x86 instruction reordering and split-stream compression benchmark

Zsombor Paróczi

Binary executables are the result of the compilation process, which produces machine code, initialized and uninitialized data and operation system specific headers from source code.

Runtime executable compression is a method which uses standard data compression methods and binary machine code transformations to achieve smaller file size, yet maintaining the ability to execute the compressed file as a regular executable. In our work we only focus on Intel x86 instruction set with 32bit registers, which mainly used in personal computers. In this paper we perform a benchmark on different compressions using various binary transformations, the main contributribution to the field is (i) testing Zopfli on binaries and (ii) benchmark the instruction reordering method combined with split-stream algorithm.

Method	Based on	Homepage
Aplib	LZ-based	http://www.ibsensoftware.com/
Gzip	deflate (LZ + Huffman)	http://www.gzip.org/
LZMA	Lempel–Ziv–Markov chain algorithm	http://www.7-zip.org/sdk.html
Zopfli	deflate based	https://code.google.com/p/zopfli/

Table 1: Compression methods

For our benchmark we selected four different compression methods, they can be seen in Table 1. Besides of the common algorithms widely used in Windows executable compression programs [1], we also added Zopfli to the benchmark. [3]. All of the data compression algorithms are lossless. In our benchmark we tested these compression method on various sample executables (details in Table 3) in four test cases (i) without modification (ii) using instruction reordering only (iii) using split-stream only (iv) the combination of both method. The combination of these two methods are: executing the instruction reordering algorithm than using the split-stream method before the compression. The (i) test case serves as a baseline for evaluation the improvements, the other three uses binary machine code transformation methods.

Instruction reordering is a method, which modifies the order of the instructions inside a basic blocks using data flow constraints, this reordering can improve code compression without changing the behavior of the code [5].

Split-stream is an algorithm that initially partitions a program into a large number of substreams with high auto-correlation and then, heuristically merges certain sub-streams in order to achieve the benefits provided by classical split-stream. It can reduce the increase in compression ratio which typically occurs when a PPM-like algorithm compresses small amounts of data [4]. The actual implementation was developed by Fabian Giesen for the compression program kkrunchy [2].

The implementations didn't use parallel processing methods, each compression method / machine code transformation method was run in a single thread mode on the same machine. We analyzed both the compression time and the result size, in Table 2 you can see each methods' runtime on the nodejs binary. The running time of the compression is influenced by two major factors: the compression method performance and the permutation count for the instruction reordering. Decompression is not effected by the permutation count of instruction reordering, the compression has to be done once for each binary.

	instr. reordering	split-stream	combined
Aplib	2877	3	3224
Gzip	1445	<1	1497
LZMA	2047	2	3620
Zopfli	7426	15	29413

Table 2: Compression times in seconds

Name	OS	Compiler	Source
libc-2.13.so (-0ubuntu13.1)	Ubuntu	gcc	http://packages.ubuntu.com/natty/libc6-i386
unzip (6.0-4)	Debian	gcc	http://packages.debian.org/squeeze/unzip
libconfig.dll (1.4.8)	Windows	VS2008	http://www.hyperrealm.com/libconfig/
node.js (0.8.8)	Mac	llvm	http://nodejs.org/dist/latest/node-v0.8.8-darwin-x86.tar.gz

Table 3: Binaries in the benchmark

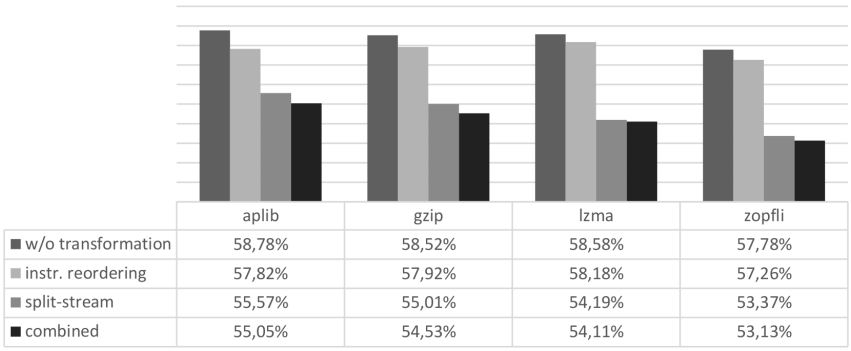

Figure 6: Compressed binary size compared to the uncompressed binary

On Figure 6 you can see the compressed binary sizes compared to the original binary size for the unzip binary. Each method produces significant improvement over the compressed original data.

The combined method gives an average of 9.46% smaller compressed file, than without any binary transformation. The gain by compression method: 6.8% in Aplib, 9.7% in Gzip, 8.7% in LZMA, 12.64% in Zopfli.

References

[1] Árpád Beszédes, Rudolf Ferenc, Tibor Gyimóthy, André Dolenc, and Konsta Karsisto. Survey of code-size reduction methods. *ACM Comput. Surv.*, 35(3):223–267, September 2003.

[2] Fabian Giesen. Working with compression. Online. Last accessed: 2013-01-15, 2006. Breakpoint 2006.

[3] Lode Vandevenne M.Sc. Google Inc. Jyrki Alakuijala, Ph.D. Data compression using zopfli. February 2013.

[4] Steven Lucco. Split-stream dictionary program compression. *SIGPLAN Not.*, 35(5):27–34, May 2000.

[5] Zsombor Paroczi. x86 instruction reordering for code compression. *Acta Cybernetica*, 21(1):177–190, 2013.