A Parallel Tree Based Strategy for 3-way Interaction Testing

Mohammad F. J. Klaib
College of Computer Sciences and Engineering, Taibah University, Madina P.O.Box 344, Kingdom of Saudi Arabia
mom_klaib@yahoo.com, mklaib@taibahu.edu.sa

Abstract

Software testing is one of the most important phases of the software development life cycle. However, software testing is traditionally seen as a difficult and time consuming activity that is hard to embed in the software development process. Software and hardware testers concentrate on how to minimize the testing time, as well as ensure that the system is also tested well and made acceptable. Basic combinatorial interaction (i.e. pairwise or 2-way) testing has been one of the commonly used methods in achieving the above goal with 50-97 percent of errors detection. However, empirical evidence has proved that the 2-way interaction testing is a poor strategy for testing highly interactive systems. Therefore there is a need for going beyond pairwise testing to uncover these errors. To speed up the process of solving problems, researchers have applied parallel algorithms to various large computationally expensive optimization problems and have succeeded in solving these problems in an acceptable time. Therefore, in this paper we have enhanced our previous strategy “A tree based strategy for test data generation and cost calculation for pairwise combinatorial interaction testing” to work effectively in parallel and to support a 3-way interaction testing. The correctness of the strategy has been proved, and the performance evaluation shows the efficient of the strategy in reducing test size.

1. Introduction

In software development life cycle (SDLC), Testing plays an importance role, which helps to improve the quality, reliability and performance of systems. To insure a high quality enhancement of software and hardware products, a good testing is required, where testing is the process of executing programs or systems with the goal of finding errors thereby assuring the correctness, completeness and quality of developed hardware or software systems [2, 3, 8, 19, 25]. Lack of testing can lead to disastrous consequences including loss of data, resources, and sometimes lives [5, 18]. A well-tested product or service is necessity to ensure customer’s satisfaction. However, exhaustive testing is
unaffordable due to combinatorial explosion problem [10-13, 20, 24] which occurs when a small increase in the number of elements that can be combined increases the number of combinations to be tested. To illustrate this problem consider testing the IC7489, a 16 by 4 random access memory [23]. Pin assignment of inputs is shown in Figure 1, we have 10 parameters: four address inputs, four data inputs, memory enable and write enable. Each of these pins has two values L or H.

![Fig. 1. IC7489, 16 X 4 random access memory](image)

Table 1. Function table for 7489.

<table>
<thead>
<tr>
<th>ME</th>
<th>WE</th>
<th>Operation</th>
<th>Data Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>L</td>
<td>Write</td>
<td>Complement of data inputs</td>
</tr>
<tr>
<td>L</td>
<td>H</td>
<td>Read</td>
<td>Complement of selected word</td>
</tr>
<tr>
<td>H</td>
<td>X</td>
<td>Disable</td>
<td>High impedance</td>
</tr>
</tbody>
</table>

In order to test this IC exhaustively there are 210 (i.e. 1024) combinations of test cases which are to be evaluated. If the time required for one test case to be evaluated is 2 minutes, then it would require nearly 4.26 working days for a complete test to be done for only one IC. If 1000 such IC’s needs to be tested then imagine the time and resources consumed for exhaustive testing to be done. Therefore, it is very clear that combinatorial explosion is a serious issue which has to be considered, and testing most hardware and software systems face this problem. Combinatorial explosion in testing may occur for configurable systems. When systems under test have many configuration parameters, each with several possible values, testing each configuration is sometimes infeasible.

Thus, to bring a balance between exhaustive testing and lack of testing, combinatorial interaction testing [28-42] has demonstrated to be an effective technique to achieve reduction of test suite size.

Although 2-way interaction testing can detect 50-97 percent of errors [1, 7, 9, 18, 21, 26, 27], empirical evidence has proved that 2-way interaction testing is a poor strategy for testing highly interactive systems [43, 18]. Therefore there is a need for going beyond pairwise testing to uncover these errors.

Concerning the issue of speed up gained from parallelization, researchers have applied parallel algorithms to various large computationally expensive optimization problems and have succeeded in solving these problems in an acceptable time [22, 44-46]. Therefore, in this paper we have enhanced our previous strategy “A tree based strategy for test data generation and cost calculation for pairwise combinatorial interaction testing” to work effectively in parallel and to support 3-way interaction testing. The correctness of the strategy has been proved, and the performance evaluation shows that the strategy is efficient in test size reduction.
2. Methodology

Two algorithms were designed; a parallel tree generation algorithm which generates the lists of test cases and a cost calculation algorithm which is used to construct in parallel the test suite with minimum number of test cases.

Algorithm 1: Tree Generation Algorithm

```
Input: A set of parameters and their possible values
Output: Tree generation
Begin
    X = number of values of first parameter p1

    {For the first parameter p1}
    Ti=Vi, where i=1,2,3,……..X ; parameter p1 has X values
    If N=1 then stop and exit
    Create X threads with unique thread ids. Assign each Ti to a separate child thread and execute all the child threads in parallel
    Wait for the termination of all the threads.
End
```

Algorithm 2: Cost Calculation and Test Suite Generation

```
Input: Lists of test cases. Each list holds the test cases generated by the tree in one particular branch of that tree.
Output: T-way test suites with minimum number of test cases
Begin
    Tempb = Tb (where b is the number of lists of test cases)
    X = number of values of parameter p1
    B=min (Value(p1), Value(p2), ……Value(pn))
    For i = 2 to N-1 do
        Begin
            Generate the i-way covering array for the given parameters.
            Wmax = n! / ((i!) * ( (n-i)! ) ) // N – is the number of parameters
            Let T' be an empty set where i-way test suites are stored.
            For a = 1 to B do
                Begin
                    Testa = concatenate the ath values of all the parameters to form a test case.
                    End
                For each Testa do
                    Begin
                        Delete all the T-way combinations that Testa covers in the covering array
                        Delete Testa from the Ti Lists
                        T' = Testa
                    End
            Creates a set of temporary lists Yi corresponding to the Ti lists, where i= 1,2,…..X, X is the number of values of parameter p1 or the number of lists.
            Create X threads with unique thread ids. Assign every child thread Th with one Ti lists, the corresponding Yi lists, i value and Wmax value, and execute all the child threads in parallel.
            Wait for the termination of all the child threads.
            Store the i-way test suite generated in the list T'
            Tb = Tempb
            End
        End
End
```
The first algorithm constructs the tree based on the parameters and values given. It constructs every branch of the tree in parallel. The number of branches the tree has depends on the number of values of the first parameter i.e. if the first parameter has 3 values then the tree also would have 3 branches. Therefore every branch construction starts by getting one value of the first parameter i.e. branch T1 gets the first value, T2 gets the second value and so on. After the base branches are constructed one child thread is assigned to every branch and the further construction takes place in a parallel manner. Each of the branches considers all values of all the other parameters two, three,…..N where N is the total number of parameters. All the branches consider the values of the parameters in the same order.

Once the parallel tree construction is over, at first, the algorithm starts by constructing the covering array, for all possible 3-way combinations of input variables. Then we are ready with all the test cases to start the parallel iterative cost calculation (algorithm 2). In this strategy the cost of the leaf nodes in each of the lists are calculated in parallel in order to reduce the execution time. The cost of a particular test case is the maximum number of 3-way combinations that it can cover from the covering array.

The second algorithm starts to include all tree branches, which might definitely give the maximum cost (Wmax) into the test suite. Then these test cases are deleted from the tree branches lists, and the corresponding 3-ways covered by it in the covering array are also deleted. In the next step, the main thread in the algorithm invokes a number of child threads equal to the number of values of the first parameter and calculates the cost of all the test cases in each of the branches in a parallel fashion. Each child thread stores all the test cases with the Wmax value from its corresponding branch into a separate sub-list. The child thread that finishes calculating the cost of all the test cases in its branch first locks the covering array. This thread then looks into its sub-list and includes the test cases stored in it into the test suite only after confirming that the test case definitely has the maximum cost or Wmax value. Then the test cases included in the test suite are deleted from the tree branches list and sub-list, and the corresponding 3-ways that these cover are deleted from the covering array.

The other threads wait in a queue until the execution of the first thread is over, after which these threads resume their execution in the order in which they are queued. These threads on resumption re-evaluate the test cases in their sub-list to confirm that these test cases have the Wmax value before including these into the test suite. Thus in the first iteration all the test cases with the maximum Wmax value from all the branches are included in the test suite. Now the Wmax value is decremented by one and the same parallel execution of all the threads continue until all the pairs in the covering array are covered.

3. Results

To prove the correctness of the above algorithms and to illustrate how the algorithms work consider the following simple system with parameters and values:

- Parameter A has two values A1 and A2
- Parameter B has one value B1
- Parameter C has three values C1, C2 and C3
- Parameter D has two values D1 and D2

To illustrate the minimum test suite construction of 3-way combinatorial interactions testing using our algorithm, for the system mentioned. The algorithm starts constructing the test-tree by considering the first parameter. As the first parameter has two values the tree is said to have two main branches with the first branch using A1 and the second branch using A2. Then each of the branches is constructed in parallel by considering all the values of the second parameter, then the third and fourth and so on. When the branches are fully constructed the leaf nodes gives all the test cases that has to be considered for cost calculation. Since all of the branches are constructed in parallel a significant reduction in time will be there. Figure 2 below shows how the test-tree would be constructed. The test cases generated by the first branch are stored in the lists T1 and the test cases generated by the second branch are stored in T2 respectively. i.e. (A1,B1,C1,D1), (A1,B1,C1,D2), (A1,B1,C2,D1), (A1,B1,C2,D2), (A1,B1,C3,D1), (A1,B1,C3,D2) are stored in T1, and (A2,B1,C1,D1), (A2,B1,C1,D2), (A2,B1,C2,D1), (A2,B1,C2,D2), (A2,B1,C3,D1) and (A2,B1,C3,D2) are stored in T2.
Once the parallel tree construction is over we are ready with all the test cases to start the parallel iterative cost calculation based on algorithm 2. In this strategy the cost of the leaf nodes in each of the lists are calculated in parallel in order to reduce the execution time. The cost of a particular test case is the maximum number of 3-way combinations that it can cover from the covering array. At First, the algorithm starts by constructing the covering array, for all possible 3-way combinations of input variables, i.e. . [A, B, C], [A, B, D], [A, C, D] and [B, C, D], for the example in Fig. 2. The covering array (in Table 2) for the above example has 28 3-way interactions which have to be covered by any test suite generated, to enable a complete 3-way interaction testing of the system.

Once the covering array is generated (Table 2), the algorithm starts to include all tree branches, which might definitely give the maximum Wmax cost into the test suite. For the above example all the test cases which are included in the test suite are identified in three iterations and there are 12 such test cases shown in Table 3.

<table>
<thead>
<tr>
<th>Table 2</th>
<th>3-way interaction covering array.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, B, C</td>
<td>A, B, D</td>
</tr>
<tr>
<td>A1, B1, C1</td>
<td>A1, B1, D1</td>
</tr>
<tr>
<td>A1, B1, C2</td>
<td>A1, B1, D2</td>
</tr>
<tr>
<td>A1, B1, C3</td>
<td>A2, B1, D1</td>
</tr>
<tr>
<td>A2, B1, C1</td>
<td>A2, B1, D2</td>
</tr>
<tr>
<td>A2, B1, C2</td>
<td>A1, C3, D1</td>
</tr>
<tr>
<td>A2, B1, C3</td>
<td>A1, C3, D2</td>
</tr>
<tr>
<td>A2, C1, D1</td>
<td>A2, C1, D2</td>
</tr>
<tr>
<td>A2, C1, D2</td>
<td>A2, C1, D1</td>
</tr>
<tr>
<td>A2, C2, D1</td>
<td>A2, C2, D2</td>
</tr>
<tr>
<td>A2, C3, D1</td>
<td>A2, C3, D2</td>
</tr>
</tbody>
</table>
To evaluate the efficiency of the proposed strategy for 3-way test data generation, we consider two different configurations. The first configuration has non-uniform parametric values. The second configuration has a uniform number of values for all parameters. The two systems configurations used are summarized as follows:

- S1: 5 parameters with 3, 2, 1, 2 and 2 values respectively.
- S2: 4 3-valued parameters.

Also to evaluate the efficiency of the proposed strategy in test size reduction by comparing its results with available testing strategies which is also based on parametric interactions. We have identified the following existing strategies that support interaction testing: AETG [4,6], IPO[7], AllPairs [14], TConfig [15], Jenny [16], TVG [17], G2Way[1], GTWay [18] tool. We consider the same two configurations as described above.

Table 3  Generated test suite for 3-way combinatorial interaction.

<table>
<thead>
<tr>
<th>Test Case No.</th>
<th>Test Case</th>
<th>Iteration/Child Thread No.</th>
<th>Max Weight</th>
<th>Covered pairs</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>A1,B1,C1,D1</td>
<td>1/1</td>
<td>4</td>
<td>[A1,B1,C1][A1,B1,D1][A1,C1,D1][B1,C1,D1]</td>
</tr>
<tr>
<td>T4</td>
<td>A1,B1,C2,D2</td>
<td>1/1</td>
<td>4</td>
<td>[A1,B1,C2][A1,B1,D2][A1,C2,D2][B1,C2,D2]</td>
</tr>
<tr>
<td>T8</td>
<td>A2,B1,C1,D2</td>
<td>1/2</td>
<td>4</td>
<td>[A2,B1,C1][A2,B1,D2][A2,C1,D2][B1,C1,D2]</td>
</tr>
<tr>
<td>T9</td>
<td>A2,B1,C2,D1</td>
<td>1/2</td>
<td>4</td>
<td>[A2,B1,C2][A2,B1,D1][A2,C2,D1][B1,C2,D1]</td>
</tr>
<tr>
<td>T5</td>
<td>A1,B1,C3,D1</td>
<td>2/1</td>
<td>3</td>
<td>[A1,B1,C3][A1,C3,D1][B1,C3,D1]</td>
</tr>
<tr>
<td>T12</td>
<td>A2,B1,C3,D2</td>
<td>2/1</td>
<td>3</td>
<td>[A2,B1,C3][A2,C3,D2][B1,C3,D2]</td>
</tr>
<tr>
<td>T2</td>
<td>A1,B1,C1,D2</td>
<td>3/1</td>
<td>1</td>
<td>[A1,C1,D2]</td>
</tr>
<tr>
<td>T3</td>
<td>A1,B1,C2,D1</td>
<td>3/1</td>
<td>1</td>
<td>[A1,C2,D1]</td>
</tr>
<tr>
<td>T6</td>
<td>A1,B1,C3,D2</td>
<td>3/1</td>
<td>1</td>
<td>[A1,C3,D2]</td>
</tr>
<tr>
<td>T7</td>
<td>A2,B1,C1,D1</td>
<td>3/2</td>
<td>1</td>
<td>[A2,C1,D1]</td>
</tr>
<tr>
<td>T10</td>
<td>A2,B1,C2,D2</td>
<td>3/2</td>
<td>1</td>
<td>[A2,C2,D2]</td>
</tr>
<tr>
<td>T11</td>
<td>A2,B1,C3,D1</td>
<td>3/2</td>
<td>1</td>
<td>[A2,C3,D1]</td>
</tr>
</tbody>
</table>

Table 4  TBGCC 3-way results.

<table>
<thead>
<tr>
<th>System</th>
<th>Exhaustive number of test cases</th>
<th>TBGCC-3 3-way Test suite size</th>
<th>3-way Reduction %</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>24</td>
<td>16</td>
<td>33.33%</td>
</tr>
<tr>
<td>S2</td>
<td>81</td>
<td>31</td>
<td>61.73%</td>
</tr>
</tbody>
</table>

Table 5  Comparison of PTBGCC-3 with other strategies for generated 3-way test suite size.

<table>
<thead>
<tr>
<th>System</th>
<th>TConfig</th>
<th>Jenny</th>
<th>TVG</th>
<th>GTWay</th>
<th>PTBGCC-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S1</td>
<td>NS</td>
<td>NS</td>
<td>13</td>
<td>12</td>
<td>16</td>
</tr>
<tr>
<td>S2</td>
<td>32</td>
<td>34</td>
<td>33</td>
<td>32</td>
<td>31</td>
</tr>
</tbody>
</table>

NS – Not Supported
4. Discussion

In this paper, we have enhanced our previous strategy “A tree based strategy for test data generation and cost calculation for pairwise combinatorial interaction testing” to work effectively in parallel and to support a 3-way interaction testing.

The correctness of the proposed strategy has been proved in section 3. Table 2 showed how to construct the generated tree in parallel. Furthermore, Table 3 showed how the cost calculation algorithm works iteratively to generate the test suite with the order in which the various test cases are actually included, also showed the covered 3-ways interactions in each test case, and they showed that all 3-ways combination have been covered in the generated test cases, which prove the correctness of the proposed strategy.

Referring to experiments and evaluation part in section 3. The second column in Tables 4, showed the exhaustive number of test cases for each system, and the third column displayed the generated test suite size for 3-way interactions using the proposed algorithm. The last column showed the percentage of reduction achieved. The presented evaluation showed that the proposed strategy is efficient in test size reduction for uniform and non-uniform parameter’s values.

Referring to comparison part in section 3. Tables 5 compared the generated test suite size of the proposed strategy with other strategies. Note that the minimum test suite size is highlighted. The proposed strategy (PTBGCC-3) has produced the best results for the second system (S2), while in the first system (S1) the Gtway strategy has performed better than PTBGCC-3. However, test reduction is a NP-complete problem, for which no strategy may perform the best for all cases.

The proposed algorithms could be extended in future to work for general higher multi-way (i.e. 4, 5, …N) and new different models for shared and distributed memory architectures can be used.

References
15. TConfig, http://www.site.uottawa.ca/~awilliam/


