Code Optimization using Genetic Algorithm

Prof R Kannadasan, K.N. Manoj Kumar, Karthik Sistla, Ashlin Sherry

School of Information Technology and Engineering,School of Computer Sciences and Engineering

VIT University, Vellore

ABSTRACT

Code optimization has always been a critical area for both programmers and researchers alike. Minimizing the code execution time and code size have the highest priority in code optimizations. Genetic Algorithm is a kind of technique that is employed in order to solve optimization problems. The main idea behind the genetic algorithm is exactly like the law of the jungle “Survival of the Fittest”. It is evolutionary. We keep on working towards getting better and improved solutions till we get an optimal/best solution. The main tasks or operations performed by the genetic algorithm are selection and crossover along with mutation which we will discuss in length. Genetic algorithm mainly uses the following operations which are Encoding, Selection, Crossover, and mutation. Genetic Algorithms are used in a wide array of fields and have lots of applications.

INTRODUCTION

In the area of optimization, researchers are always looking to find the best/optimized solution. In this paper, we mainly focus on how the genetic algorithm helps optimize a solution. Compiler optimization means making the output of a compiler increase or decrease the impact of certain parameters of a computer program. The main aim of optimization is to find out the efficient and best value for all the given attributes in order to gain credible performance. Performance is measured based on parameters such as the time taken to execute a code or the size of the code (Kilo lines of code) etc. which also has an impact on the execution of the whole application. Most of the work done in the area of optimization has found out that evolutionary algorithms (EA) are most efficient when it comes to optimization. EA's consider a group of the population and then carries out observations in the behavior and characteristics of the population. Many of the evolutionary algorithms mainly focussed on reducing the search time and not on the performance which was a major drawback as performance is a crucial factor. Hence genetic algorithms came into play which reduced the above-mentioned flaw by providing an optimal solution with reduced search time and high performance from a huge set of solutions Genetic algorithm comes under the concept of evolutionary computation and works most efficiently for carrying out global optimization. They are EA's which are mainly search based and they replicate the process which naturally occurs and then performs the selection. Genetic Algorithm has 4 crucial steps or factors which are

1) Selection 2) Crossover 3) Mutation 4) Termination 5) Encoding

With regard to compiler optimization, we assume a flag of the compiler as a gene and when more than two flags combine (i.e. when more than 2 genes combine), then they form something which we call as a chromosome.

LITERATURE SURVEY

In [1] the authors reviewed work associated with genetic algorithm operations and applied the genetic algorithms to the pipeline systems. Genetic Algorithm can be easily applied to situations where there is little prior understanding about the problem. Due to its stochastic nature, GA can find the global optimum. In water pipeline systems, there are certain goals to be achieved such as minimum construction cost, minimal pipe size, etc. GA outperformed the other optimization techniques. Moreover, the implementation and use was easier in GA.

In [2], the authors proposed constructing composite sorting algorithms from primitives according to the target platform and input data. GA was used to search for sorting techniques. The results showed that the best sorting techniques were obtained when using GA to create a classifier system. The resultant algorithm is an amalgam of different sorting techniques selected based on the entropy and the number of keys to sort. Most of the cases, the selected technique was radix based with different parameters dependent on the input and target machine. The combined sorting algorithm was on average 36% faster than the pure sorting algorithms.

GENETIC ALGORITHM

GA's are the most famous Evolutionary algorithms in existence today. They have a wide array of applications and are used in numerous fields for a variety of problems.

Genetic Algorithm is loosely based upon the renowned scientist Charles Darwin's "Theory of Evolution" whose base concept is survival of the fittest which means the fittest member of the population is the one that will overcome all odds and survive and then reproduce to continue to the next generation. Therefore the central idea behind GA is taken from biology and is based on survival change and adaptation. The GA is basically a search algorithm which is based upon selection in a natural manner and has the added advantage of searching complex spaces to provide optimal solutions. However, it is different from other regular search and optimization algorithms. GA works along with the coding of particular solutions and not the solutions. Also, it performs the search operation on the entire population of solutions and not on each individual solution. It also has no need for derivatives and makes use of the cost function, unlike the other search and optimization algorithms.

Before going into the genetic algorithm we need to understand few terms which are biological but related to genetic algorithms.

Gene: Gene is the tiniest unit which carries some information about a particular set or population

Individual: An individual is a group of genes which carries information .Individuals are also known as chromosomes

Population: A group of individuals can be termed as population

The steps involved in implementing the genetic algorithm are as follows

1) Encoding: Encoding is carried out on all the decision variables present in the problem. All these variables are encoded and then a string of finite length is generated. The string could be either a binary string or string with integers.

2) Initialization: In the initialisation phase we take into consideration a large no of parameters such as the size of the population, the probability of crossovers and mutations and the no of individuals in a generation etc.

3) Evaluation: In the evaluation phase we make use of a fitness function whose goal is to find the fitness of each individual of the population. Based on the fitness value the selection process is carried out.

4) Selection: It is a highly crucial step in the genetic algorithm. Based on the selection, the search spaces are determined and also the chance of survival of individuals is also based upon the selection. The selection operator comes to work mainly at the individual level. Fitness is one crucial factor which determines the level of goodness of the individual. It can be predicted by

using an objective function and can also be determined by using a subjective judgment procedure which is unique for each problem. With the passage of time and generations, the individual in the population should get fitter which means that the individuals should be nearer to the desired solution. There are many selection mechanisms in existence. There are user specified selection mechanisms as well as the selection mechanisms which are traditionally in use. Different mechanisms work optimally with context to the given situation. Therefore the optimal mechanism needs to be chosen in order to obtain an optimal solution

5) Crossover: Crossover plays a very crucial role in creating a new generation as the features are carried onto the new generation and its population. Crossover is the mechanism where two parents mate to produce an offspring. Speaking in computational terms crossover is where two chromosomes combine in order to produce a new chromosome. The central idea behind crossover is that the resulting chromosome will have better features and survivability than that of their parents as the best features from each individual are considered and then crossover is carried out. There are a variety of crossover operators in existence such as:

Single Point Crossover

A single point is selected on the parents. All the existing content beyond that point in both the individuals is swapped between the two parent chromosomes and the result is the child chromosomes with those features.

Two Point Crossover

As the name suggests, two points are selected on the parent chromosomes. All information contained between the points is swapped, resulting in two child chromosomes or child organisms.

Cut and Splice:

In this approach, each parent has a different crossover point. This causes a difference in the length of the children strings. This is known as the cut and splice approach.

Three Parent Crossover:

As the name suggests, the child chromosome is obtained from three parents who are randomly chosen. A part of the first parent is compared with the same part of the second one. If both are same then it is carried over to the child, otherwise, the part from the third parent chromosome is carried over to the offspring.

Uniform Crossover:

In Uniform crossover the bits are copied in a random fashion either from the first parent or the second parent.

6) Mutation: Mutation is employed in order to ensure diversity in the new population and it occurs during the evolution phase. Mutation changes one or more genes of a parent chromosome from

its starting state. Therefore the solution might be an entirely different one when compared to the older solution. Therefore there is a probability of obtaining better solutions by performing mutation.

7) Decoding: Decoding of the encoded string is carried out in this final phase and the encoded string is reverted back.

GENETIC ALGORITHM FOR CODE RUNTIME OPTIMIZATION

The steps of genetic algorithm for runtime optimization are listed below:

Step 1: An initial population of chromosomes (compiler flags of the compiler).

Step 2: A sample program of user interest is considered, it is then input into the Genetic Algorithm.

Step 3: Cross-over the chromosomes to produce new chromosomes.

Step 4: The program is compiled using combination of various flags and executed.

Step 5: The compilation and execution time durations are noted.

Step 6: The Fitness function is derived as a function of compilation and execution times which is as shown below:

$$Fitness = \frac{1}{compilation\ time + Execution\ time}$$

Step 7: The fitness value for each chromosome in the generation is calculated.

Step 8: The chromosome with the maximum fitness value and corresponding combination of flags is said to be the best combination of flags for that particular program since it results in minimum compilation and execution times.

Step 9: Repeat the process and select the best chromosome over generations.

Step 10: Terminate when the best combination of flags is identified for compiling and executing a particular code.

The result of this process is the chromosome which has the best fitness value. GA uses a fitness function to measure the performance of each chromosome. In this case, the fitness function is measured as the inverse sum of compilation and execution times.

1) Constant Propagation: Constant propagation is also known as cprop and is largely based upon sparse conditional cprop technique proposed by Wegman and Zadeck. Cprop is executed by choosing an option (-m) that does not allow the conversion of multiply into shifts and adds. This results in increased code sizes. Dead Code Elimination:Dead Code Elimination is referred to as DCE and is a compiler optimization mechanism, which is employed to remove dead or non-functioning areas of the code. However it has no effect on the results of the program
2) Empty Basic Block Removal: Many blocks have only a single control flow operation and are of no real use. This eliminates all such blocks.
3) Global Value Numbering: Value numbering (valnum) implementation has many options to choose form. We have particularly considered using the constant folding, algebraic simplification and value driven code motion.
4) Lazy Code Motion: Delaying of evaluation of expression unless until absolutely necessary.
5) Partial Redundancy Elimination: It is a technique that tries to eliminate all the parts of a code that are recurring. It is a form of common subexpression elimination. An expression can be termed as partially redundant if it is available on some selective paths but not all the existing paths. This removes the redundant ones by putting the redundant expressions on those paths which do not compute it.
6) Peephole Optimization: Peephole Optimization is generally performed over a limited or a small set of instructions in a predefined segment of the code and not the entire code. The small set of instructions taken into consideration is called the peephole. This basically focuses on replacing the prescribed set of selected instructions with instructions that are shorter or instructions that can run faster.
7) Re-Association: This is used to re order the given expressions that using basic laws like the law of associativity, laws of commutativity etc. The central idea behind it is to re associate expressions which are complex in order to bring out subexpressions which are loop invariant.
8) Register Coalescing: Coalescing is basically coming together to form a single whole. This brings together the register copies by making use of the interference graph.
9) Operator Strength Reduction:This basically reduces the strength of the operator as the name suggests using certain algorithms. These algorithms are different from other conventional algorithmsbecause the algorithm does not repeat all the steps in an iterative manner. It also finds the non-primary effects given in a single pass.

CONCLUSION

In this paper, we have described the techniques to optimize code runtime, and code space using Genetic Algorithm. GA gives superior performance compared to any other optimization technique. GA designed to find compiler-optimization sequences gives reduced static code sizes. Use of Genetic Algorithm enabled testing of programs for different chromosomes and across many generation, and thus gives a set of compiler flags which result in the lowest compilation and execution time.

REFERENCES

[1] I. Abuiziah, N. Shakarneh, "A Review of Genetic Algorithm Optimization: Operations and Applications to Water Pipeline Systems", World Academy of Science, Engineering and Technology International Journal of Mathematical, Computational, Physical, Electrical and Computer Engineering Vol:7, No:12, 2013

[2] Xiaoming Li, Mar´ıa Jesus Garzar ´ an and David Padua "Optimizing Sorting with Genetic Algorithms", Proceedings of the International Symposium on Code Generation and Optimization (CGO'05)

[3] A.H. Wright, "Genetic Algorithms for real parameter optimization: Foundations of Genetic Algorithms" J. E. Rawlins, Ed. San Mateo, CA: Morgan Kaufmdnn, 1991, pp. 205-218.

[4] Battiti Roberto, Mauro Brunato, Franco Mascia (2008), "Reactive Search and Intelligent

Optimization", Springer Verlag. ISBN 978-0-387-09623-0

[5] Han Lee1,, Daniel Von Dincklage,1 Amer Diwan,1,_And J. Eliot B. Moss, "Understanding The Behavior Of Compiler Optimizations

[6] http://en.wikipedia.org/wiki/Genetic_algorithm

[7] Steven John Beaty. *Instruction Scheduling Using Genetic Algorithms*. PhD thesis, Colorado State University, Fort Collins, Colorado, 1991.