ATF: A Generic Auto-Tuning Framework

Ari Rasch
University of Münster, Germany
a.rasch@wwu.de

Sergei Gorlatch
University of Münster, Germany
gorlatch@wwu.de

ABSTRACT

We describe the Auto-Tuning Framework (ATF) – a simple-to-use, generic framework for automatic program optimization by choosing the most suitable values of program parameters, such as number of parallel threads, tile sizes, etc. ATF combines four major advantages over the state-of-the-art auto-tuners: i) it is generic regarding the programming language, application domain, tuning objective, and search technique; ii) it can auto-tune a broader class of applications by allowing tuning parameters to be interdependent, e.g., when a parameter is divisible by another parameter; iii) it allows tuning parameters to have substantially larger ranges by implementing an optimized search space generation process; and iv) it is arguably simpler to use: the ATF user prepares an application for auto-tuning by annotating its source code with simple tuning directives. Our experimental results demonstrate that ATF shows significantly better tuning results as compared to the state-of-the-art auto-tuners OpenTuner and CLTune.

ACM Reference format:
Ari Rasch and Sergei Gorlatch. 2018. ATF: A Generic Auto-Tuning Framework. In *Proceedings of HPDC '18: The 27th International Symposium on High-Performance Parallel and Distributed Computing, Tempe, AZ, USA, June 11–15, 2018 (HPDC '18),* 2 pages.
https://doi.org/10.1145/3220192.3220194

1 MOTIVATION AND RELATED WORK

Auto-tuning is an approach to automatically find optimal values of tuning parameters, e.g., the number of parallel threads. To auto-tune a program, the programmer has to identify program's tuning parameters and perform the following three steps: 1) generate the application-specific *search space*, 2) implement a *cost function* for estimating program's cost, e.g., its runtime or energy consumption, and 3) explore the search space by using an automatized *search technique*. Auto-tuning systems have been successfully applied in different application areas, e.g., ATLAS [8] for linear algebra routines and PATUS [5] for stencil computations. The common weakness of these approaches is that they are not generic, i.e., they cannot be used for applications from other domains.

OpenTuner [4] and CLTune [3] are recent state-of-the-art auto-tuning approaches that are generic regarding the application domain: given the specification of the tuning parameters (i.e., their names and the ranges of possible values), OpenTuner and CLTune automatically generate a search space and explore it by using pre-implemented search techniques. However, OpenTuner does not provide mechanisms for expressing interdependencies between tuning parameters (e.g., some parameter must be divisible by another parameter). CLTune allows interdependencies but is applicable only for parameters with small ranges due to its time-intensive process of search space generation. Moreover, CLTune is restricted to auto-tuning only OpenCL and only in terms of runtime performance. Furthermore, both approaches – OpenTuner and CLTune – require from the user specific programming skills for auto-tuning, e.g., in Python (OpenTuner) or in C++ (CLTune), making auto-tuning hard for common application developers.

We propose the *Auto-Tuning Framework (ATF)* which combines the following advantages over the state-of-the-art auto-tuners:

i) ATF allows auto-tuning programs written in arbitrary programming languages and of arbitrary application domains, using a user-defined tuning objective and search technique;
ii) ATF allows interdependencies between tuning parameters by introducing *parameter constraints*;
iii) ATF allows substantially larger parameter ranges by optimizing the process of search space generation;
iv) ATF is arguably simpler to use: the user annotates program's source code with simple tuning directives, rather than having to implement an auto-tuning program, e.g., in Python or C++.

2 ILLUSTRATION OF ATF

We illustrate the usage of ATF by a simple example: auto-tuning the OpenCL `saxpy` kernel of the popular *CLBlast* library [2]. The kernel has two integer tuning parameters: the *work per thread (WPT)* and the OpenCL *local size (LS)*, which are interdependent: `LS` has to divide `N/WPT`, where `N` is the input size (line 1).

Listing 1 demonstrates how ATF is used for auto-tuning the `saxpy` kernel. The user annotates the kernel's source code (line 25) with *ATF tuning directives* (line 1-23): they specify 1) the tuning parameters (line 3-9) – interdependencies are expressed via parameter constraints (line 9), 2) the OpenCL cost function (line 11-17), e.g., by setting the device to use (line 11-12) and the kernel's input arguments (line 14-17), and 3) the search space exploration process, i.e., the search technique (line 22), e.g., *exhaustive search* or *simulated annealing*, and the *abort condition* which defines when to stop the tuning (line 23). The annotated source code is passed to ATF which yields as its result the best found configuration in terms of high runtime performance. To auto-tune for a tuning objective that is different from runtime performance, the auto-tuned program has to write its corresponding cost (e.g., its energy consumption) after each run to a *cost file* which is then read by ATF.

HPDC '18, June 11–15, 2018, Tempe, AZ, USA

ACM ISBN 978-1-4503-5899-6/18/06…$15.00
https://doi.org/10.1145/3220192.3220194

```
#atf::var::N $1

#atf::tp name       "WPT"
         range      interval<size_t>( 1,N )
         constraint divides( N )

#atf::tp name       "LS"
         range      interval<size_t>( 1,N )
         constraint divides( N/WPT )

#atf::ocl::platform "NVIDIA"
#atf::ocl::device   "Tesla K20"

#atf::ocl::input scalar<int>( N )   // N
#atf::ocl::input scalar<float>()    // a
#atf::ocl::input buffer<float>( N ) // x
#atf::ocl::input buffer<float>( N ) // y

#atf::ocl::global_size N/WPT
#atf::ocl::local_size  LS

#atf::search_technique annealing
#atf::abort_conition   duration<minutes>( 10 )

// SAXPY kernel's OpenCL code
```

Listing 1: ATF tuning directives for saxpy in OpenCL.

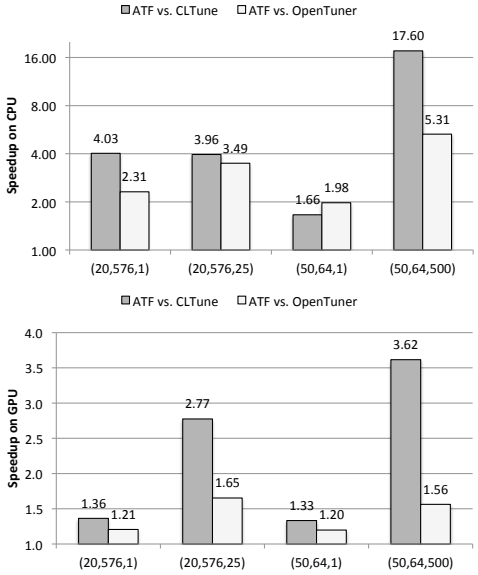

Fig. 2: Speedup (higher is better) of CLBlast's GEMM routine [2] auto-tuned by ATF over auto-tuning by CLTune and OpenTuner on Intel CPU (top) and NVIDIA GPU (bottom), using different input sizes (M, N, K) for $M \times K$ and $K \times N$ input matrices.

3 EXPERIMENTAL RESULTS

ATF provides significantly better tuning results as compared to CLTune and OpenTuner for the important routine GEMM (General Matrix Multiplication) [6] as implemented in the state-of-the-art auto-tunable CLBlast [2] library. For evaluation, we use a dual-socket system equipped with two Intel Xeon E5-2640 CPUs, as well as an NVIDIA Tesla K20m GPU.

Figure 2 shows the measured speedup of the GEMM routine auto-tuned by ATF as compared to the routine auto-tuned by CLTune and OpenTuner on four different input sizes that are heavily used in the deep-learning framework Caffe [9]. We employ the CLTune program that CLBlast uses for auto-tuning GEMM [2], and we implement the OpenTuner program for this kernel according to [7] where we use the unconstrained search space and set an `ERROR` state in case of an invalid configuration.

We observe that in comparison to CLTune, ATF improves GEMM's runtime by factors from 1.66× to 17.60× on the CPU (upper part of the figure, logarithmic scale), and from 1.33× to 3.62× on the GPU (lower part of the figure). The reason is that CLBlast artificially limits CLTune's tuning parameter ranges, apparently because of CLTune's time-intensive process of search space generation, thereby missing optimal solutions.

Compared to OpenTuner, ATF speedups the GEMM routine by factors from 1.98× to 5.31× on the CPU (top), and by factors from 1.20× to 1.65× on the GPU (bottom). This is because OpenTuner uses unconstrained search spaces and, thus, cannot find a valid configuration since they make only a tiny fraction of CLBlast GEMM's search space: e.g., for the four input sizes in Figure 2, valid configurations comprise $< 0,001\%$ of the unconstrained search space.

4 CONCLUSION

We present ATF – a highly generic framework for program auto-tuning that has several advantages as compared to the state-of-the-art approaches. ATF can auto-tune programs written in an arbitrary programming languages and belonging to an arbitrary application domain; tuning parameters are allowed to be interdependent. The user can auto-tune for an arbitrary objective (e.g., high runtime performance and/or low energy consumption) and choose among pre-implemented search techniques. We demonstrate that ATF is easy to use: the ATF user annotates the source code with simple tuning directives, rather than implements a tuning program, e.g., in Python or C++ – which makes auto-tuning appealing to common application developers. Our experimental results show that ATF provides significantly better tuning results — speedups of up to 17× — than the state-of-the-art approaches CLTune and OpenTuner for General Matrix Multiplication (GEMM) written in OpenCL on important input sizes as used in deep learning.

A detailed explanation of ATF is provided in [1].

REFERENCES

[1] A. Rasch, and S. Gorlatch. 2018. ATF: A Generic, Directive-Based Auto-Tuning Framework. *Concurrency and Computation: Practice and Experience* (2018), 0–15.

[2] C. Nugteren. 2017. CLBlast: A Tuned OpenCL BLAS Library. *arXiv preprint arXiv:1705.05249* (2017), 0–7.

[3] C. Nugteren et al. 2015. CLTune: A Generic Auto-Tuner for OpenCL Kernels. In *Embedded Multicore/Many-core Systems-on-Chip (MCSoC)*. IEEE, 195–202.

[4] J. Ansel et al. 2014. OpenTuner: An Extensible Framework for Program Autotuning. In *Int. Conf. on Parallel Architectures and Compilation*. ACM, 303–316.

[5] M. Christen et al. 2011. PATUS: A Code Generation and Autotuning Framework For Parallel Iterative Stencil Computations on Modern Microarchitectures. In *Parallel & Distributed Processing Symposium (IPDPS), 2011 IEEE International*. IEEE, 676–687.

[6] Netlib. 2017. BLAS. (2017). netlib.org/blas/

[7] OpenTuner. 2018. Interdependent Tuning Parameters (Issue 106). https://github.com/jansel/opentuner/issues/106. (2018).

[8] R. Whaley et al. 1998. Automatically Tuned Linear Algebra Software. In *Proc. of the 1998 ACM/IEEE Conf. on Supercomputing*. 1–27.

[9] Y. Jia et al. 2014. Caffe: Convolutional Architecture for Fast Feature Embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*. ACM, 675–678.