International Journal on
"Technical and Physical Problems of Engineering"
(IJTPE)
Published by International Organization of IOTPE

ISSN 2077-3528
IJTPE Journal
www.iotpe.com
ijtpe@iotpe.com

September 2014 | Issue 20 | Volume 6 | Number 3 | Pages 97-102

A CODE REORDERING BASE METHOD FOR STALL REDUCTION USING GENETIC ALGORITHM

S. Saeedvand A. Allahvirdizadeh N. Fathalizadeh S.J. Mojaveri M. Zolfy Lighvan

Computer Engineering Department, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran
s-saeedvand91@ms.tabrizu.ac.ir, a-allahverdizadeh91@ms.tabrizu.ac.ir, n-fathalizadeh91@ms.tabrizu.ac.ir
s-j-mojaveri91@ms.tabrizu.ac.ir, mzolfy@tabrizu.ac.ir

Abstract- The present study was aimed at minimizing the number of mandatory stalls between the instructions of a processor in compiling time in order to improve the execution time of an instruction sequence. To this end, a code reordering mechanism was employed. Given that the code reordering problem is an NP-complete one (Non deterministic polynomial), this study proposed a genetic algorithm (GA) structure for improving processors' performance. The algorithm, considering hardware limitations of processors, was able to statically minimize the number of mandatory stalls between instructions at compile time, thus increasing processing speed.

Keywords: Code Reordering, Genetic Algorithm, Stall, Chromosome, Fitness.

I. INTRODUCTION

Today, a wide variety of pipeline processors are used in digital systems [1], and making these processors more efficient seems highly important. This study is an attempt to increase the speed of these processors through code-reordering. Code reordering is generally used for improving the speed of instructions execution in pipeline processors. Different reordering methods have different purposes and characteristics. One of the most important methods of improving the run-time in pipeline processors is code reordering in a dynamic manner. In such methods, using different algorithms, instructions are dynamically reordered during the run time by making changes to the hardware structure of processors so that it can reduce the number of mandatory stalls between the instructions.

Most dynamic code-reordering methods have their own problems, especially hardware problems, which are beyond the scope of this study. Another application of code-reordering in processes is to develop a code-reordering algorithm to minimize the amount of cache-miss-rate. Code reordering is also used to improve the speed of retrieving data from the memory [2-5] or even for tracking systems [15]. This study sets out to introduce a method for static code reordering in order to present a pre-compiler which is able to reorder the processor's instructions before they are compiled, and to reduce the amount of mandatory stalls between the instructions caused by hardware limitations.

Static code-reordering is a non-deterministic polynomial problem and is regarded as an NP-complete problem (The non-deterministic polynomial problem is a matter of decision which is solvable by temporal non-deterministic polynomial algorithms) [8]. The current paper provides a solution for this problem by introducing a genetic algorithm with a new chromosome structure. The proposed algorithm is considered within the limits of instructions of MIPS processors.

A number of studies have been conducted on improving the efficiency of processors. Some of these studies are mentioned here, each of which offers a different method for improving processor efficiency. John Ruttenberg et al. [6] compared two code-producing techniques to find out which one could produce the best cods for pipeline processors with ILP architecture at the compile-time. They examined their first technique on MIPS processor compiler and the second one by allocation of registers and scheduling. Hung Wang et al. [7] presented a method for register renaming and scheduling the dynamic performance of predicted codes. They could enhance the efficiency of processors up to 16% by evaluating and improving the performance prediction in a dynamic schedule. In the following of the paper, first section of the paper illustrates the proposed method; the second section explains the implementation of the method, and the third section presents the results.

II. BACKGROUND

A. Genetic Algorithms

A genetic algorithm (GA) [10] is a search heuristic that mimics the process of natural evolution. This heuristic is routinely used to produce useful solutions to optimization and search problems. Genetic algorithms belong to the larger class of evolutionary algorithms (EA), which provide solutions to optimization problems using techniques inspired by natural evolution such as inheritance, mutation, selection, and crossover. In genetic algorithms, a chromosome is a set of parameters which define a proposed solution to the problem that the genetic algorithm is trying to solve [14].

B. MIPS Processors

The processor addressed in this study is the MIPS processor. The MIPS processor, designed in 1984 by researchers at Stanford University, is a RISC (Reduced Instruction Set Computer) processor. In this processors, execution of an instruction in a processor can be divided into a number of stages. The number and functions of stages are different in different processor designs [11-12].

C. Load Hazards

Consider the following code fragment:

```
lw   $4, 1($3)
// Load the value at address 1+$3 into $4
add $1, $4, $3
```

The "lw" only writes the value from memory into the register file in the WB phase, whereas the "add" retrieves its operands in the ID phase. Thus, the "add" actually uses an old value of "$1". As before, it is better to ignore this problem and call it a feature. The pipeline could also be stalled for two cycles [13].

III. THE PROBLEM FORMULATION

In the proposed code reordering mechanism, each MIPS code in the initial specified order is considered a chromosome the size of which is calculated by using the following equation.

$$L_{ch} = (L * R) \quad (1)$$

where L_{ch} is the chromosome size, L is the number of the instructions, and R is a constant with 7 values, indicating the number of chromosome columns (for each instruction, the individual data item is stored in the specified column). The Stored information and details of chromosomal structure will be explained in the 'chromosome display section'.

In the suggested method, the increase of population size will decrease the required run-time of the proposed algorithm. On the other hand, the small population size may direct the problem to a solution, which is not necessarily optimal. As a solution, it is suggested that the population size be a function of the chromosome length; that is, the longer the chromosomes are, the larger the population will be.

$$P_{Size} = \frac{L}{L-E} \quad , \quad Ch_{Size} = P_{Size} * K \quad (2)$$

The equation on the right shows that the child population size (Ch_{Size}) is K times as large as the parent population size (the constant K is obtained by a try and error method). In the left-hand equation, P_{Size} stands for the suggested parent population size; L is the number of instructions; E is the number of independent instructions in the chromosome, and K is a constant coefficient for determining the parent population size, which is calculated through a try and error method and its amount is approximately equal to 6.

IV. LIMITATIONS

There are some limitations to the solution of the code-reordering problem, which can be divided into two major groups: hard limitations, which should be totally considered in the schedule, and soft limitations, which should be taken into account in the problem as far as possible.

A. Hard Limitations

Hard limitations refer to the maintenance of the date flow between instructions; in fact, in every code, some instructions are dependent upon the results of previous ones. Therefore, during the code reordering, instructions cannot be transferred to a position before those dependent on them. It should also be noted that the logical structure of codes can by no means be violated in branches. These are regarded as hard limitations, and the production of those codes in different stages of running the genetic operators in each chromosome is prevented.

B. Soft Limitations

Soft limitations are the limitations that reduce the fitness of a chromosome, but their presence in the chromosome is tolerable. In this study, soft limitations are defined as the number of stalls in each chromosome specifying a certain temporal penalty for running that chromosome's codes in the processor.

C. Chromosome Display

As mentioned in the problem formulation and population size calculation, each permutation is counted as one chromosome, and each chromosome is defined as a two- dimensional array.

Row Number	Op-code	Reg_Name1	Reg_Name2	Reg_Name3	Depend_D	Depend_U
1						
2						
. . . . .						
N-1						
N						

Figure 1. Chromosome structure

Each code instruction is called a gene (the term will be used later). In fact, the production of a chromosome involves the conversion of a phenotype space to genotype space. Figure 1 shows a two-dimensional example of these chromosomes. As seen in the Figure, the first column indicates the gene number of each instruction; the second column shows the operation code of each gene; the third up to fifth columns reveal the names of the used registers in each gene; the sixth column indicates the first successor dependent gene, and the seventh column shows the first predecessor dependent gene. It should be noted that the sixth and seventh columns specify the data flow of the genes in the chromosomes.

D. Sample Chromosome Generation

The codes that were reordered are in the "Text" format, and in order to use this text format, it had to be parsed by a parser. As shown in Figure 2, a parser parsed the entered MIPS codes and initialized a sample-created chromosome so that for each line of the entered instruction codes, one

gene or one line was created in the chromosome structure Figure 1. The parser filled the fields in the chromosomal structure Figure 1 and set the register numbers and the dependencies between genes considering the order of all instructions in the code so that they would be used in the genetic algorithm.

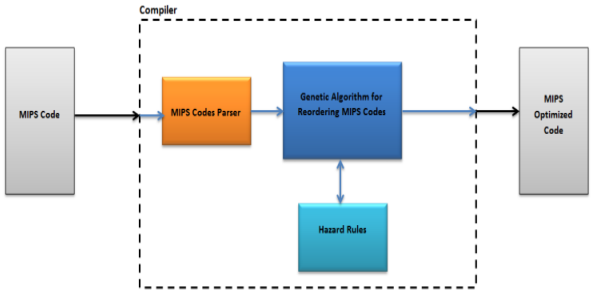

Figure 2. Pre-compiler parts and MIPS code parser

E. Creating Population

In the next stage, using the sample chromosome created by the parser described at stage 4, an array of chromosomes was created. This array indicated that the initial population and its length were equal. The initial population (i.e. the parent population which was considered an array of chromosomes) was generated Equation (2). The initialization process of the created initial population was as follows: with the aid of the sample chromosome created by the parser, the genes of each chromosome were randomly produced in the new chromosome, given the hard limitations and maintenance of permutation in genes (each gene of the sample chromosome placed only once). It is noteworthy that the purpose of considering the hard limitations was to prevent disassembling the data flow in the genes. Finally, after generating each chromosome, the amounts of Depend-D and Depend-U were again updated for each gene in the chromosome.

F. Mutation

In the presented algorithm, the mutation operator performed the major task of making intelligent changes to the genes of each chromosome to generate new children. Here, the Mutation operator was viewed as two different mutations in one chromosome. The amount of each mutation was determined with regard to the mutation rate of the parent population and therefore a new child population was created.

F.1. First Type of Mutation

In the first mutation, the line of an independent gene in each chromosome was changed. In a dependent gene, changing the gene's location in the code violates the hard limitation and disorganizes the code data flow. In this mutation, in each operation time, a gene was removed from a line and was transferred to another line, while the general data flow of the genes of corresponding chromosome was maintained. Figure 3 provides an example of the first type of mutation.

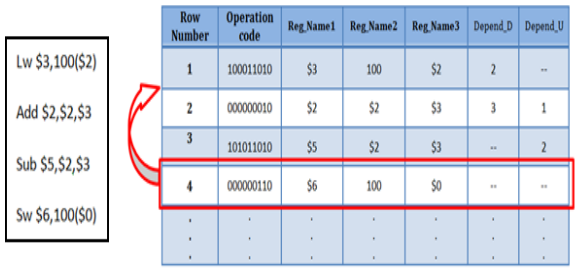

Figure 3. First type of mutation, on the left hand the related MIPS codes

F.2. Second Type of Mutation

In the second type of mutation, instead of the displacement of the line of a single gene, the line of a group of dependent genes in the chromosome was displaced Figure (4). The rate of mutation in the first mutation type was L/10 (1/10 of the genes of the relevant chromosome), and in second type was L/100 (1/100 of the genes of the relevant chromosome). It should be noted that after the occurrence of each mutation in each chromosome, the "depend-U" and "depend-D" columns of each chromosome was revised and its amounts were updated.

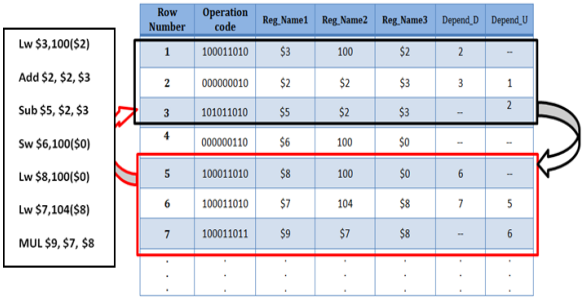

Figure 4. Second type of mutation, on the left hand related MIPS codes

G. Fitness Function

The Fitness function indicated how acceptable a chromosome was in the proposed method. Since the hard limitations were to be prevented, to violate the soft limitation (mentioned in the limitation section), a penalty was imposed on each chromosome, which was defined as follows:

$$Fitness(n) = \sum (S_i * K_i) \quad (3)$$

where n is the specified chromosome, $Fitness(n)$ shows the total number of genes which violate the soft limitation, K is the constant factor of each limitation, and S_i indicates the violation or non- violation of the soft limitation for each gene related to the specified chromosome. To obtain the weight of the violated soft limitation related to each chromosome (S_i), the defined hazard rules were used for defining the problem. As stated in code parsing and mutation Section, for each chromosome in the child society, the amounts of Depend-U and Depend-D columns in the chromosomal structure were calculated. These amounts helped to detect the hazards and stalls between the instructions. For each stall, one unit was added to the fitness amount of each chromosome.

H. Selection Function

In this selection, from the created population (i.e. the chromosomes of the child population), the initial chromosome population (or the parent population) was selected again, regarding the fitness of each chromosome. Considering different tests conducted by different methods, the best result was achieved by using the roulette-wheel [9]. In Figure 5, a sample of the selection for four chromosomes is demonstrated, in which the selection probability of each selection is defined on the basis of its amount of fitness. In fact, in this method, the amounts of fitness for each chromosome determined the selection chance, but it is noteworthy that the selection was made from a population ¼ of the size of the initial population, and the number of children, according to Equation (2), was more than that of parents, and thus there were more children to selcet from.

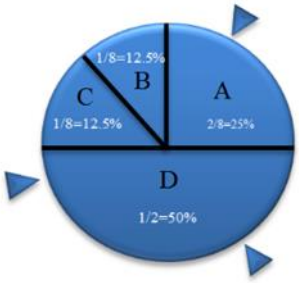

Figure 5. Roulette-wheel

V. IMPLEMENTATION

This section addresses the stages of implementation and performance of the genetic algorithm. After determining the number of parent and child population, as discussed in creation of initial population, the parents (as an initial population) were initialized in a semi-random way and then the amount of fitness for each chromosome was calculated, using Equation (4). Finally, the target test function was administered to the initial population.

The function of the target test is to investigate the amount of each chromosome to find out whether the chromosome response was optimal or not. Here, the function of the target test function was to bring the amount of the soft limitation down to zero. It should be noted that in the run-time of the target test function, the best obtained chromosome in every generation is stored as a response. In the next stage, the mutation operator was administered to the initial population (or parents).

After creating the child population, the functions of fitness amount calculation and target test were administered to the child population. In the next stage, if the target test function failed to find the intended chromosome, the selection function was administered to the child population, and the parent population was replaced by new chromosomes through the method introduced in the selection Section. The process continued until the calculated amounts for repeating the generations ended or an optimal response (i.e. zero fitness of one of the chromosomes) was produced.

VI. THE RESULT OF IN IMPLEMENTATION

The algorithm was implemented in C#.Net to examine the results. As seen in Figure 6, this software program has a code in MIPS language. After determining the mandatory stalls in this code, the program specified the number of parents' and children's chromosomes and the rate of mutation and, having run the presented genetic algorithm, it displayed the best possible results.

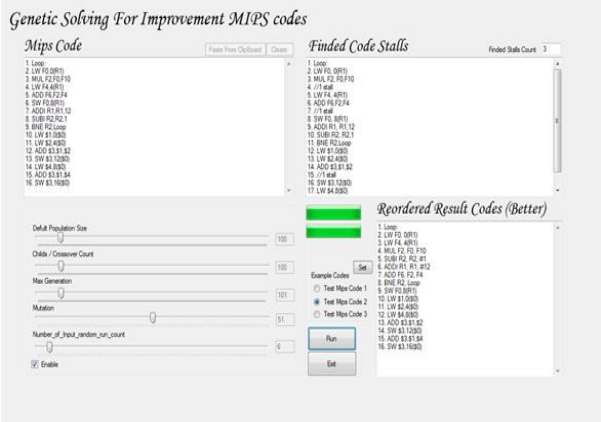

Figure 6. Implementation of the proposed algorithm

However, for different tests in the software program, determining the amounts of genetic operations such as parent and child population size was considered to be manually changeable. The proposed method was tested with different codes.

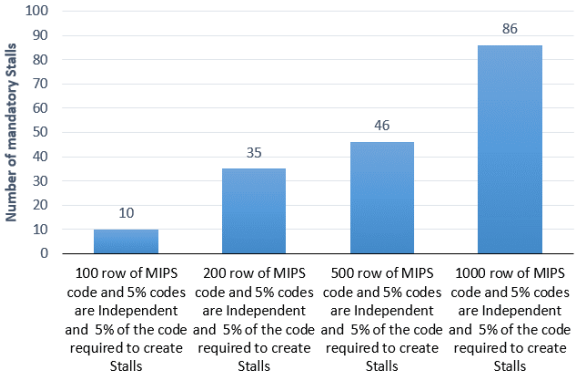

Figure 7. Results of implementation the proposed algorithm

As shown in Figure 7, some different states were tested in 4 different values respectively. In the first step, 100 rows of MIPS codes were tested, whereby 10% of the codes were improved. This tests were implemented for 1000 instructions (such implementation required that 5% of the codes be independent, and 5% of the codes make mandatory stalls). In the 1000 rows of MIPS code, the proposed algorithm could eliminate an average of 86% of the mandatory stalls between the running codes, boosting the processor's efficiency. The results showed the number of the mandatory stalls decreased by the algorithm, and as mentioned earlier, the number of mandatory stalls directly affects the run-time of the system processor.

VII. CONCLUSIONS

Nowadays, enhancing processors' speed is an important challenge in designing and manufacturing processors. Designing processors by using the pipelining method has considerably increased the speed of processors; however, due to different limitations like manufacturing expenses and different hardware-related problems in the architecture of pipeline, it is not possible, in some cases, to make use of the maximum potential of processors; for instance, due to architectural problems in processors, the occurrence of mandatory stalls between the instructions is inevitable. This paper proposed a method by which, using genetic algorithms, the instruction code of an MIPS processor was compiled in Pre-Compile way before being compiled by the MIPS processor; then the written instruction code was reordered, and the results of running, given the dependence existing in the instructions of the code, were improved as much as possible.

REFERENCES

[1] H. Ma, "The Design of Five-Stage Pipeline CPU Based on MIPS", International Conference on Electrical and Control Engineering (ICECE), pp. 433-435, Yichang, 16-18 Sept. 2011.

[2] A. Gordon Ross, F. Vahid, N. Dutt, "A First Look at the Interplay of Code Reordering and Configurable Caches", ACM Great Lakes Symposium on VLSI, pp. 416-421, 2005.

[3] Y. Chen, F. Zhang, "Code Reordering on Limited Branch Offset", Parallel Architectures and Compilation Techniques, ACM Transactions on Architecture and Code Optimization (TACO), Vol. 4, Issue 2, ACM New York, USA, June 2007.

[4] X. Huang, S.M. Blackburn, D. Grove, K.S. McKinley, "Fast and Efficient Partial Code Reordering", 5th International Symposium on Memory Management Taking Advantage of Dynamic Recompilation, pp. 184-192, 2006.

[5] J.C. Pichel , D.B. Heras, J.C. Cabaleiro, F.F. Rivera, "Performance Optimization of Irregular Codes Based on the Combination of Reordering and Blocking Techniques", Parallel Computing, Vol. 31, Issues 8-9 , pp. 858-876, 2005.

[6] J. Ruttenberg, G.R. Gao, A. Stoutchinin, W. Lichtenstein, "Software Pipelining Showdown: Optimal vs. Heuristic Methods in a Production Compiler", ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 1-11, 1996.

[7] P.H. Wang, H. Wang, R.M. Kling, K. Ramakrishnan, J.P. Shen, "Register Renaming and Scheduling for Dynamic Execution of Predicated Code", 7th International Symposium on High-Performance Computer Architecture, 2001.

[8] T. Cooper, J. Kingston, "The Complexity of Timetable Construction Problems", Lecture Notes in Computer Science, Vol. 1153, pp. 281-295, 1996.

[9] T. Weise, "Global Optimization Algorithms Theory and Application", pp. 124-129, 2009.

[10] M. Melanie, "An Introduction to Genetic Algorithms", MIT Press, Cambridge, MA, 1996.

[11] J.L. Hennessy, D.A. Patterson, "Computer Architecture: A Quantitative Approach", 3rd Edition, Morgan Kaufmann Publishers, 2002.

[12] J.R. Hauser, J. Wawrzynek, "Garp: A MIPS Processor with a Reconfigurable Coprocessor", 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 12-21, Napa Valley, CA, 16-18 Apr. 1997.

[13] M.L. Golden, "Reducing the Penalty of Branch and Load Hazards in Pipelined Microprocessors", Doctoral Dissertation, University of Michigan Ann Arbor, MI, USA, 1995.

[14] J.H. Holland, "Adaptation in Natural and Artificial Systems", University of Michigan Press, Ann Arbor, 1975.

[15] P. Mazurek, "Code Reordering Using Local Random Extraction and Insertion (LREI) Operator for GPGPU-Based Track-before-Detect Systems", Springer-Verlag, Soft Computing, Vol. 17, Issue 6, pp. 1095-1106, June 2013.

BIOGRAPHIES

Saeed Saeedvand received his B.Sc. degree in Computer Software Engineering from Islamic Azad University, Iran in 2011. Currently, he is pursuing his M.Sc. degree in Computer Software Engineering in Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran. He is working as a Lecturer in Islamic Azad University, Iran since 2012. He is Capitan of Iran SoRoBo humanoid kid size robotic team in Islamic Azad University and he has two champions in IRANOPEN 2011 and 2012 robotic games and also he has one champions in Khwarizmi AUT robotic games in 2010. His research interest is in robotic and artificial intelligence.

Ali Allahvirdizadeh received his B.Sc. degree in Computer Software Engineering from Payame Noor University, Iran in 2010. Currently, he is pursuing his M.Sc. degree in Computer Software Engineering in Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran. His current interests include parallel programming in GPU and CUDA software and sensor networks using artificial intelligence approaches.

Nemat Fathalizadeh received his B.Sc. degree in Computer Software Engineering from Payame Noor University, Iran in 2010. Currently, he is pursuing his M.Sc. degree in Computer Software Engineering in Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran. His current interests include data mining algorithms and algorithms modeling software.

Seyed Jamaledin Mojaveri received his B.Sc. degree in Computer Software Engineering from Islamic Azad University, Iran in 2011. Currently, he is pursuing the M.Sc. degree in Computer Software Engineering in Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran. His current interests include distributed systems and algorithm modeling.

Mina Zolfy Lighvan received her B.Sc. degree in Computer Engineering (hardware) and M.Sc. degree in Computer Engineering (Computer Architecture) from ECE Faculty, University of Tehran, Tehran, Iran in 1999, 2002, respectively. She received Ph.D. degree in Electronic Engineering (Digital Electronic) from Electrical and Computer Engineering Faculty, University of Tabriz, Tabriz, Iran. She currently is an Assistant Professor and works as a Lecturer in Tabriz University. She has more than 20 papers that were published in different national and international conferences and journals. Her major research interests include text retrieval, object oriented programming & design, algorithms analysis, HDL simulation, HDL verification, HDL fault simulation, HDL test tool VHDL, Verilog, hardware test, CAD tool, synthesis, digital circuit design & simulation.