CIE 6020 Information Theory (Spring, 2018)

Course Project: Profile-Guided Source Coding in Dual-Compilation

Name: Chenhao Wu ID Number: 117010285

Profile-Guided Source Coding in Dual-Compilation

Chenhao Wu

March, 2019

1 Introduction

Within this decade, the computing performance of mobiles and embedded systems have been significantly raised such that more computing workloads can be undertaken on mobiles and embedded systems, such as digital image processing, machine learning and deep learning applications, augmented-reality applications, and so on. In practice, one of the critical bottlenecks in mobiles and embedded systems design and optimization is the run-time memory space and power consumption. Since the complexity of program and data set grew much faster than the performance of memory capacity in the mobile and embedded system, rather than merely use expensive computing components to feed the needs of applications with high computing overhead, it is more promising to find ways to best utilize the system use of memory space. To realize this objective, we proposed a profile-guided source coding (PGSC), which will perform an adaptive compiler-guided data compression on memory space during run-time according to different applications and run-time environment.

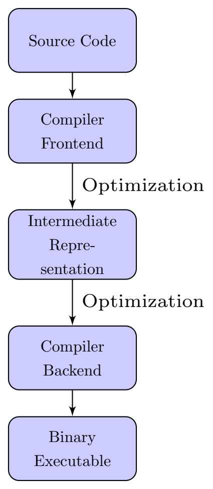

Figure 1: Traditional Compilation Procedure

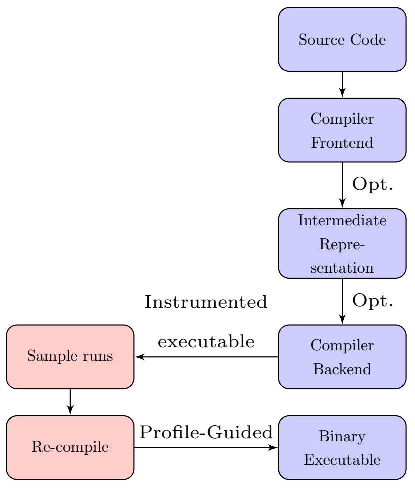

Figure 2: Compilation Procedure with PGO

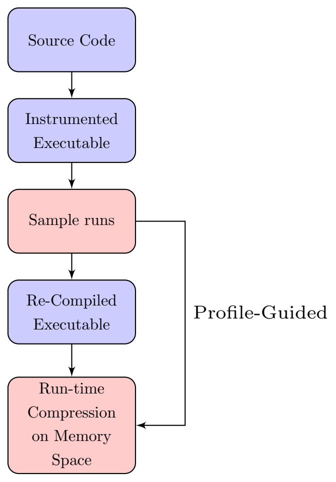

Figure 3: Profile-Guided Source Coding

The idea of this approach comes from a modern compilation technique, Profiled-Guided Optimization (PGO), that as shown in Figure 1 and Figure 2, differed from the traditional approach [1], the compiler will perform a recompilation based on initial executable and sample runs. A profile, for instance, breaks the source code into blocks and devises a table of the frequency each block is executed via sample runs, and during recompilation, code blocks will be reordered with respect to the frequency table. For example, blocks with higher frequency will be allocated in memory area with lower memory address and blocks with lower frequency will be allocated in higher address so that the additional I/O overhead consumed in function callbacks will be reduced. Similar to previous compiler optimization methods, like the deletion of unreachable code, the constant folding optimization and propagation optimization, the objective of PGO is to further reduce the redundancy of unnecessary and/or unimportant logic branches in the source program.

The usage of this approach, however, can be extended into attempts in reducing the memory space, that after sample runs, not only the frequency of code blocks can be collected, but also the preliminary investigation of frequency that each block of memory is accessed can be gathered as well. As shown in Figure 3, the procedure of PGSC consists of a dual-stage compilation, that the compiler will initially compile the source code into an instrumented executable, and based on a profile generated after sample runs, the compiler will re-compile the source code such that during the run-time, the program will compress specific areas of memory space in order to reduce the redundancy of memory use, and will instantaneously decompress the compressed contents whenever the program needs.

In the tide that more and more applications on mobile and embedded systems behave to buffer long arrays of audio, images and video frames into the memory space during run-time, that in a low frequency the program access to them but they still need to be remained in the memory space, such as intermediate training matrices, sample sets of images and so on, PGSC can regularly compress them to reduce the buffer size. The compression is compiler-guided, thus it can be performed no matter which optimization has been applied onto the program and what third-party library the program has imported.

In this report we will briefly introduce the control flow of PGSC, and methodologies we applied to perform the compression over memory space, and the implementation of a compiler with PGSC.

2 Related Work

[2] [3] [4] [5]

References

[1] J.D. Ullman A. Aho R. Sethi. *Compiler: Principles, Techniques, and Tools.* Addison-Wesley, 1986.

[2] "Compiled Code Compression for Embedded Systems using Evolutionary Computing". In: ().

[3] Mary Jan Irwin Ozcan Ozturk Mahmut Kandemir. "Using Data Compression for Increasing Memory System Utilization". In: (2009), pp. 901–914.

[4] Takeshi Ohkawa Takashi Yokota Kohta Shigenobu Kanemitsu Ootsu. "A Translation Method of ARM Machine Code to LLVM-IR for Binary Code Parallelization and Optimization". In: (2017), pp. 575–579.

[5] Sharmila Chidaravalli Manjunath T.K. "Static Analysis of Designing A Compiler Tool That Generates Machine Codes for The Predicted Execution Path: A Bypass Compiler". In: (2011), pp. 418–422.