Inadequacy of Genetic Algorithm as Scheduling Optimization to Enhance the Performance of Source Code for Real Time System

M R Dhande,
Department *of Computer Technology, VYWS Polytechnic, Badnera*
Amravati, India

R A Tiwari,
Department *of Information Technology, VYWS Polytechnic, Badnera*
Amravati, India

R R Tuteja,
Department *of Computer Science & Engg., PRMIT, Badnera*
Amravati, India

R V Wasu
Department *of Computer Technology, VYWS Polytechnic, Badnera*
Amravati, India

V K Patil
Department *of Information Technology, VYWS Polytechnic, Badnera*
Amravati, India

ABSTRACT

Without any optimization option, the compiler's goal is to reduce the cost of compilation and to make debugging produce the expected results. Statements are independent: if you stop the program with a breakpoint between statements, you can then assign a new value to any variable or change the program counter to any other statement in the function and get exactly the results you would expect from the source code. Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program. The compiler performs optimization based on the knowledge it has of the program. To optimizing performance of program for real time system doesn't always mean what we might think. It's not just a matter of outright speed; sometimes it's about tuning the code and data so that it fits into a small memory footprint. It is hard-pressed to find a programmer that does not want to make programs that run faster, regardless of the platform. Real time system programmers are not exception for that some take an almost over-enthusiastic approach to the job of optimizing their code for performance.

1. INTRODUCTION

Now a days many traditional compiler optimizations are designed to reduce the execution time of compiled code, but not necessarily the size of the compiled code. Further, different results can be achieved by running some optimizations more than once and changing the order in which optimizations are applied. The interactions between different optimizations can cause more spill code to be generated. The compiler for real time systems must take care to use the best sequence of optimizations to minimize code space. Since much of the code for real time systems is compiled once and then burned into ROM, the software designer of real time system will spend lots of their time on compilation of source code in the hope of reducing the size of the compiled code. Some researchers take advantage of this by proposing new techniques to find optimization sequences that generate small object codes. For example Keith D. Cooper et al propose use of genetic algorithm to set optimum flag sequence. They shows the solutions generated by GA are optimal as compared to solutions found using a fixed optimization sequence and solutions found by testing random optimization sequences. Based on the results found by the genetic algorithm, a new fixed sequence is developed to reduce code size. They explore the idea of using different optimization sequences for different modules and functions of the same program.

2. GENETIC ALGORITHM (GA)

Genetic algorithm [3] is an algorithm which allows finding the best solution of a given problem, basing on a process of natural selection, known from biological systems. Set of parameters that solve the problems are coded in genes of individuals. Genetic algorithms process set of individuals called population by performing cyclic operations of evaluation, selection, crossover and mutation. Fitness function, used for evaluations of individuals, is crucial for proper work of genetic algorithms. Results of evaluation are used for selection of individuals. Like in natural processes, solutions with good fitness go through, whereas bad solutions are rejected. .

3. INADEQUACY'S OF GA TO SCHEDULING OPTIMIZATION

We face many difficulties while implementing GA based approach for optimizing source code. Actually in the basis of GA there are some limitations like

Population Diversity

One of the most important factors that determine the performance of the genetic algorithm performs is the *diversity* of the population. If the average distance between individuals is large, the diversity is high; if the average distance is small, the diversity is low. Getting the right amount of diversity is a matter of start and error. If the diversity is too high or too low, the genetic algorithm might not perform well.

Setting the Population Size

The population size field determines the size of the population at each generation. Increasing the population size enables the genetic algorithm to search more points and thereby obtain a better result. However, the larger the population size, the longer the genetic algorithm takes time to compute each generation. So

for GA it is always the matter of search what is the approximate population size that balance well between better result and compute time.

Fitness Scaling

Fitness scaling converts the raw fitness scores that are returned by the fitness function to values in a range that is suitable for the selection function. The selection function uses the scaled fitness values to select the parents of the next generation. The selection function assigns a higher probability of selection to individuals with higher scaled values. The range of the scaled values affects the performance of the genetic algorithm. If the scaled values vary too widely, the individuals with the highest scaled values reproduce too rapidly, taking over the population gene pool too quickly, and preventing the genetic algorithm from searching other areas of the solution space. On the other hand, if the scaled values vary only a little, all individuals have approximately the same chance of reproduction and the search will progress very slowly.

Selection

The selection function chooses parents for the next generation based on their scaled values from the fitness scaling function. For this selection function must needed varied scaled value. Here another problem arrives that is if any individual comes with the highest scaled values than that individual gene repeated by this selection function in almost every individuals of next generation. This results that the scaled values vary only a little and all individuals of next generation have approximately the same fitness and each have chance of reproduction it makes system slow.

Now we see some inadequacy of Keith D. Cooper et al works with respect to GA. As they shown their result which is the outcome of GA sequence in first sequence the result generated by GA consist some repetition of optimizations. So it is required to generate a second sequence that gives reduced sequence as shown in Table 1.

Benchmark	GA sequence	Reduced sequence
adpcm	tonosdnzscno	osnzc
compress	ncsnzosvndvs	nczvnds
fmin	tdcotcscnnvo	dcsvo
svd	odvdvdsnnnss	odvs
urand	cnottcdtvooc	nodvc
zeroin	rsdosvncnsss	rsosvcs

Table 1: Optimization sequences found by the GA.

And in Table 2 corresponding optimizations are shown.

Gene	Optimization
c	cprop
d	dead
l	partial
n	clean
o	combine
r	shape
s	coalesce
v	valnum
z	lazy

Table 2: Corresponding optimizations.

In second sequence smaller sequences were found by an automatic tool, starting from the sequence returned by the GA and removing optimizations from the sequence one at a time. To set automatic tool one another work-list algorithm is used, so to the work-list being initialized with the sequence returned by the GA. In first look it seems like paying penalty for quality by bearing extra work load [2]. Keith D. Cooper and his team work to computing fitness values at same time they also simulate a run of the optimized code to test for correctness. Because there is chance for error encountered during the optimization or the execution. In case they assigned fitness value to infinity to the chromosome [2].

Also it takes lots of time to calculate fitness values as number of generations increased. This fitness values actually requires to optimizing the code.

Next inadequacy of GA is that it produces quality result only if the number of generations for GA is limited that number must be limited up to 100 generations [2].

Another difficulty with GA as a code optimizer is the fine tuning of the GA parameters [2].

4. CONCLUSION

In this paper, we described inadequacies of a genetic algorithm that is designed to find compiler-optimization sequences resulting in reduced static code sizes.To get the best results from the genetic algorithm; you usually need to experiment with different options. Selecting the best options for a problem involves start and error. So we fill that there is a requirement of an optimization procedure which is simple to understand, easy to implement and no partial redundancy elimination that partial frequently adds operations to the code in order to reduce the lengths of some paths through the code. The effects of partial can reduce run times but tend to increase code size.

REFERENCES

[1] GNU Compiler Collection *http://gcc.gnu.org/onlinedocs/gcc/Gcov.html*

[2] Keith D. Cooper, Philip J. Schielke, and Devika Subramanian , “Optimizing for Reduced Code Space using Genetic Algorithms”, Rice University Houston,Texas, USA.

[3] Goldberg D. E., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley Longman Publishing, Boston, MA, USA, 1989.