.ed.ac.uk, michel.steuwer@ed.ac.uk, hleather@inf.ed.ac.uk

[2]This work was supported by the UK EPSRC under grants EP/L01503X/1 (CDT in Pervasive Parallelism), EP/H044752/1 (ALEA), and EP/M015793/1 (DIVIDEND).

2 The SkelCL Stencil Pattern

SkelCL is an Algorithmic Skeleton library which provides OpenCL implementations of data parallel patterns for heterogeneous parallelism using CPUs and multi-GPUs [4]. Figure 1 shows the components of the SkelCL stencil pattern, which applies a user-provided *customising function* to each element of a 2D matrix. The value of each element is updated based on its current value and the value of one or more neighbouring elements, called the *border region*. When a SkelCL stencil pattern is executed, each of the matrix elements are mapped to OpenCL work-items; and this collection of work-items is divided into *workgroups* for execution on the target hardware. A *tile* of elements the size of the workgroup and the perimeter border region is allocated in local memory for use by work-items. Changing the workgroup size affects both the number of workgroups which can be active simultaneously, and the amount of local memory required for each workgroup. While the user defines the size, type, and border region of the matrix being operated upon, it is the responsibility of the SkelCL implementation to select an appropriate workgroup size to use.

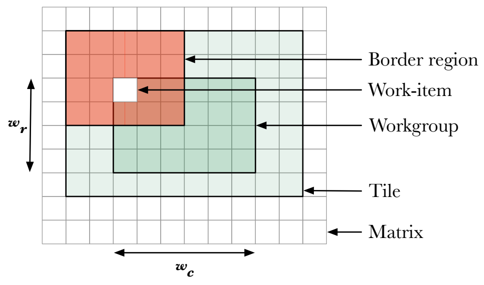
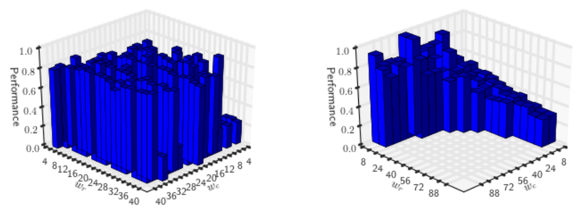


Figure 1: The execution of a stencil: an input matrix is decomposed into workgroups, and elements within workgroups are mapped to work-items. The values used by work-groups are mapped to local memory.

3 OpenCL Workgroup Size Optimization Space

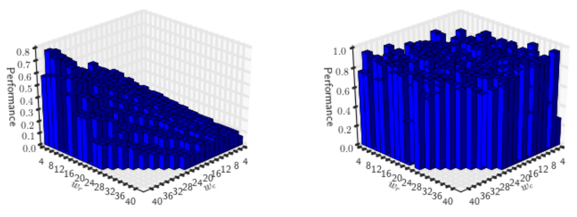
Preliminary experiments demonstrated that selection of an appropriate workgroup size w from the space of all possible workgroup sizes $w \in W$ depends on properties of the kernel, hardware, and dataset, illustrated in Figures 2 and 3. Additionally, the optimisation space is subject to hard constraints which cannot conveniently be statically determined. Each OpenCL device and kernel imposes a maximum workgroup size determined by the hardware resources and resource requirements of a program. Preliminary experiments found that not all points in the workgroup size space provide correct programs, even if they satisfy the kernel and architectural constraints. We call these points in the space *refused parameters*. For a given *scenario* s (a combination of program, device, and dataset), we define a *legal* workgroup size as one which satisfies the architectural and kernel constraints $w < W_{max}(s)$ and is not refused $w \notin W_{refused}(s)$. The subset of legal workgroup sizes for a given scenario $W_{legal}(s) \subset W$ is then:

$$W_{legal}(s) \subset W = \{w | w \in W, w < W_{\max}(s)\} - W_{refused}(s) \tag{1}$$



(a) (b)

Figure 2: The performance of different workgroup sizes for the same stencil program on two different devices: (a) Intel CPU, (b) NVIDIA GPU.



(a) (b)

Figure 3: The performance of different workgroup sizes for the same device executing two different stencil programs.

4 Autotuning Methodology

The goal of our machine learning-enabled autotuner is to predict, for a given scenario, the workgroup size that will provide the shortest runtime. The OpenCL workgroup size optimization space is large, complex, and non-linear. Successfully applying machine learning to this problem requires plentiful training data, the careful selection of explanatory variables, and appropriate machine learning methods:

Explanatory variables 102 variables are extracted to capture information about the device, program, and dataset of a scenario. Device variables capture information about the host and target hardware (e.g. number of compute units, local memory size, cache sizes). Program variables include static instruction densities, the total number of basic blocks, and the total instruction count. Dataset variables include the data types (input and output), and dimensions of the input matrix and stencil region.

Training Data A parameterised template substitution engine is used to generate synthetic stencil programs. To generate training data, these synthetically generated programs are executed on a range of different execution devices, using different datasets, and exhaustively enumerating the legal workgroup sizes. The workgroup size which gives the best performance for each particular device, program, and dataset is combined with the explanatory variables to create training data.

Classification Our approach uses a J48 Decision Tree classifier [5], due to its low runtime cost and ability to efficiently handle large dimensionality training data. In training, the classifier correlate patterns between explanatory variables and the workgroup sizes which provide optimal performance. After training, it predicts the workgroup size for new sets of explanatory variables. If the predicted workgroup size is not legal for that scenario, a *nearest neighbor* workgroup size is iteratively selected based on Euclidian distance to the predicted value.

5 Evaluation

To evaluate the efficacy of our approach we perform an exhaustive enumeration of the workgroup size optimisation space for 429 combinations of architecture, program, and dataset. Seven architectures are used (3 NVIDIA GPUs, 3 Intel CPUs, and 1 AMD GPU). Synthetically generated stencil programs are combined with six reference kernels taken from image processing, cellular automata, and partial differential equation solvers. Datasets of sizes 512×512, 1024×1024, 2048×2048, and 4096×4096 are used. We perform 10-fold cross validation to evaluate the prediction quality of the classifier, comparing the runtimes from predicted workgroup size against the workgroup size which provides the best average case performance across all scenarios (static tuning), and human expert selected workgroup size.

From our experimental results, we find an average $15.14\times$ performance gap between the best and workgroup sizes for each scenario. Of the 429 combinations of program, architecture, and dataset, 135 of them had unique optimal workgroup sizes. Statically tuning the workgroup size fails to extract this potential performance, achieving only 26% of the optimal performance. Our autotuner achieves good performance, with a median $3.79\times$ speedup of predicted workgroup sizes over static tuning in 10-fold cross-validation. The measured overhead of predicting a workgroup size using our autotuner is only 2.5ms, of which only 0.3ms is required for classification, with the remaining time required for feature extraction.

6 Conclusions

We present an machine learning-enabled autotuner for predicting the OpenCL workgroup size of stencil patterns. Our implementation is extensible and open source[3], and provides near-optimal prediction of workgroup sizes for unseen programs, devices, and datasets. By performing autotuning at the skeletal level, the system is able to exploit underlying similarities between pattern implementations which are not shared in unstructured code [6]. In future work we will use machine learning to perform a directed search through the program space by guiding the generation of synthetic benchmark programs.

References

[1] C. Nugteren and V. Codreanu. "CLTune: A Generic Auto-Tuner for OpenCL Kernels". In: *MCSoC*. 2015.

[2] J. Ansel, S. Kamil, K. Veeramachaneni, U. O. Reilly, and S. Amarasinghe. "OpenTuner: An Extensible Framework for Program Autotuning". In: *PACT*. ACM, 2013.

[3] Y. Zhang and F. Mueller. "Auto-generation and Auto-tuning of 3D Stencil Codes on GPU clusters". In: *CGO*. IEEE, 2012.

[4] Michel Steuwer, Philipp Kegel, and Sergei Gorlatch. "SkelCL - A Portable Skeleton Library for High-Level GPU Programming". In: *IPDPSW*. IEEE, May 2011.

[5] J. Han, M. Kamber, and J. Pei. *Data mining: concepts and techniques*. Elsevier, 2011.

[6] Y. Hu, D. M. Koppelman, and S. R. Brandt. "Model-Driven Auto-Tuning of Stencil Computations on GPUs". In: (2015).

[3]Source code available at: `https://github.com/ChrisCummins/phd/tree/master/src/omnitune`