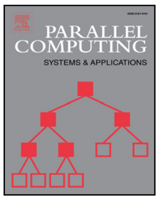
Contents lists available at ScienceDirect
Parallel Computing
journal homepage: www.elsevier.com/locate/parco



Multi-Objective region-Aware optimization of parallel programs

Juan J. Durillo[a], Philipp Gschwandtner[b,*], Klaus Kofler[b], Thomas Fahringer[b]

[a] Leibniz Supercomputing Centre, Boltzmannstrasse 1, 85748 Garching, Germany
[b] University of Innsbruck, Technikerstrasse 21a, Innsbruck 6020, Austria

A R T I C L E   I N F O

Article history:
Received 30 November 2017
Revised 18 March 2018
Accepted 30 March 2018
Available online xxx

Keywords:
Auto-tuning
Code region
Multi-objective
OpenMP
Parallel program
Energy

A B S T R A C T

Auto-tuning has become increasingly popular for optimizing non-functional parameters of parallel programs. The typically large search space requires sophisticated techniques to find well-performing parameter values in a reasonable amount of time. Different parts of a program often perform best with different parameter values. We therefore subdivide programs into several regions, and try to optimize the parameter values for each of these regions separately as opposed to setting the parameter values globally for the entire program. In order to manage this enlarged search space, we have to extend existing auto-tuning techniques to ensure high quality solutions to this optimization problem. In this paper we introduce a novel enhancement to the RS-GDE3 algorithm that is used to explore the search space for auto-tuning programs with multiple regions regarding several objectives. We have implemented our auto-tuner using the Insieme compiler and runtime system, and provide a detailed analysis of the obtained results with the aim of gaining a better understanding of non-functional inter-region behavior in the context of auto-tuning. In comparison to a non-optimized parallel version of the tested programs, our novel approach achieves improvements of up to 7.6X, 10.5X, and 61.6X for three tuned objectives wall time, energy consumption, and resource usage, respectively.

1. Introduction

Optimizing parallel programs becomes increasingly difficult with the rising complexity of modern parallel architectures, which include, for example, a dramatic increase in the number of cores per chip or the availability of multi-level, partially shared cache hierarchies. Automatic software tuning, or simply *auto-tuning* [1], arose as an attempt to take better advantage of hardware features by automatically tuning applications. An auto-tuner tries to find promising *configurations* for a given program executing on a given target platform. A configuration consists of a set of non-functional parameters with a corresponding set of values. A configuration can influence a program's performance by transforming the source code of the application, or by finding optimal values for the parameters that govern the execution of that program on a given architecture (e.g., number of threads, frequency per core).

This paper describes a novel auto-tuning approach for programs with multiple single-entry single-exit code regions, whose non-functional behavior depends on at least one tunable parameter. We assume that we can measure the non-

* Corresponding author.
E-mail addresses: juan.durillobarrionuevo@lrz.de (J.J. Durillo), philipp@dps.uibk.ac.at, philipp.gschwandtner@uibk.ac.at (P. Gschwandtner), klaus@dps.uibk.ac.at (K. Kofler), tf@dps.uibk.ac.at (T. Fahringer).

https://doi.org/10.1016/j.parco.2018.03.010
0167-8191/© 2018 Elsevier B.V. All rights reserved.

functional behavior of these regions for the optimization objectives such as wall time or energy consumption. Tuning multi-region applications exposes additional challenges for auto-tuning techniques. First, there is usually no single set of parameter values that is optimal for all the regions of a program. However, setting the parameter values individually for every region leads to a very large search space that grows exponentially with the number of tuning opportunities, i.e. the number of regions. Second, the execution of a region may be influenced by the parameter values applied to neighboring regions. Previous work [2] observed that the optimal parameter values for individual regions of hybrid MPI/OpenMP applications can lead to sub-optimal overall performance.

Auto-tuning techniques are widely used [3–11] but are often limited to using the same parameter values for every region, i.e. globally for the entire program, ignoring the fact that different parts of the code may benefit from specific parameter values.

Our auto-tuner can optimize a generic set of objectives that do not necessarily correlate with each other. This lack of correlation makes it impossible to find a single solution that is best in every objective. For example, if some regions are optimized for one objective while others are optimized for another objective, it is very likely that the overall performance of the program will suffer. A configuration may exhibit short wall time at the cost of high resource usage for one region, but the contrary for another region, resulting in sub-optimal overall performance.

The approach proposed in this paper extends the method presented by Jordan et al. [12], which is limited to optimizing each region in isolation by adding region-aware auto-tuning support for the entire program. The contributions of this work are the following: (1) A region-aware multi-objective auto-tuner; (2) a compiler-runtime system that automatically identifies regions and enables automatic tuning of their parameters; (3) and an evaluation and comparison of several global and region-aware auto-tuning strategies for several codes on different target architectures, demonstrating the importance of region-aware auto-tuning compared against global optimization. These issues were already discussed in a previous version of this paper [13]. The current manuscript extends the content of the previous paper with: (1) Thorough analysis and discussion on how the regions composing a program behave within different (quasi-) Pareto optimal program configurations; (2) and a study of the computed trade-off in terms of the three considered criteria.

The rest of this paper is organized as follows: Section 2 exemplarily shows the inherent advantages of region-aware auto-tuning while Section 3 gives detailed insights on the auto-tuning approach presented in this paper. The implementation details of the our work can be found in Section 4, while Section 5 discusses the metrics that we use to compare the quality of the different auto-tuning approaches. The results of the experiments performed in this paper are shown in Section 6 including a comparison among different multi-objective region-aware tuners. Further analysis of the solutions computed by the best approach in the preceding comparison is included in Section 7. Section 8 gives an overview of the related work and the conclusion of our findings is summarized in Section 9.

2. Motivation

In this section, we motivate the need for region-aware auto-tuners by using a simple example program consisting of two regions. Both regions perform a parallel matrix multiplication. In the first region, only the outermost loop is parallelized; in the second, we parallelize only the innermost loop. As a consequence, we have two regions with different execution behavior: the first one scales well with the number of threads whereas the second one does not. In order to have similar wall times for both regions, the matrix size in the second region is only one quarter of the matrix size in the first region.

The experiments in this section are performed on the Ivy Bridge-EP architecture described in Section 6. Our goal is to find the optimal configuration for executing this program. For the sake of feasibly performing an exhaustive search of all possible program configurations, we assume that these regions only expose the number of threads as a tunable parameter.

We select configurations for the previously described program using three different approaches:

- Isolated: This approach optimizes both regions in isolation.
- Global: This approach is constrained to finding a single set of parameter values to be used in both regions.
- Region-Aware: This approach optimizes both regions using individual parameter values for each of them to maximize the program's overall performance.

The best configurations found by these approaches are summarized in Table 1. The first two rows of the table show the number of threads for each region chosen by the corresponding approach — depicted in different columns. The following two rows list the wall time for each of these regions with the aforementioned indicated number of threads. The fifth row shows the program's wall time when the regions are configured as indicated in the first two rows. Finally, the last row shows the relative difference to the best wall time found across all three approaches.

The worst wall time corresponds to the Isolated approach. The reason is that it will run the first region with a large number of threads, since that region scales well. However, this has a negative effect on the second region, which does not scale well. Therefore, this region will be executed with a small number of threads. The change from 20 to only a few threads between the regions introduces a significant overhead due to the different number of sockets required for both regions and data residing in caches of the socket not in use for the second region. The fastest option for the second region is to use two threads, taking 5798 ms. For example, using 7 threads, which is the fastest option after the first region has been executed with 10 threads, results in a wall time of 6344 ms in this scenario.