Auto-tuning Techniques for Compiler Optimization

Amir Hossein Ashouri, Gianluca Palermo and Cristina Silvano

DEIB - Politecnico di Milano, ITALY

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 address the problem. 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).

Categories and Subject Descriptors

[**DATE PhD Forum**]: Compiler Optimization

Keywords

Compiler Auto-tuning, Machine Learning, DSE

1. 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 optimization options* 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 have been proposed to accommodate and address the problem of selecting the best compiler optimization for a given application.

2. 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 target is a multi-objective optimization problem: a) to maximize the performance of the platform and b) to minimize the power consumption or other non-functional metrics. 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 and two open-source compilers; 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 [2].

Furthermore, adding the architectural properties to the problem, the proposed DSE work is dealing with efficiently co-exploring the design-space by first generating a set of promising architecture candidate points that tailors to the characteristics of the target application. Then, optimizing on the performance-intensity trade-off curve with respect to the overall hardware allocated resource. Besides, the methodology will assess the quality of the candidates with statistical analysis of distributions generated over the compiler transformation space performed on the set of these selected customized VLIW solutions. This enables the designer to characterize the effects of each compiler transformation in both an architecture specific manner and a cross-architecture manner. The Roof-Line performance model [5] has been used as the basis for both generating the custom architecture configurations and characterizing the effect of the compiler passes and relates processor performance to off-chip memory traffic. 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 [4].

Figure 1 represents the proposed methodology for compiler co-exploration with DSE techniques. The work-flow starts on the left-side of the figure by inferring the *pareto-optimal* architectural design space and then it feeds the found architectural properties to the compiler framework on the right-side. Statistical analyses will be applied at the end to assess the correlation between utilizing the certain compiler options and the observed performance metrics.

Table 1 representing the assessment of the utilization of

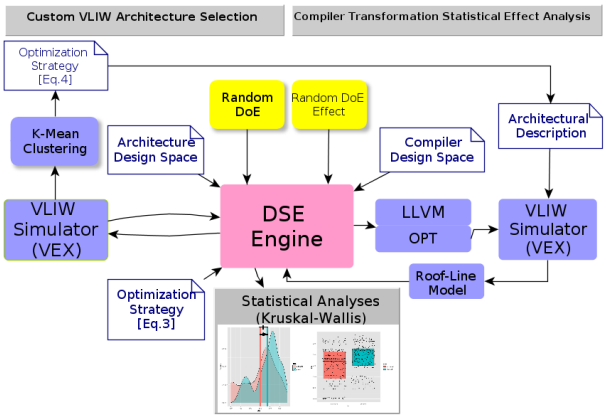


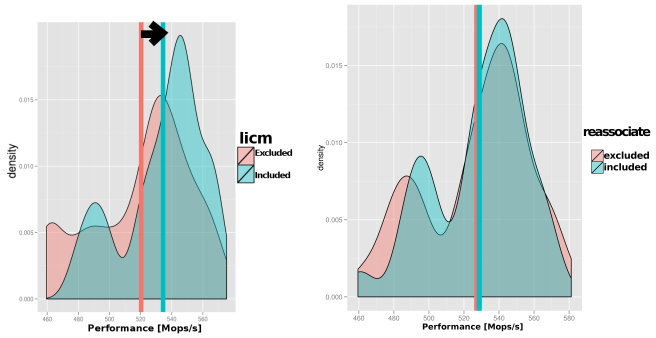
Figure 1: DSE Proposed Methodology

DATE 2016, PhD Forum, Dresden, GERMANY
Copyright is held by the author/owner(s).

certain compiler options on the target applications with the *Kruskal-Wallis* test. As observed, not all the utilized compiler flags are effective in changing the performance metrics on a target application. Figure 2 is showing different distributions derived by applying the proposed DSE methodology that reveals the effect of utilizing the certain compiler option on the performance metric.

Option	GSM	AES	ADPCM	JPEG	Blowfish
Constprop	-	-	-	-	-
Dce	-	-	-	-	-
Inline	√	-	√	√	-
Instcombine	√	-	√	√	√
Licm	√	√	√	√	√
LoopReduce	√	√	√	√	√
LoopRotate	√	√	√	√	-
LoopUnroll	-	-	-	-	-
LoopUnswitch	-	-	-	-	-
Mem2reg	√	√	√	√	√
Memcpyopt	-	-	-	-	-
Reassociate	√	-	-	-	-
Scalarrepl	√	-	-	-	√
Sccp	-	-	-	-	-
Simplyfycfg	-	-	-	-	-

Table 1: Kruskal-Wallis Test on Performance



(a) (b)

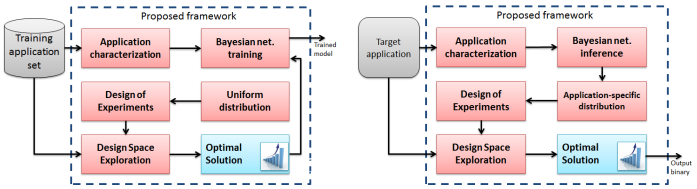
Figure 2: Visualization of (a) *licm*'s significant positive effect, (b) *reassociate*'s no significant effect

3. 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 the selection of compiler optimization options and to predict the right optimization to be applied given an unseen application [1].

We propose an innovative approach to tackle the problem of identification of compiler optimization that maximize the performance of a target application. The proposed work starts by applying statistical methodology to infer the probability distribution of the compiler optimization 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. Bayesian Networks are used for the statistical model. Given a new unseen application, its features are fed in to the machine leaning algorithm as *evidence* on the distribution. This evidence imposes a bias on the distribution and 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 proposed approach we have proposed and 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 [3].

Figure 4, is the result of our proposed ML algorithm w.r.t standard optimization levels -O2 and -O3 on *cBench* suite.



(a) (b)

Figure 3: Overview of the proposed M.L Methodology: a) training phase b) predicting phase

It clearly represents significant speedup factor over majority of the evaluated applications with average 56% and 47% improvement on respectively -O2 and -O3.

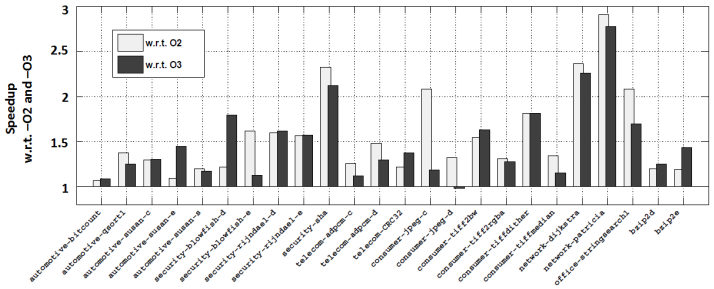


Figure 4: Performance improvement of our Bayesian Networks w.r.t -O2 and -O3

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. Readers can refer to [3] for further details.

4. CONCLUSION AND FUTURE WORK

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

5. REFERENCES

[1] Felix Agakov, Edwin Bonilla, John Cavazos, Björn Franke, Grigori Fursin, Michael FP O'Boyle, John Thomson, Marc Toussaint, and Christopher KI Williams. Using machine learning to focus iterative optimization. In *Proceedings of the International Symposium on Code Generation and Optimization*, pages 295–305. IEEE Computer Society, 2006.

[2] Amir Hossein Ashouri. Design space exploration methodology for compiler parameters in vliw processors, 2012. `http://hdl.handle.net/10589/72083`.

[3] Amir Hossein Ashouri, Giovanni Mariani, Gianluca Palermo, and Cristina 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*, pages 90–97. IEEE, 2014.

[4] Amir Hossein Ashouri, Vittorio Zaccaria, Sotirios Xydis, Gianluca Palermo, and Cristina 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*, pages 124–129. IEEE, 2013.

[5] Samuel Williams, Andrew Waterman, and David Patterson. Roofline: an insightful visual performance model for multicore architectures. *Communications of the ACM*, 52(4):65–76, 2009.