Genetic Algorithm Based Bank Selection for Partitioned Memory Architectures

Liao Ximi\textsuperscript{1,2}, He Yanxiang\textsuperscript{1,2}, Chen Yong\textsuperscript{1,2}, Li Fuyang\textsuperscript{1,2}

\textsuperscript{1} School of computer, Wuhan University, Wuhan 430072, China
\textsuperscript{2} State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China

ximil@whu.edu.cn, yxhe@whu.edu.cn, cyong1000@163.com, dxfly@yahoo.com.cn

Abstract—For partitioned memory architectures, bank selection is a challenging problem. As we all know, it is a NP-hard problem. To solve this NP-Hard problem, people have studied in this area for many years. But we still have some distance away from the optimal solution. In this paper, I will introduce how to solve the bank selection problem by using Genetic Algorithm\textsuperscript{[1][2]}. Though we still can't give the best solution, the approximate optimal one can be found through this method. And the main attribution of this paper is not only giving an efficient bank selection algorithm, but a new idea of the bank selection for partitioned memory architectures.

Keywords: partitioned memory architectures, bank selection, Genetic Algorithm.

1 Introduction

For partitioned memory architectures, bank selection is a challenging problem. And it is of vital importance for optimizing compilers. Bank switching is a common technique to increase the size of code and data memory without extending the address buses of CPU. The address space is partitioned into memory banks, and the CPU can only access one bank at a time, which is called the active bank. A bank selection instruction is issued to switch between banks. In this paper, we want to find the right data banks that the variables of a program are placed. We did this because it can affect many aspects, such as runtime, low power, small code size or a combination of these parameters. And we concentrate on the code size. That is to say, we used the genetic algorithm to present an optimization that inserts a minimum number of bank selection instructions in the program to guarantee that banked memory is used completely.

For instance, there is a program which has m variables. They have been represented as P (Vi, value) (0<i<=m). In which Vi is the identifier of a variable and value represents the size of the variable. And we also have an interference graph G to show the interaction of different variables. We assume there are 2k banks in our architecture. They were signed from 0 to 2k- 1. So our work in this paper is allocating these m variables to the 2k banks and it should satisfy the two rules below:

1) All the variables should be allocated and for every bank, the total size of the variable allocated in this bank can’t be bigger than it is capacity.

2) We should try the best to minimize switching between the banks so that a minimum number of bank selection instructions are used.

The remainder of this paper is organized as follows. In section two, we introduce the conception and principle of genetic algorithms and analyze why we choose this method to solve the bank selection problem. In section three, we introduce the algorithm used in this paper in detail. And then in section four, we will give a small example to show how this algorithms worked. Some related work will be given in section five. We round off the article with a conclusion and list some further work as well in section six.

2 Background

A genetic algorithm is a search heuristic that mimics the process of natural evolution. It is usually used to generate useful solutions to optimization and search problems\textsuperscript{[3]}. Genetic algorithm is a simulation of Darwinian natural selection of genetic selection and biological evolution process model. In a genetic algorithm, it expresses the solution of a problem as "chromosome" and through the change of the "chromosome" from generation to generation which including reproduction, crossover and mutation, eventually converge to the "best adapted individuals".

We can then see that the principle of genetic algorithms is simple:

1) Encoding of the problem in a binary string.

2) Random generation of a population. It includes a genetic pool representing a group of possible solutions.

3) Reckoning of a fitness value for each subject. It will directly depend on the distance to the optimum.

4) Selection of the subjects that will mate according to
their share in the population global fitness.

5) Genomes crossover and mutations.

6) And then start again from point 3.

The steps were shown in Figure 1.

1) **Initialization:** Many individual solutions are randomly generated to form an initial population. The population size depends on the nature of the problem. We encode candidate solutions in binary as strings of 0s and 1s. The different combinations of structural data strings constitute a different individual.

2) **Estimate:** We use the fitness evaluation function to show the advantages and disadvantages of each of the individual or the solution. The fitness evaluation function is defined in different ways according to different issues.

3) **Selection:** We have to select good individuals from the current population according to the fitness evaluation to be the "parents". To do this they have the opportunity to breed generations.

4) **Crossover:** The crossover operation is the most important genetic algorithm operation. Two selected "parent" could generate a new individual by crossover operation. It embodies the idea of information exchange.

5) **Mutation:** At first, we choose an individual randomly and change a random code of it in a certain probability. The mutation operation provided an opportunity to generate new individual.

6) **Stop condition:** If the stop condition is met, we choose the individual with the largest fitness in the current population to be the final result and terminate the algorithm.

As we introduced, the genetic algorithm is a good choice to solve optimization problems. The bank selection problem is an optimization problem which asks for the optimal selection of the bank to make sure that the variables use minimum space of the CPU. Therefore, use the genetic algorithm to solve this problem is a good choice.

3 Genetic algorithms based bank selection

As we introduced in section one, we need to solve the problem of bank selection by genetic algorithm. In the problem, m variables need to be assigned to 2k banks. At the same time, we should try to use the least amount of space and bank selection instructions. Now we will introduce the algorithm in detail.

3.1 The individual coding

Because we have 2k banks, for every variable, there are 2k kinds of placement. We will encode the individual according to the bank of a variable.

For example, if there are 5 variables and 4 banks, we need two bits to represent the selected bank of a variable. That is to say in the total, ten bits will be used to encode and there are 4^5 kinds of encodings. Assuming that a selection as shown in figure 2 below.

![Figure 2: the selection of an example](image-url)
can encode it as 00 00 11 01 10.

To solve the problem in this paper, we need \( m^k \) bits and there will be \( 2^{mn} \) kinds of encodings.

### 3.2 Population initialization

In this paper, we will define the size of the initial population to \( S = m^k \cdot 2k \). We will use an array to store these numbers. Then use the algorithm as shown below randomly generated \( S \) individuals of the initial population.

1. Initialize array, make array \([i] = 0\);
2. for ( int \( i=0; \ i<=S; \ i++\) ){
3. randomly generate a \( m^k \) bits number \( s \);
4. Make array\([i]=s;\} 
5. exit

![Figure 3 a simple algorithm for initiation](image)

### 3.3 Fitness function

The fitness function is used to calculate the fitness of individual. In this paper, the fitness value is the amount of space it will take for an individual. At first, we should calculate the bank selection instructions according to the interference graph \( G \). We use \( n \) to records the number of bank selection instructions. At first, we set \( n=0 \). Then, we traverse the graph \( G \). If there is an edge from node1 to node2, the two nodes are in different banks, \( n=n+1 \). Then the total size of the selection is calculated using the formula (1) as follows:

\[
f(x) = \sum_{i=0}^{m} V_i + n
\]  

(1)

As shown above, the total size is the number of bank select instructions plus the whole size of the variables.

However, due to the special problems involved in this paper, we should do another thing before we calculate the fitness of individual. We need to first determine the reasonableness of such coding. It is very easy, and we just need to calculate the sum size on each bank. If it bigger than the size of the bank size, the selection must be given up.

### 3.4 Parent selection

In this paper, roulette wheel selection method \(^4\) was used to select the parents. Roulette wheel selection method simulates gambling game roulette. A roulette wheel is divided into \( N \)-sectors. Each sector represents a chromosome of the population. The area of each sector is proportional to the fitness value of the chromosome. In order to select the individuals from the population, we assuming there is a pointer to the wheel, turn the wheel, when the wheel stopped, the pointer points to the selected chromosome. Therefore, the greater the fitness value of a chromosome indicates the larger of the chromosome area and the greater likelihood of it to be chosen. The specific process is as follows:

1. Calculate the sum of the fitness value of all the individuals of the population using formula (2):

\[
Total = \sum_{i=0}^{n-1} f(i)
\]  

(2)

2. Calculate selection probability for each individual using formula (3):

\[
nSelPr(o(k)) = \frac{f(k)}{Total} \quad (k = 0, \ldots, n-1)
\]  

(3)

3. Calculate the sum of probability of all the individuals using formula (4):

\[
nToPr(o(k)) = \sum_{i=0}^{1} nSelPr(o(i)) (k = 0, \ldots, n-1)
\]  

(4)

4. Turn the wheel \( N \) times. We use a random number in \([0, 1]\) to simulate the location pointed when the wheel stopped. If \( r \leq nToPr(o(0)) \), it means the pointer pointed the first sector and we choose the first individual. Generally if \( nToPr(o(k-1)) \leq r \leq nToPr(o(k)) \), it shows it pointed the \( k \)-sector, and then select the \( k \)-chromosome.

### 3.5 Crossover

In this article, we choose Single-point crossover. It is a simple algorithm, but enough for our experiments using. The Single-point crossover algorithm uses two parent bodies to crossover and produce two offspring individuals. Assuming the length of a binary bit string is \( L \). In this paper, it is \( S \). It generates random integer \( pos \) to be the crossover point. Then we exchange the substring of two parent bodies which is the right of crossover point and generate two offspring individuals. The schematic diagram of single-point crossover is in figure 4.

![Figure 4 Schematic diagram of single-point crossover](image)

### 3.6 Mutation

We use a pre-set probability \( pm \) to mutate the chromosome genes of population. If a gene was chosen to mutate, we turned 1 to 0 and 0 to 1. For every gene of the population, we generate a random number \( r \). If \( r \geq pm \), then the gene mutation.
3.7 Stop conditional

The user type in the number of iteration Num. we will test whether the number of iteration is bigger than Num. If not, it will go on to do the iteration. After every time of iteration, we should find the best solution of the current population and compare it with the previous one. We will store the better one until we get to Num. Then we quit and return the best solution.

4 Experimental result

To test the effect of the algorithm, we did an experiment on an embedded system. It is HR6P serial micro control unit designed by an integrated circuit company in China, which owns 8-bits RISC instruction set. In this paper, we focus on the method of bank selection based on genetic algorithm. The user cases are designed especially for the compiler. We will compare the normal algorithm with the genetic algorithm with the number of bank selection instructions.

We test our genetic algorithm on a real system. The experimental environment is shown in Table 1.

<table>
<thead>
<tr>
<th>Table 1 the experimental environment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CPU</strong></td>
</tr>
<tr>
<td>Intel Pentium 4 3.06G</td>
</tr>
<tr>
<td><strong>operating system</strong></td>
</tr>
<tr>
<td>Windows XP sp3</td>
</tr>
<tr>
<td><strong>internal memory</strong></td>
</tr>
<tr>
<td>DDR2, 1G</td>
</tr>
<tr>
<td><strong>develop environment</strong></td>
</tr>
<tr>
<td>Visual Studio 2005</td>
</tr>
<tr>
<td><strong>Target Chip</strong></td>
</tr>
<tr>
<td>RISC</td>
</tr>
<tr>
<td><strong>Test case</strong></td>
</tr>
<tr>
<td>Cooperation provided</td>
</tr>
<tr>
<td><strong>Test board</strong></td>
</tr>
<tr>
<td>the actual system</td>
</tr>
</tbody>
</table>

As we shown in Figure 5, we choose 17 cases to evaluate the feasibility and effectiveness of the algorithm. The x-axis present the cases used and the y-axis is the number of instruction used for the bank’s selection. From these cases, we can see that it reduces about 12.94% on average and 20.66% for the fifteenth case. So the genetic algorithm is useful to solve the problem of bank selection. After the operation of this algorithm, the number of bank selection instructions significantly reduced.

5 related work

As we all know, many works have been done for optimizing bank selection instructions. Bernhardt et al. [5] formulated the problem of optimizing bank selection instructions for multiple goals as a form of Partitioned Boolean Quadratic Programming (PBQP). They assume that the variable had been assigned to specified banks. Liu Tiantian et al.[6] claimed they had integrated variable partitioning into optimizing bank selection instructions. With the analysis of code patterns, they placed the emphasis on the positions for inserting bank selection instructions. Yuan Mengting et al. [7] take the variable partitioning on shared memory. Especially, Li Qingan et al. [8] present an algorithm to reduce the overhead of page switching. To pursue small code size, they place the emphasis on the selection of functions into suitable pages with a heuristic algorithm. Many other works [8][10][11] about variable partitioning focused on DSP processors, where parallelism and energy consumption attracted the main attentions. There is also work to improve the overall throughput for MPSoC architecture by variable partitioning [12].

6 Conclusion and future work

In this paper, we present the genetic algorithm to optimize the bank selection. Our experimental results showed it achieved a great improvement with respect to code size. However, there is still much work to be done to improve this algorithm. In this algorithm, we have not considered a matter of time. And in the future, we will make improvements in this area. Finally, complete content and organizational editing before formatting. Please take note of the following items when proofreading spelling and grammar:

7 References


