Exploring Parameter-Type Compiler Options for Accelerating Program Execution

Takeo Saeki, Junya Goto, Nagisa Ishiura

School of Science and Technology, Kwansei Gakuin University
2-1 Gakuen, Sanda, Hyogo, 669-1337, Japan

***Abstract*—This paper proposes an extensive use of parameter-type compiler optimization options to speed up program execution. Compilers usually provide a lot of options to control their optimizers. However, existing methods of exploring the best option sets for given programs dealt only with switch-type options which specifies activation/deactivation of individual optimizers. This paper extends the search domain to parameter-type options which control optimizers by numerical values. A genetic algorithm is employed to explore the best combination of the parameters as well as the optimization switches for given programs. An experiment on GCC 4.8.1 with four benchmark programs shows that the proposed method accelerates the program execution by 10.4%, at the cost of 38.0% increase in the exploration time.**

I. Introduction

Optimization to enhance the efficiency of codes is the core task of compilers, and modern compilers implement a lot of transformations for optimization. Since the effects of the transformations depend on the properties of the programs and some optimization may even degrade the performance, some compilers provide options to control individual optimizers. However, as the number of options increases, it gets difficult to find option sets which best match with given programs. Since GCCs later than version 4.0 have more than 100 options, it is virtually impossible to tune the option set manually.

To meet this issue, there have been a lot of researches to automate the searches for the best combinations of optimization options for given programs. Pan [1] and Lin [2] attempted to find the option sets by iterative methods and by an genetic algorithm, respectively. Their main target was the speed of the resulting codes and they succeeded in outperforming GCCs' `-O3` option which turns a pre-selected set of optimization options. On the other hand, Patyk [3] tried to reduce energy consumption through automatic selection of compiler options based on a statistical non-parametric analysis. Haneda [4] used the similar method to reduce the size of object codes.

All those researches have dealt only with *switch-type* options, which activate or deactivate optimizers. However, more recent compilers such as GCC 4.8.1 accept *parameter-type* options, through which optimizers are controlled by numeric values. Such options might further enhance the performance of the execution codes, but there have been no research to extend the option set search to the parameter-type options.

In this paper, a genetic algorithm based search for the best options, considering both the switch-type and the parameter-type options, is proposed. A vector of parameter values to the options (0 or 1 for the switch-type options) is used as the chromosome representation of an individual in a population. The execution time of compiled codes is measured using a UNIX time command where codes are run multiple times to make the measurement precise.

An experiment on GCC 4.8.1 with four benchmark programs shows that the utilization of parameter-type options accelerated the program by 10.4% on an average, which makes the programs faster than `-O3` option by 40.2%.

II. Compiler Option Sets for Optimization

Recent compilers implement many optimizing transformation routines. They also provide command line options to control each of the optimizers. The options are classified into switch-type options and parameter-type options, where the former specify activation/deactivation of the optimizers and the latter feed numeric values to the optimizers.

For example, GCC 4.8.1 has 186 switch-type options. An `-fdse` command line option turns on a dead code elimination routine and an `-fno-dse` turns it off. Besides them, it has 154 parameter-type options. A `--param max-inline-insns-single=100` option, for example, limits the number of the instructions resulting from inline expansion to 100. Options to specify optimization levels, such as `-O1` through `-O3`, set predefined values to these options. For example, the `-O1` option turns on 38 out of the 185 options.

Options sets tailored for given programs often produce higher code performance than the standard optimizing options. As will be shown in the experimental results, tuning of the switch-type options only

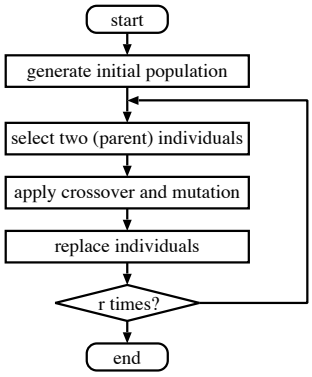

Fig. 1. Flow of genetic algorithm based search.

accelerates the code by 27.0% as compared with the `-O3` option, so its worth exploring the option sets.

However, switch-type options of GCC 4.8.1 alone result in 2^{186} possible combinations. Since it is infeasible to find the best one by hand or by exhaustive search, there have been researches to search optimal solutions for given programs by meta heuristics, machine learning, etc.

III. Incorporating Parameter-Type Options in Option Set Search

A. Genetic Algorithm Based Search

Given a source program, our goal is to find a set of the switch-type options to be turned on and a value vectors to all the parameter-type options that minimizes the execution time of the compiled code. In this paper, a genetic algorithm is employed to find optimal solutions.

B. Encoding of Option Set

A solution, or an individual in a population, is represented simply by a vector of the values to the options, where the value for each switch-type option is 0 or 1, meaning deactivation or activation, respectively. Some parameter-type options may have large domains. To curve the search complexity, we reduce the domains to the set of representative values. For example, when an option's domain is $\{1, 2, 3, 4, 5, \cdots, 1024\}$ (of 1024 elements), we assume a reduced domain $\{1, 2, 4, 8, \cdots, 2014\}$ (of 11 elements). The representative values are selected taking the properties of the options into account.

C. Fitness Function

Since our goal is the reduction of the execution time of a given program, the fitness function of each individual is defined as the execution time of the code generated with the option set encoded in the individual.

We assume that the time is in terms of the real run time on a target computer, which is measured for example by a UNIX `time` command. Here, fluctuation becomes a problem, since the code execution time varies largely depending on various conditions, such as the status of the cache or the other processes running on the computer system.

To solve this problem, we run the code multiple times and pick the smallest run time as the fitness function value. We use the minimum time instead of the average time because the factors outside of the program only delays the execution, so the minimum time best reflects the effect of the optimizers.

We change the number (k) of runs during the search; we start with $k = 1$ (to curve total execution time) and gradually increase k as the search converges (to increase accuracy).

TABLE I
Example of selection probabilities.

f_i	$w_i = (f_{\max} - f_i)^2$	$p_i = w_i/W$
1.2	0.64	45.1%
1.3	0.49	34.5%
1.5	0.25	17.6%
1.8	0.04	2.8%
2.0	0.00	0.0%

$f_{\max} = 2.0, \quad W = \sum w_i = 1.42$

D. Selection, Crossover, and Mutation

Fig. 1 shows the flow of the search. It starts with generating an initial population. Let the population consist of n individuals. The option value vector for each individual is randomly chosen.

Then, the loops are repeated for r times. First, two individuals (parents) are chosen from the population. Crossover and mutation are applied to generate two new individuals. Some two individuals in the population are replaced by the new individuals.

In our method, the parents are picked up by a roulette selection method. Namely, each of the two individuals is chosen with the probability defined to the individual. Let f_i be the fitness value of the individual i, and $f_{\max}$ be the maximum value of f_i among the population. Then individual i is selected with a probability proportional to its weight w_i which is defined as:

$$w_i = (f_{\max} - f_i)^2$$

Let W be the sum of w_i over all the individuals, then the probability p_i with which i is chosen is $p_i = w_i/W$.

TABLE I shows an example of the selection probabilities, where the population consists of individuals with fitness values 1.2, 1.3, 1.5, 1.8, and 2.0. The second column and third column are the weights and the probabilities of the individuals. The weight is designed so that individuals with better fitness values may be selected more probably.

TABLE II
EXPERIMENTAL RESULT.

program	execution [s]				exploration [h]	
	-O0	-O3	sw	**sw+p**	sw	**sw+p**
b_queen.c	5.42	1.41	1.08	0.86	2.71	4.67
fasta.c	6.19	1.60	1.07	1.10	3.13	5.20
nbody.c	14.17	2.16	1.97	1.71	6.40	6.72
bitcnts.c	4.24	1.59	1.31	1.24	4.06	4.90
ratio [%]	563.0	140.2	110.4	100.0	100.0	138.0

Compiler: GCC 4.8.1 (x86)
CPU: Intel Core i2 (1.4 GHz)
Memory: 3.8 GB
OS: Ubuntu 13.10

Crossover and mutation operations in our method are very conventional. A uniform crossover operation is employed. During mutation, the value of each chromosome is changed at a certain mutation probability.

New two individuals generated from the parents are inserted into the population. Instead, the two individuals with the worst fitness values are discarded.

IV. EXPERIMENTAL RESULT

Based on the method stated so far, a search program has been implemented in Perl 5 which runs on Unix operating systems.

An experiment was done on four C programs. The compiler used in the experiment was GCC 4.8.1 for x86 target. Out of all the optimizing, those caused errors or warnings were excluded. This resulted in 164 switch-type and 134 parameter-type options. The program was run on Ubuntu 13.10 on Intel Core i2 CPU (1.4 GHz) with a 3.8 GB memory.

The parameter settings for the genetic algorithm were as follows:

- population size: 100
- iteration: 1,000
- mutation probability: 0.05

Candidate option sets obtained during the search were given to GCC compiler together with the `-O3` option. This is because the GCC has implicit optimizing options which can only be activated by either of the standard optimization option. Generated codes were executed 1 to 5 times, depending on the iteration count, to measure the execution time of the codes.

TABLE II shows the result of the experiment. Columns `-O0` and `-O3` are the execution time (seconds) of the generated codes by GCC's standard optimization options, where `-O0` means no optimization and `-O3` optimization with the highest level. The next two columns are the execution time resulted from option set search with the switch-type options only (sw) and with the options of the both types (sw+p), where the latter is the proposed method in this paper. The last two columns indicate time (hours) consumed for the option set search for the two methods. The bottom row "ratio" shows the geometric means normalized to sw+p for execution time and sw for exploration time.

We can see that the programs were accelerated by 10.4% by incorporating the parameter-type options into the search. This was a speed-up by 40.2% as compared with the `-O3` options. The search time was increased by 38.0% which we believe is reasonable.

V. CONCLUSION

This paper has proposed the utilization of parameter-type compiler options as well as switch-type options to enhance the performance of compiled codes. An experimental result demonstrated that genetic algorithm based search successfully found option sets which accelerates the execution speed of the code.

Future work includes refinement of the genetic search procedure, and consideration of other objectives such as the size or power consumption of the generated codes.

ACKNOWLEDGMENT

We would like to thank Prof. Tomiyama of Ritsumeikan University for his advice on the definition of fitness function. Authors would like to express our thanks to all the members of Ishiura Laboratories of Kwansei Gakuin University for their advice and discussion for this work.

REFERENCES

[1] Z. Pan and R. Eigenmann: "Fast and Effective Orchestration of Compiler Optimizations for Automatic Performance Tuning," in Proc. International Symposium On Code Generation and Optimization, pp. 319–332 (Mar. 2006).

[2] San-Chih Lin, Chi-Kuang Chang, and Nai-Wei Lin: "Automatic Selection of GCC Optimization Options Using a Gene Weighted Genetic Algorithm," in Proc. Computer Systems Architecture Conference, pp. 1–8 (Aug. 2008).

[3] T. Patyk, H. Hannula, P. Kellomaki, and J. Takala: "Energy Consumption Reduction by Automatic Selection of Compiler Options," in Proc. International Symposium on Signals, Circuits and Systems, 2009, pp. 1–4 (July 2009).

[4] M. Haneda, P. M. W. Knijnenburg, and H. A. G. Wijshoff: "Code Size Reduction by Compiler Tuning," in Proc. International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, pp. 186–195 (July 2006).