A Design of Experiments Approach to Autotuning under Tight Budget Constraints

Pedro Bruel*†, Arnaud Legrand*, Brice Videau* and Alfredo Goldman†
*University of Grenoble Alpes, CNRS, INRIA, LIG - Grenoble, France
Email: {arnaud.legrand, brice.videau}@imag.fr
†University of São Paulo - São Paulo, Brazil
Email: {phrb, gold}@ime.usp.br

Abstract—Abstract

I. Introduction

Optimizing code for objectives such as performance and power consumption is fundamental to the success and cost effectiveness of industrial and scientific endeavors in High Performance Computing. A considerable amount of highly specialized time and effort is spent in porting and optimizing code to GPUs, FPGAs and other hardware accelerators. Experts are also needed to leverage bleeding edge software improvements in compilers, languages, libraries and frameworks.

The automatic configuration and optimization of High Performance Computing applications, or autotuning, decreases the cost and time needed to adopt efficient hardware and software. Typical targets for autotuning include algorithm selection, source-to-source transformations and compiler configuration.

The autotuning of High Performance Computing applications can be studied as a search problem, where the objective is to minimize single or multiple software of hardware metrics. The exploration of the search spaces defined by configurations and optimizations present interesting challenges to search strategies. These search spaces grow exponentially with the number of considered configuration parameters and their possible values. They are also difficult to extensively explore due to the often prohibitive costs of hardware utilization and program compilation and execution times. Developing autotuning strategies capable of producing good optimizations using as few resources as possible is therefore essential. The capability of acquiring knowledge about an optimization problem is also a desired feature of an autotuning strategy, since it can decrease the cost of subsequent optimizations of the same application or for the same hardware.

It is common and usually effective to implement search meta-heuristics such as genetic algorithms and simulated annealing in autotuning systems, but these strategies usually attempt to exploit local properties of the search space and are not capable of fully exploiting global structures. These strategies are also not much more effective in comparison with a naive uniform random sample of the search space[1], [2], and usually rely on a large number of measurements and frequent restarts to achieve good performance improvements.

Search strategies based on gradient descent also are commonly used in autotuning and rely on a large number of measurements. Their effectiveness diminishes additionally in search spaces with complex local structures. Completely automated machine learning autotuning strategies are effective in building models for predicting important optimization parameters, but still rely on a sizable data set in order to train useful models.

Search strategies based on meta-heuristics, gradient descent and machine learning require a large number of measurements to be effective, and are usually incapable of providing knowledge about search spaces to users. It is impossible at the end of each autotuning session to decide whether further exploration is warranted and to know which parameters were responsible for the observed improvements.

II. Related Work

A. Source-to-source Transformation

B. Autotuning

C. Search Space Exploration Strategies

III. Design of Experiments

A. D-Optimal Designs

IV. Applying Design of Experiments to Autotuning

A. Experimental Methodology

B. Performance on a GPU Laplacian Kernel

V. Results on the SPAPT Benchmark

VI. Conclusion

Acknowledgment

References

[1] K. Seymour, H. You, and J. Dongarra, "A comparison of search heuristics for empirical code optimization." in CLUSTER, 2008, pp. 421–429.

[2] P. M. Knijnenburg, T. Kisuki, and M. F. O'Boyle, "Combined selection of tile sizes and unroll factors using iterative compilation," The Journal of Supercomputing, vol. 24, no. 1, pp. 43–67, 2003.