Profile Guided Source Coding in Compiler Optimization

An Approach to Reduce The Run-time Memory Occupancy

Chenhao Wu

The Chinese University of Hong Kong, Shenzhen
School of Science and Engineering

chenhaowu@link.cuhk.edu.cn

香 港 中 文 大 學（深 圳）
The Chinese University of Hong Kong, Shenzhen

深圳市物联网智能系统与无线网络技术
重点实验室
Shenzhen Key Laboratory of IoT Intelligent Systems and WirelessNetwork Technology

Abstract

In order to best utilize the run-time memory occupancy and to give powerful and intensive memory-consumption applications with higher compatibility on mobile and embedded systems, we proposed a compiler optimization, Profile Guided Source Coding (PGSC), which is led by the compilation process and aims to reduce the run-time memory occupancy of compiled programs. The optimization of PGSC is based on a profile generated by sample runs on the initially compiled executable, and the profile entails the access (load/read) frequencies on the allocated memory blocks performed by the program during run-time. Our strategy is during the compilation the memory accessing instructions of the program will be substituted by compress and decompress instructions, such that the memory blocks with lower frequency will be compressed into a compressed area when idled and will be decompressed into a run-time area when they are to be loaded. To achieve this objective we apply the Lempel-Ziv coding on the compression process and artificially selected several sample programs as the benchmark to examine the performance of programs compiled with PGSC optimization. The experiment shows that after applying PGSC optimization the average memory occupancy (AMO) and the maximum memory occupancy (MMO) of the identical program are decreased. Even though it would also bring an additional overhead in distributing the computing resources in run-time compression and decompression, the experiment shows this optimization is considerable in practice. In the future, we will research on the combination of the PGSC optimization and the distributed cache system, and also the investigation on the capability of combining PGSC optimization with contemporary channel coding schemes.

Introduction

In the past two decades, the computing performance of mobile 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 & deep learning algorithms, augmented-reality applications, and so on. In practice, one of the critical bottlenecks in mobiles and embedded systems design and optimizations is the run-time memory space and power consumption. Since the complexity of program and data set grew much faster than the amelioration of hardware, rather than using expensive electronic components to fulfill the increasing demand of computation resource, it is more promising to find approaches to best utilize the system use of memory space. To achieve this objective, we proposed a Profile Guided Source Coding (PGSC) optimization that will perform an adaptive data compression on allocated memory space during run-time according to different applications and run-time environment.

Profile Guided Compilation

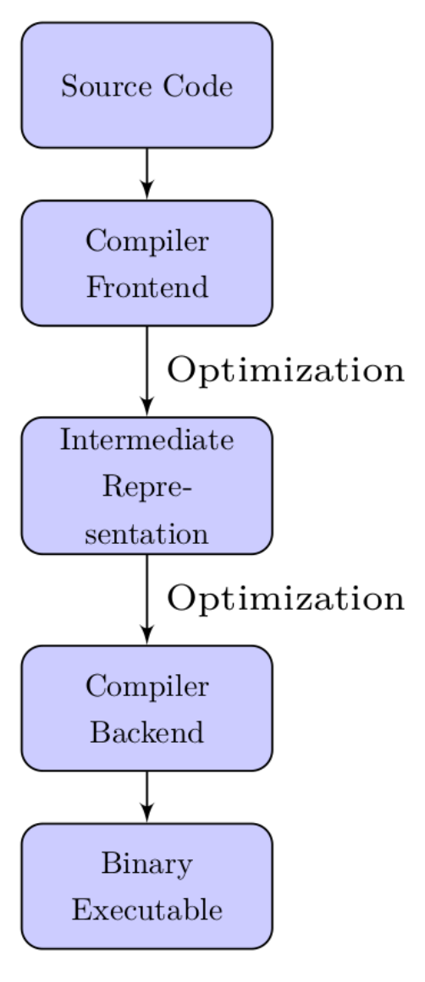

Figure 1: Traditional Compilation Procedure

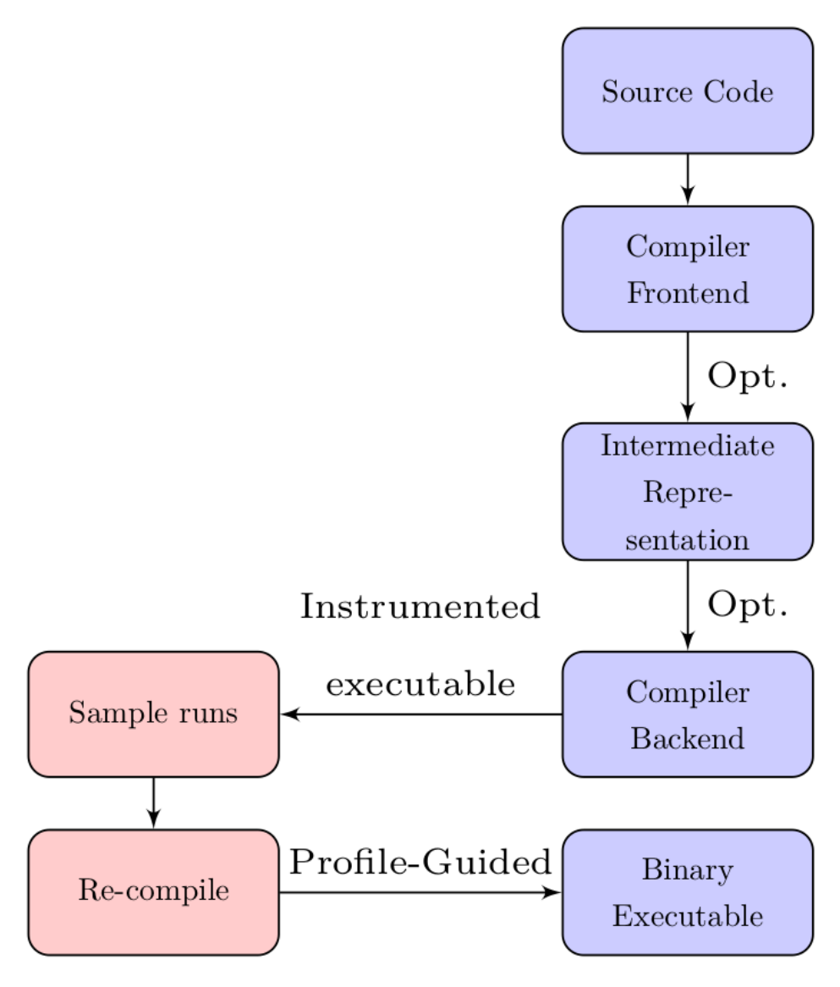

Figure 2: Compilation Procedure with PGO

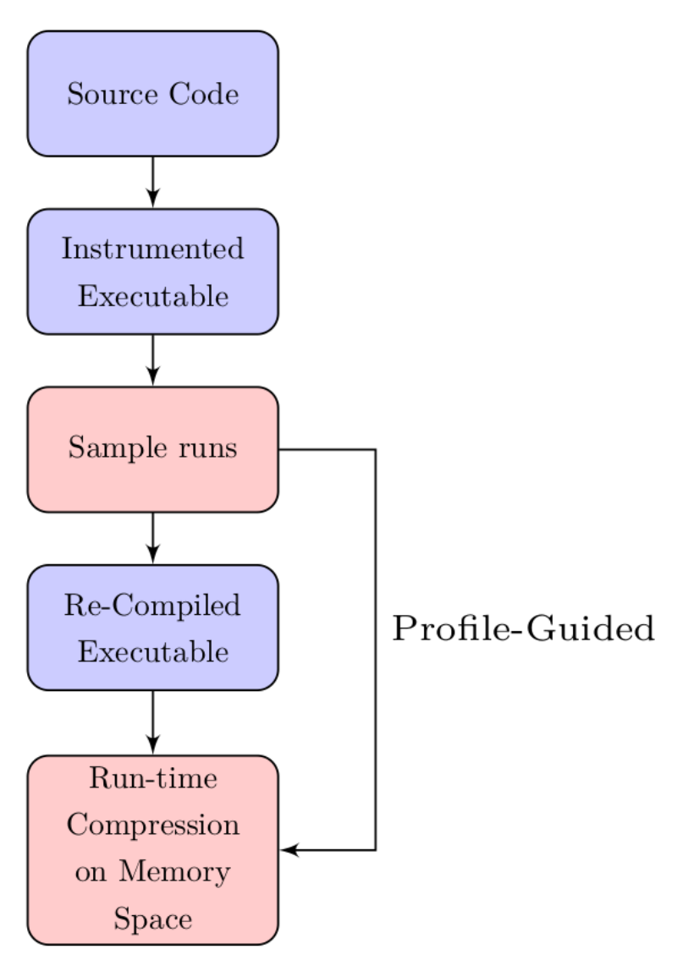

Figure 3: Profile-Guided Source Coding

Inspired by Profile Guided Optimization, the optimization of PGSC requires profiling the compiled executable so as to amend the programs adapting to the run-time behaviors. As shown in Figure 1 and Figure 2, differed from the traditional compilation process, the compiler with profile guided optimization performs a recompilation based on a profile generated by sample runs on initial executable, and applies optimizations according to the profile.

Workflow of PGSC

1. Apply the generic compiler to guide the initial compilation and transform the source code into binary executable
2. Analyze sample runs of the program, profiling the heap access frequency
3. Pass the profile to compiler and rewrite instruction according to the profile

Run-time Compression & Decompression

In the PGSC scheme during run-time the memory space will be separated into two areas, run-time area and compressed area.

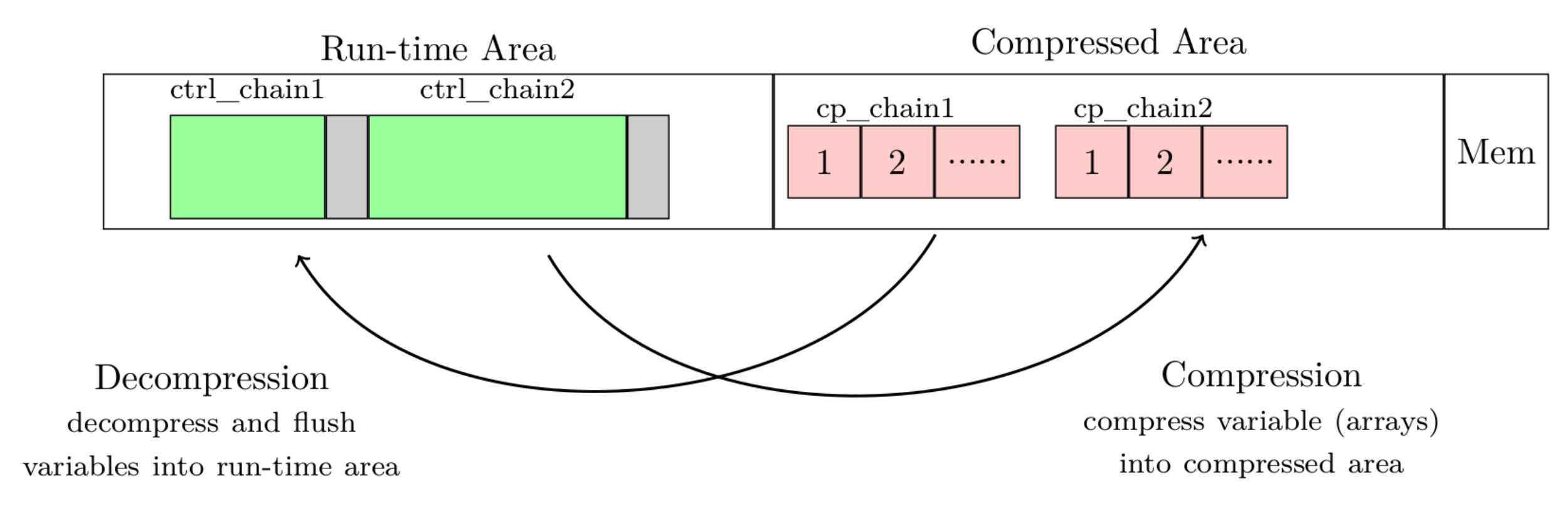

Figure 6: Run-time Compression & Decompression

In the compilation, the compiler specifies the areas of data to be compressed during idle time, and also generates codeword for each block so as during run-time the program would automatically compress them from the run-time area into the compressed area. Whenever the compressed data are to be loaded, the program will instantaneously decompress the data from the compressed area, and flush into the run-time area.

Experiments & Results

To examine the performance of programs compiled with PGSC optimization, we apply several testing programs to present a comparison between programs compiled with PGSC and programs without PGSC.

Testing Environment

- **Operating System**: Debian 9 x86-64 virtual machine
- **RAM**: 512 MB
- **Number of Processors**: 1
- **Clock Frequency**: 2.8 GHz
- **Generic Compiler**: clang/clang++

Results

Program	Functionality	MMO (clang)	AMO (clang)	MMO (PGSC)	AMO (PGSC)	Extra Terminating Time
LU-Dec	Solve a set of linear eqs. using LU decomposition	214.3 Mb	165.7 Mb	153.7 Mb	53.9 Mb	+47%
Face-Rec	Learn new faces captured from webcam	470.0 Mb	365.2 Mb	217.8 Mb	93.7 Mb	+239%
LZO	Compress sample text files using LZO algorithms	253.7 Mb	103.8 Mb	217.8 Mb	93.7 Mb	+65%
Huff-Cod	Compress sample text files using Huffman Coding	324.1 Mb	214.7 Mb	293.7 Mb	196.5 Mb	+53%

From the result of experiments, both the maximum memory occupancy and the average memory occupancy are shown to decrease after applying PGSC optimization. For the LU Decomposition program and the Facial Recognition program the decrease rate is more remarkable than the rest two testing programs. Due to the basic algorithms the former two programs tend to store lots of intermediate matrices or intermediate picture data on the heap. Thus, the experiment result also implies the run-time compression can achieve better optimization for programs that store more intermediate and redundant data on the heap.

Even though the extra compression time and terminating time have been increased after applied PGSC in the compilation, the overhead is still within an acceptable range. Hence, applying run-time compression as a optimization is considerable for programs.

Forthcoming Research

The current process of PGSC we proposed are still being performed on a single computer, as well as all the experiments involved. However, it is still feasible to apply such a compiler-directed technique to a distributed system, because during run-time the memory space has been virtually divided into two partitions after PGSC optimization. Hence, our future research will focus on applying this technique to distributed systems, and also to investigate the capability of combining PGSC optimization with contemporary channel coding schemes. In addition, we will also take efforts to enhance the computational efficiency of PGSC in order to decrease the additional overhead.

References

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

[2] M S Ali. Compiled code compression for embedded systems using evolutionary computing. pages 1173–1174, 2008.

[3] Mary Jan Irwin Ozcan Ozturk, Mahmut Kandemir. Using data compression for increasing memory system utilization. pages 901–914, 2009.

[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. pages 575–579, 2017.

[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. pages 418–422, 2011.