Redundancy in instruction sequences of computer programs

Graduate Adviser: Shuichi ICHIKAWA 013740: Kazuhiro HATTANDA

1 Background

There are many instruction sequences that correspond to a program, any of which serves equally for users as long as they are functionally equivalent. In other words, there is redundancy in constructing instruction sequence of a program. Such redundancy can be utilized for watermarking or information hiding [1].

The purpose of this study is to evaluate the redundancy in instruction sequences, particularly in the order of variables and instructions. However, it is very difficult to count all redundancy, because there are many options. Thus, in this paper, only four options are examined; (1) reordering global variables, (2) reordering local variables, (3) reordering basic blocks, and (4) reordering instructions in each basic block. The performance and object size of benchmark programs are also examined before and after the reordering. All measurements were made with ELF object files for Intel x86 architecture, which were generated by GCC 2.95.3 and binutils 2.13.

2 Reordering of variables

Generally, users are unaware of the addresses of variables. Therefore, we can construct functionally equivalent programs by reordering variables on main memory. Since there are $n!$ options to arrange n elements, we can generate $n!$ functionally equivalent instruction sequences with n variables.

There are global and local variables in C language. Global variables are categorized into three types; external variables, static variables, and initialized variables. It is possible to reorder external variables by adding a feature to a linkage loader (*ld*), which arranges the addresses of external variables. Static variables and initialized variables are registered in *.bss* and *.data* sections, respectively, and thus can be reordered by changing the order of definition in assembly files after compilation.

Local variables are allocated on stack or on registers. Local variables on stack are accessed via EBP register with the offset values that are assigned by compiler. Therefore, it is possible to reorder local variables on stack by adding a feature to C compiler. There is also certain redundancy in register allocation, which could be also utilized by enhancing C compiler. However, this redundancy is not examined in this work, leaving it for future works.

3 Reordering of instructions

An instruction sequence is divided into basic blocks, each of which is a sequence of instructions that is executed straight from its beginning to its end. The addresses of basic blocks can be arbitrarily reordered, if the order of execution is maintained by adding unconditional jump instructions (Fig. 1). This redundancy is usable to embed a digital signature into a program [2][3]. In this study, a program was developed to read an assembly file, to divide it into basic blocks, and to reorder these basic blocks.

It is also possible to reorder instructions in a basic block, if the resulting instruction sequence is functionally equivalent to the original one. For example, in Fig. 2, the instructions (1) and (2) are exchangeable, because both sequences yield the same result.

Let $Ref[x]$ and $Sto[x]$ be the sets of read operands and write operands of an instruction x, respectively. Instructions x and y are independent, if the following condition holds: $(Ref[x] \cap Sto[y]) \cup (Sto[x] \cap Ref[y]) \cup (Sto[x] \cap Sto[y]) = \phi$. In this study, a program was developed to reorder instructions in each basic block and to count possible combinations of functionally equivalent instruction sequences in an assembly file.

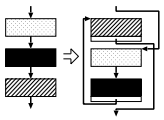

Fig. 1: Reordering basic blocks

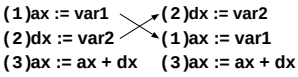

Fig. 2: Reordering instructions

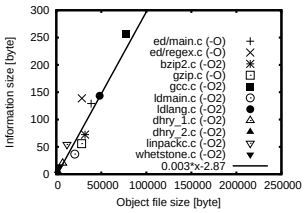

Fig. 3: Information density with basic block reordering

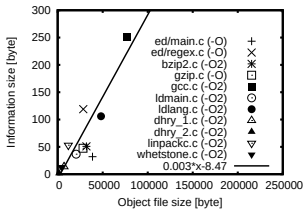

Fig. 4: Information density with instruction reordering

4 Experiments and results

The options to arrange n items are $n! = O(n^n)$, which is difficult to handle when n is large. Thus, PPS (Partial Permutation Scheme) [4] was adopted in this study. Items are divided into chunks, each of which includes 6 items and $6! = 720$ options. Odd items were excluded from measurements. Table 1 lists the open source programs in C language, which were selected for experiments in this study.

Detected redundancy is summarized in Table 1. The redundancy in instructions is larger than that in variables, because the number of instructions is usually larger than the number of variables. In optimized object codes, local variables are allocated to registers and practically no redundancy is derived from them. It is thus necessary to examine the redundancy of register allocation, but it is left for future studies.

Figures 3 and 4 display the relationships between object file size and the information capacity in reordering basic blocks and instructions in basic blocks, respectively. Information capacity was converted from detected redundancy. Information density, which is defined by information capacity divided by object file size, was estimated to be 0.3% for reordering of basic blocks (Fig. 3), 0.3% for reordering of instructions in basic blocks (Fig. 4), and 0.02% for reordering of global variables.

These techniques may have some negative impacts on the performance and the size of instruction sequence. In reordering basic blocks, maximally 6.1% performance degradation was observed in three benchmark programs on a Xeon 2.8GHz system. The performance degradation was less than 5.1% for other three cases. Basic block reordering also incurs (maximally) 4.7% increase in object file sizes of the programs shown in Tab. 1, while no increase was observed for other three techniques.

References

[1] Collberg, C. S. and Thomborson, C.: Watermarking, Tamper-Proofing, and Obfuscation - Tools for Software Protection, *IEEE Trans. Software Eng.*, Vol. 28, No. 8, pp. 735–746 (2002).

[2] Davidson, R. I. and Myhrvold, N.: Method and system for generating and auditing a signature for generating and auditing a signature for a computer program, *USPatent*, No. 5,559,884 (1996).

[3] Hattanda, K. and Ichikawa, S.: The Evaluation of Davidson's Digital Signature Scheme, *IEICE Trans. Fundamentals*, Vol. E87-A, No. 1, pp. 224–225 (2004).

[4] Ichikawa, S., Chiyama, H. and Akabane, K.: Redundancy in 3D Polygon Models and Its Application to Digital Signature, *Journal of WSCG*, Vol. 10, No. 1, pp. 225–232 (2002).

Tab. 1: Options of sample programs for four reordering methods

Program	Compile Option	#Func.	#Line	Object file size [byte]	Global	Local	Basic Block	Inst.
dhry_1.c	-DHZ=100 -DTIME	6	385	7464	5.18e+05	7.20e+02	1.40e+57	1.47e+37
dhry_1.c	-DHZ=100 -DTIME -O2	6	385	7064	5.18e+05	1.00e+00	3.76e+48	4.23e+32
dhry_2.c		6	192	1936	1.00e+00	1.00e+00	1.39e+17	2.15e+10
dhry_2.c	-O2	6	192	1600	1.00e+00	1.00e+00	1.93e+14	8.06e+03
linpackc.c	-DDP -DUNROLL	12	907	15856	5.18e+05	2.69e+11	1.42e+157	6.81e+164
linpackc.c	-DDP -DUNROLL -O2	12	907	11848	5.18e+05	1.00e+00	2.74e+131	1.11e+128
whetstone.c		4	433	5984	7.20e+02	2.69e+11	5.22e+45	1.13e+43
whetstone.c	-O2	4	433	4096	7.20e+02	1.00e+00	1.94e+34	7.56e+31
ed/main.c	default option (-O)	26	1684	38704	7.22e+22	1.00e+00	2.81e+311	2.10e+77
ed/regex.c	default option (-O)	26	5171	28428	1.00e+00	1.00e+00	1.46e+337	4.60e+286
bzip2.c	default option (-O2)	43	2103	31936	3.73e+08	1.00e+00	1.98e+174	5.17e+122
gzip.c	default option (-O)	23	1744	28132	7.22e+22	1.00e+00	1.97e+134	3.82e+114
gcc.c	default option (-O2)	50	5840	77448	5.22e+45	1.00e+00	1.53e+617	7.18e+602
ldmain.c	default option (-O2)	20	1376	20512	7.20e+02	1.00e+00	3.78e+88	1.61e+88
ldlang.c	default option (-O2)	126	5525	48128	1.93e+14	1.00e+00	5.46e+345	2.39e+255