Proceedings of 1st Workshop on Resource Awareness and Application Autotuning in Adaptive and Heterogeneous Computing (RES4ANT) 2016

Evaluation of Different Autotuning Methodologies on The Compiler Optimization Problem

Amir Hossein Ashouri, Gianluca Palermo and Cristina Silvano

Politecnico Di Milano, ITALY
{name.familyname}@polimi.it

***Abstract*—Diversity of today's architectures have forced programmers and compiler researchers to port their application across many different platforms. Compiler auto-tuning itself plays a major role within that process as it has certain levels of complexities that simply the standard pre-defined optimization levels fail to bring the best results due to their average performance output. To address the problem, different optimization techniques has been used for traversing, pruning the huge space, adaptability and portability. In this short paper, we propose our different approaches including the use of Design Space Exploration (DSE) techniques and Machine Learning to further tackle the both problems of selection and the phase-ordering of the compiler optimizations. It has been demonstrated and assessed that utilizing these techniques have positive effects on the performance metrics of the given applications and can bring up to 60% performance improvement with respect to standard optimization levels (e.g. -O2 and -O3) on the selection problem and up to 4% w.r.t. to LLVM standard optimization on the phase-ordering problem.**

***Keywords*—Compiler Optimization; Machine Learning; Autotuning; Phase-ordering;**

I. INTRODUCTION

Conventional software applications are first developed in the desired high-level source-code (e.g. C, C++) and then are passed through the compilation phase to build the executable. The later phase includes *compiler optimization* process in which the target metrics such as *execution time, code-size, power*, etc are optimized depending on the desired scenario. *Compiler optimizations* are playing an important role to transform the source-code to an *optimized* variation. Usually, open-source/industrial compiler platforms are coming off-the-shelf with some standard optimization levels (e.g. -O1, -O2, -O3 or -Os) to bring the average-good results for conventional platforms. However, quite often they fail to bring the optimal results for specific applications, architectures and platforms. In the short paper, two different techniques for compiler auto-tuning , namely, *DSE* and Machine Learning based techniques have been proposed to accommodate and address the problem of selecting the best compiler optimization for a given application.

II. APPROACH (I): DSE APPROACH

DSE refers to the activity of exploring the design parameters alternatives before the actual design. It deals with pruning and exploring the design space efficiently. The proposed work targets the exploration of compiler options parameters, in order to automatically explore the design space and analyze the compiler-architecture co-design. As per evaluation platform, we assessed the proposed methodology in Very-long-Instruction-Word (VLIW) architecture as a promising embedded systems processor by applying random design of experiment algorithm and tackle the aforementioned problem by proposing an automatic methodology based on a tool-chain including our Multi-Objective System Tuner tool (MOST), a wrapper, an open-source compiler and a compiler/simulator; namely, LLVM and VLIW-EXample (VEX). The proposed tool-chain enables the designer to automatically explore, optimize and analyze the options by using several standard benchmarks for both high-end embedded and signal processing applications [1]. Being focused more on the analysis, we showed that the adoption of the specific methodology either in a cross-architecture and/or cross-application manner, can deliver significant application specific insights thus enabling the designer to guide through decisions regarding the architecture and the compilation optimization strategy [2]. Figure 1 represents the proposed methodology for compiler co-exploration with DSE techniques. The work-flow starts on by inferring the *pareto-optimal* architectural design space and then it feeds the found architectural properties to the compiler framework. Statistical analyses will be applied at the end to assess the correlation between utilizing the certain compiler options and the observed performance metrics. Figure 2 is showing different distributions derived by applying the proposed DSE methodology that reveals the effect of utilizing certain compiler optimizations on the performance metric.

III. APPROACH (II): MACHINE LEARNING APPROACH

Diversity in applications and architecture, simply makes it barely possible to manually optimize and port the source-codes for each application/architecture. *Random Iterative Compilation* fails to efficiently bring the optimal results due to its high demand on time and number of iterations. In order to improve the portability of compiler optimization with respect to the handcrafted approaches, machine learning has been used to address both the *selection of compiler optimization options* and

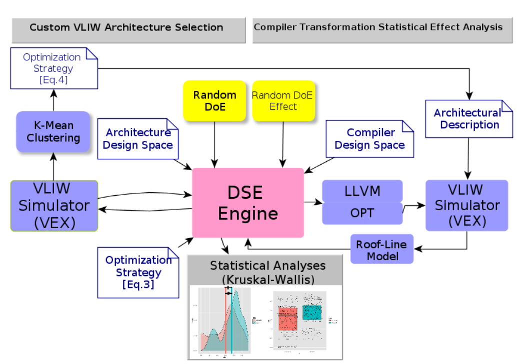
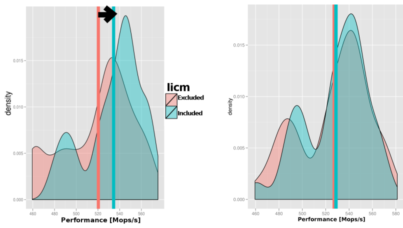


Fig. 1: Approach (I): DSE Proposed Methodology



(a) (b)

Fig. 2: Visualization of (a) *licm*'s significant positive effect, (b) *reassociate*'s no significant effect

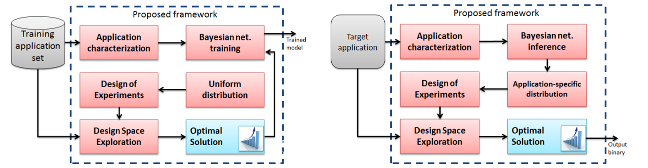
phase-ordering problem [3] to predict the right optimization to be applied given an unseen application [4].

A. The Selection Problem

Addressing the issue on the second approach, we propose a Machine Learning based autotuning framework that maximizes the performance of a target application. The proposed work starts by applying statistical methodology with Bayesian Networks to infer the probability distribution of the compiler optimizations to be enabled to achieve the best performance. We start to drive the iterative compilation process by sampling from the probability distribution. Likewise most machine learning approaches, here we use a couple of sets of training applications to learn the statistical relations between application features and the compiler optimizations. Given a new unseen application, its features are fed into the machine leaning algorithm as *evidence* on the distribution. This evidence imposes a bias on the distribution. Since compiler optimizations are correlated with the software features, we can redo the process of sampling for the new target application. Figure 3 demonstrates the second proposed approach that is assessed on an embedded *ARM* device with GCC compiler. The obtained probability distribution is indeed application-specific and effectively exploits the use of iterative compilation process as it only drives with the most promising compiler optimizations [5]. Figure 4, represents the result of our proposed ML algorithm w.r.t standard optimization levels -O2 and -O3 on *cBench* suite. It represents significant speedup factor over the majority of the evaluated applications with the average 56% and 47% improvement on respectively -O2 and -O3. Utilizing the proposed machine learning approach led to reach a factor of 3× exploration speedup when it comes to compare with the *random iterative compilation* having a fixed number of extraction.

B. The Phase-ordering Problem

On the other side, when order of the compiler optimizations are considered, the so-called *phase-ordering* problem should be tackled. The space gets enormously bigger and simple classic supervised techniques are not able to come up with a quality models for prediction. Addressing the phase-ordering problem, we propose an immediate speedup predictor that is able of predicting the next-best optimization to be applied given the current status of the code. We utilized predictive models and dynamic software characterization to construct the application specific models. In order to speedup the exploration on the space, we defined two traversing heuristics that use DFS and exhaustive search within the prediction space and introduced up to4% speedup w.r.t LLVM default compilation. Readers can refer to [3] for further details.



(a) (b)

Fig. 3: Approach (II): Overview of the proposed M.L Methodology a) training phase b) predicting phase

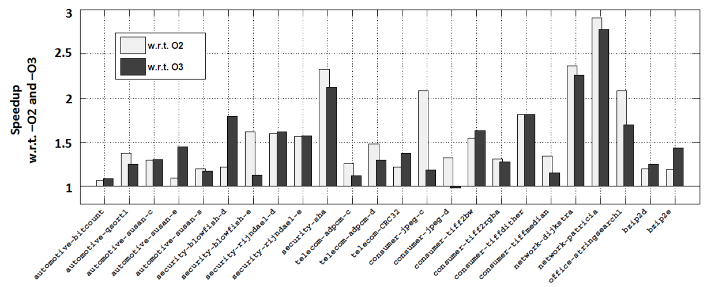


Fig. 4: Performance improvement of our Bayesian Networks w.r.t -O2 and -O3

IV. Conclusion and Future Work

This short paper presents two main different approaches on the compiler autotuning problem using DSE techniques and machine learning. The assessments demonstrate positive speedup on the performance metrics while classifying the effective compiler optimizations derived by the methodology in DSE approach and 40%-60% speedup with respect to GCC -O2 and -O3 on an ARM embedded-board. Future work will be focused on more fine-grain analyses on the dependability of the input data with the compiler optimization options and applying more sophisticated machine learning techniques to the phase-ordering problem of the compiler optimization.

References

[1] A. H. Ashouri, "Design space exploration methodology for compiler parameters in vliw processors," 2012, http://hdl.handle.net/10589/72083.

[2] A. H. Ashouri, V. Zaccaria, S. Xydis, G. Palermo, and C. Silvano, "A framework for compiler level statistical analysis over customized vliw architecture," in *Very Large Scale Integration (VLSI-SoC), 2013 IFIP/IEEE 21st International Conference on.* IEEE, 2013, pp. 124–129.

[3] A. H. Ashouri, A. Bignoli, G. Palermo, and C. Silvano, "Predictive modeling methodology for compiler phase-ordering," in *Proceedings of 7th Workshop on Parallel Programming and Run-Time Management Techniques for Many-core Architectures and 5th Workshop on Design Tools and Architectures for Multicore Embedded Computing Platforms*. ACM, 2016.

[4] F. Agakov, E. Bonilla, J. Cavazos, B. Franke, G. Fursin, M. F. O'Boyle, J. Thomson, M. Toussaint, and C. K. Williams, "Using machine learning to focus iterative optimization," in *Proceedings of the International Symposium on Code Generation and Optimization.* IEEE Computer Society, 2006, pp. 295–305.

[5] A. H. Ashouri, G. Mariani, G. Palermo, and C. Silvano, "A bayesian network approach for compiler auto-tuning for embedded processors," in *Embedded Systems for Real-time Multimedia (ESTIMedia), 2014 IEEE 12th Symposium on.* IEEE, 2014, pp. 90–97.