SOME EXPERIMENTS IN GLOBAL MICROCODE COMPACTION

Bogong Su and Shiyuan Ding

Department of Computer Science and Technology
Tsing Hua University
Beijing, China

Abstract

Global microcode compaction is an open problem in firmware engineering. Although Fisher's trace scheduling method may produce significant reductions in the execution time of compacted microcode, it has some drawbacks. There have been four methods, Tree, SRDAG, ITSC, and GDDG, presented recently to mitigate those drawbacks in different ways.

The purpose of the research reported in this paper is to evaluate these new methods. In order to do this, we have tested the published algorithms on several unified microcode sequences of two real machines and compared them on the basis of the results of experiments using three criteria: time efficiency, space efficiency, and complexity.

1. Introduction

One of the critical issuses in developing a high level microprogramming language is how to generate efficient microcode; the major way is through microprogram compaction. Although the local compaction problem is considered to be essentially solved and several methods were reviewed in [1,7], global compaction is still an open problem in firmware engineering.

There are two kinds of global microcode compaction methods, the block-oriented methods proposed by Tokoro[11], Poe[9], and Wood[12], and the trace-oriented method proposed by Fisher[2]. Since Fisher's trace scheduling may produce significant reductions in the execution time of compacted microcode, it has been regarded as the most promising global technique[3]; however, extra space may be sometimes required during bookkeeping and the efficacy of microcode loop compaction is lower than that of hand compaction[3,10]. Hence, there have been four methods, Tree[6], SRDAG[8], ITSC[10], and GDDG[5], presented recently to mitigate those drawbacks in different ways. Although these new algorithms have each been tested by their own author, it is very difficult to compare their performance because of the big differences among the test microcodes in the literature. The purpose of the research reported in this paper is to evaluate these new methods of global microcode compaction. In order to do this, we have tested the published algorithms on several unified microcode sequences of two real machines and compared them on the basis of the results of experiments using three criteria: time efficiency, space efficiency, and complexity.

2. Algorithms

A. Tree algorithm

The Tree method has been developed by Lah[6] for reducing the space requirement of the compacted result. It is like trace scheduling but the schedule range is limited. This method partitions a given microcode into subtrees -- top trees and bottom trees -- and applies the idea of trace scheduling to each subtree separately; the root of a top(bottom) tree is the entry or rejoin(exit or branch). The movement of microoperations is not allowed to cross the root of a subtree; thus the number of copies can be reduced and the complexity of the algorithm decreased.

```
Algorithm: Tree
  1) Identify_Blocks and Live_Reg_Analysis;
  2) Partition_Graph(into top trees);
  3) FOR each tree DO
       WHILE (uncompacted trace) DO
         BEGIN
           Pick_A_Trace;
           List Scheduling;
           Bookkeeping;
         END;
  4) Partition_Graph(into bottom trees);
  5) Perform 3) until all blocks are compacted;
```

B. SRDAG algorithm

The SRDAG method is an extention of trace scheduling[8]. Linn indicated that the trace should be reselected after the first block has been compacted, the rule R4 for moving microoperation[2](as shown in Fig.1) should be used and the rule R6 should be used more generally. The list scheduling is applied on a single rooted directed acyclic graph (SRDAG) instead of a path and only the root block of the SRDAG is compacted in each iteration. If a particular microoperation is free at the top of several blocks in the SRDAG, the SRDAG algorithm may move up such duplicated microoperations with rule R4.

```
Algorithm: SRDAG
  1) Live_Reg_Analysis;
  2) WHILE (uncompacted block) DO
       BEGIN
         Select_SRDAG;
         Build_DAG;
         Compact_Root_SRDAG;
         Modify_Graph and Live_Reg_Analysis;
       END;
```

C. ITSC algorithm

The ITSC method is an improvement of trace scheduling [10]. We have proposed the following:

1) A modified rule set for moving microoperation;
2) An Improved Trace Scheduling (ITS) algorithm for reducing extra copies and keeping time efficiency:
   a) Patition the microoperations of a path into two parts;
   b) Compact the MAIN part consisting of microoperations in critical path of DAG with list scheduling;
   c) Arrange each remaining microoperation according to its type and characteristic in flow graph to prevent overmoving and generating unnecssary copies;
3) An URCR algorithm for loop compaction:
   a) Unroll the loop body to creat two bodies.
   b) Compact the unrolled loop body with list scheduling;
   c) Find a new loop body, delete the redundant microoperations, and reroll the loop;

D. GDDG algorithm

Isoda's [5] algorithm depends upon a generalized data dependency graph (GDDG) which integrates the DAG with two primitives Y-joint and Y -joint, representing the joining and forking of control flow respectively.

Algorithm: GDDG

1) Building of the initial GDDG;
2) Flow Analysis to find live registers and loops;
3) Transformation of the GDDG with five rules;
4) Compaction of microoperations:
   a) Classifying microoperations according to their status on the GDDG;
   b) Determining the compaction order of basic blocks depending on frequency comparisons for adjacent basic blocks;
   c) Compacting microoperations limited within some basic blocks with list scheduling;
   d) Accommodating remaining microoperations in the vacant fields of the existing microinstruction; otherwise, putting them into the less frequently executed adjacent basic blocks;

The first three of the above mentioned algorithms are improvements of trace scheduling and the last one is essentially a block-oriented algorithm.

3. Experiments

We have chosen four floating arithmetic microcode sequences of the PUMA and five complicated microcode sequences of a VAX_like machine as examples. PUMA, which is a horizontally microprogrammed emulator of the Control Data 6600, has a single phase microinstruction cycle and a relatively wide microinstruction(81 bits) with minimal encoding. It is capable of three concurrent data operations: a load or store to the register file, a main arithmetic unit operation, and an exponent arithmetic unit operation [4]. The VAX_like machine has a wide microinstruction(96 bits). It is capable of four concurrent operations: an arithmetic logic unit operation, an exponent arithmetic logic unit operation, shifting and unpacking in the data section, and an address operation.

To provide the input to the scheduler we rewrote the selected sequences of production microcode as sequential microcode of these two machines. This was not just a process of serializing parallel code. We went back to flow charts for the arithmetic operations and tried to write the clearest possible sequential code, without regard for subsequent compaction.

The characteristics of the sequential microcodes are shown in Table 1 and Table 2. The features of PUMA sequential microcods are: 1) The basic blocks are rather short; the average number of microoperations in a basic block is two. 2) The movement range of microoperartion is rather wide. Whereas, the features of the VAX_like sequential microcodes are: 1) The average length of basic blocks is almost twice as long as PUMA's. 2) The movement range of microoperation is narrow. 3) There are microsubroutine calls and case-type multiway branches in those microcodes. One of the reasons why we selected these two real machines with quite different features as our examples was to test the applicable scope of global microcode compaction methods.

Experiments were carried out with our microcode compaction system MCS, written in PASCAL and running on a PDP-11/23. Since we had not implemented the whole algorithm of these methods expect for ITSC, some simulation with MCS system was necessary. In addition to dealing with the general model of compaction, the MCS system has special capabilities for : 1) Permitting the valid reading of a register up to the time that register writes occur, and permitting a write to a register following a read of that register within a single cycle -- less strict data precedence relationship. 2) microoperations occupying multiple micro-cycles. 3) case_type multiway branch. 4) microsubroutine call.

4. Results and Discussion

Table 3 and Table 4 summarize the weighted average execution time of the results of the four new compaction methods. Table 5 and Table 6 show the space requirement of the results. For convenient comparison, the results of local compaction, trace scheduling and hand compaction ("production" microcode), which is probably optimally compacted or nearly so, are given in Table 3-6. The four new compaction methods are evaluated on the basis of Tables 3-6 using three criteria: time efficiency, space efficiency and complexity.

A. Tree algorithm

Table 5 shows that the space efficiency of the results of the Tree algorithm is better than for the original trace scheduling algorithm; the

percentage of average improvment in Table 5 is 8.7%. The most obvious example is Floating Multiplication, because microoperations can not move up above a branch during top tree compaction and can not move down below a rejoin during bottom tree compaction; the limited movement of microoperations reduces the number of copies generated by the movement of microoperations through basic block boundaries. Lah[6] indicated that, in the worst case, the number of copies generated by trace scheduling is $O(2^n)$, where n equals the number of microoperations which move through a basic boundary, but it will be reduced to $O(n^2)$ with tree compaction.

The complexity of building the DAG, list scheduling and bookkeeping of Tree compaction is lower, as paths are shorter within tree and the movement of microoperations is simpler. The testing and debugging of microprogram are easier, as the toplogical structure of microcode is not changed by Tree compaction.

Table 3 and Table 4 indicate that the weighted average execution time of results of tree compaction is longer than trace scheduling. The worst case is Floating Division on the PUMA, because the limited movement of microoperations in Tree compaction reduces the opportunity for compacting microoperations between different blocks. In fact, this is related to the improvements of space efficiency and complexity of Tree compaction.

B. SRDAG algorithm

Table 5 and Table 6 show that SRDAG produces a slight improvement in the weighted execution time and space requirement in PUMA microcodes. Our experiments prove that the advantage of SRDAG compaction over the other three is due to its more general use of rule R6 and its use of rule R4, which is useful for case-type multiway branches. Linn has conjectured[8] that if a particular microoperation is free at the top of n successive blocks of a case type n way branch, these n duplicated microoperations will be moved up above the case type branch and compacted by R4, so the execution time and space requirement will be 1 cycle and n-1 copies less than trace scheduling does respectively. Table 4 and Table 6 show that both the weighted average execution time and space requirement of DIVF, having a case type four way branch, are less than the result with trace scheduling.

However, since recomputing the data precedence relationship within the whole SRDAG is necessary during each root block compaction, the complexity is much higher than trace scheduling.

C. GDDG algorithm

Table 3 and Table 4 illustrate that the weighted average execution time of the results of GDDG compaction is much longer than for trace scheduling. The worst cases are Floating Division in Table 3 (5.5 cycles longer) and ADDF2 & SUBF2 in Table 4 (1.5 cycles longer). The five transformation rules used in GDDG do not generate any copies, but the effectiveness of global compaction essentially depends to a large extent on the number of movements which compact the microoperations between different blocks and may generate copies. Consequently, these five rules reduce the opportunity of compaction. On the other hand, they improve the space requirement as shown in Table 5. The advantage of GDDG lies in integrating the data precedence relationships with the flow graph and automatically determining the order of compaction of blocks without dynamic frequency information.

D. ITSC algorithm

Table 3 and Table 4 show that ITSC compaction provides almost as short a time for the outer loop microcode as trace scheduling. The results of loop compaction of ITSC using the URCR algorithm were quite comparable with the hand compacted code; this is shown in the Floating Addition and Floating Division of the PUMA and CALL and MULF of the VAX_like machine.

Table 5 shows that the space requirements of the output of ITSC compaction are significantly better than for the original trace scheduling algorithm; the percentage of improvement ranges from 9% to 24%. The most obvious example is Floating Multiplication on the PUMA, because the ITSC algorithm avoids the many extra copies generated by pushing branch microoperations too early in the code. Another reason for the improvement of the space requirements using ITSC is the implementation of the rule dealing with the situation[4] in the microcode where a path forks into two basic blocks which subsequently rejoin; this rule increases the opportunity for compacting microoperations from different blocks and reduces the number of copies.

Because the improved trace scheduling and URCR algorithms are used in ITSC, its complexity is slightly higher than trace scheduling, but the implementation is not very difficult. However, there are two drawbacks to ITSC: 1) The method dealing with case-type multiway branches may significantly increase the space requirements. For example, three extra copies were generated in CALL of VAX_like machine as a microoperation moves down below a case-type four-way branch. Therefore, it seems reasonable to set a threshold in algorithm to avoid generating too many copies. 2) The rule R4 for moving microoperations has not implemented, so that there are more copies than SRDAG produces in MULF and DIVF in Table 6.

5. Conclusions and Suggestions

Table 3-6 indicate that these four new compaction methods, whose results are between those of local compaction and hand compaction, have different tradeoffs between time efficiency and space efficiency as well as between the efficiency of compacted results and complexity of algorithm.

In summary, ITSC method improves both time efficiency and space efficiency by using improved

trace scheduling and URCR algorithms. SRDAG method performs a little better than trace scheduling by extending the trace scheduling technique. The Tree method simplifies the compaction algorithm and the GDDG method does not need dynamic frequency information. Unfortunately, although the space efficiency of the last two methods may improved, their time efficiency is worse than trace scheduling.

Since different machines may have quite different features of their microcode sequences, we should apply an appropriate compaction method to a given machine to obtain the optimum effectiveness. For example, the difference between the results of local compaction and hand compaction in the PUMA is quite big, because the microoperation movement range is rather wide and basic blocks are short. Table 3 and Table 5 show that ITSC compaction performs almost as well as hand compaction both in time and space efficiency. Although its complexity is slightly higher than trace scheduling, ITSC compaction is worth using for the PUMA. In our experiments, the compacting time for ITSC was 20% longer than for trace scheduling, but the compacting time for trace scheduling was 200% longer than for local compaction.

On the other hand, because the difference between local compacted results and hand compacted results for the VAX_like machine is small (as shown in Table 4 and Table 6), the difference among the results of those global compaction methods is also small. In this case, one may prefer a simpler global compaction method.

For the VAX_like machine the relatively low efficiency of global compaction is due to the narrow microoperation movement range, the rather long basic blocks and the case-type multiway branches. Besides, the presence of microsubroutine calls is an important factor. We compact the microsubroutine first, and then regard the microsubroutine call as the situation mentioned in [4,10], where a path forks into two basic blocks which subsequently rejoin. If (using Fisher's terminology[2]) the union of readreg, writereg and condreadreg of microoperation MO does not intersect the readreg or writereg of any microoperation in microsubroutine, MO may be moved up above or down below the call microoperation without any associated bookkeeping. This approach is simpler, but has lower efficiency than the method suggested by Fisher [2]. To increase the time efficiency, we suggest compacting the path which has the highest execution frequency, together with the copy of microsubroutine called by it. The execution time of EDIV in the VAX_like machine with this approach was four cycles shorter, but the space requirement was eight copies more.

We look forward to more extensive tests on microcode sequences with much longer basic blocks, in particular on some compiler produced codes.

Acknowlegdements

We would like to express our appreciation to Professor Ralph Grishman of the Courant Institute for his valuable encouragement and significant help on improving the paper.

References

[1] S. Davidson, D. Landskov, B. D. Shriver and P. W. Mallett, "Some Experiments in Local Microcode Compaction for Horizontal Machine", IEEE Trans. on Computers, vol. C-30, No.7, pp.460-477, July, 1981.

[2] J. A. Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction," IEEE Trans. on Computers, vol.C-30, No.7, pp.478-490, July, 1981.

[3] J. A. Fisher, D. Landskov, and B. D. Shriver, "Microcode Compaction: Looking Backward and Looking Forward," AFIPS National Computer Conference, pp.95-102, 1981.

[4] R. Grishman and Su Bogong, "A Preliminary Evaluation of Trace Scheduling for Global Microcode Compaction,"IEEE Trans.on Computers, vol. C-32, No.12, pp.1191-1194, Dec., 1983.

[5] S. Isoda, Y. Kobayaski, and T. Ishida, "Global Compaction of Horizontal Microprograms Based on Generalized Data Dependency Graph," IEEE Trans. on Computers, vol.C-32, No.10, pp.922-933,1983.

[6] J. Lah and D. E. Atkin, "Tree Compaction of Microprograms,"The Proc. of 16th Annu. Workshop on Microprogramming, pp.23-33, Oct., 1983.

[7] D. Landskov, S. Davidson, B. D. Shriver, and P. W. Mallett, "Local Microcode Compaction Techniques," Computing Survey, vol.12, No.3 pp.261-294, Sept., 1980.

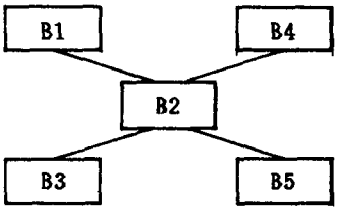
[8] J. Linn, "SRDAG Compaction -- A Generalization of Trace Scheduling to Increase the Use of Global Context Information," The Proc. of 16th Annu. Workshop on Microprogramming, pp.11-22, Oct., 1983.

[9] M. D. Poe, "Heuristic for the Global Optimization of Microprograms," The Proc. of 13th Annu. Workshop on Microprogramming, pp.12-22, 1980

[10] Bogong Su, Shiyuan Ding, and Lan Jin, "An Improvement of Trace Scheduling for Global Microcode Compaction," The Proc. of 17th Annu. Workshop on Microprogramming, pp.78-85, Oct., 1984.

[11] M. Tokoro, T. Takizuka etc., "Optimization of Microprograms," IEEE Trans. on Computers, vol. C-30, No.7, pp.491-504, July, 1981.

[12] G. Wood, "Global Optimization of Microprograms through Modular Control Constructs," The Proc. of 12th Annu. Workshop on Microprogramming, pp.1-6, 1979.



Rule Number	MO can move From	To	Under the Conditions that
R1	B2	B1 and B4	the MO is free at the top of B2
R2	B1 and B4	B2	identical copies of the MO are free at the bottoms of both B1 and B4
R3	B2	B3 and B5	the MO is free at the bottom of B2
R4	B3 and B5	B2	identical copies of the MO are free at the tops of both B3 and B5
R5	B2	B3(or B5)	the MO is free at the bottom of B2 and all registers written by the MO are dead in B5(or B3)
R6	B3(or B5)	B2	the MO is free at the top of B3(or B5) and all registers written by the MO are dead in B5(or B3)

Fig.1 The Rules for Moving Microoperation

Table 1. Characteristics of Sequential Microcodes of PUMA

Code Segment	NO. MO's	NO. Branch	NO. Rejoin	NO. BB's	Max MO's/BB's	Average MO's/BB's	NO.Loops & Body Length
Floating Addition	41	11	7	23	5	1.8	3*3n
Floating Mutiplication	38	8	3	14	21	2.7	/
Floating Division	49	14	7	24	4	2.0	8n
Normalization	32	9	6	20	4	1.6	6n

Table 2. Characteristics of Sequential Microcodes of VAX_like Machine

Code Segment	NO. MO's	NO. Branch	NO. Rejoin	NO. CASE	NO. SUB.Call	NO. BB's	Max. MO's/BB's	Average MO's/BB's	NO.Loops & Body Length
ADDF2 & SUBF2	82	9	10	4	2	28	5	2.9	/
CALL	102	7	4	2	0	19	30	5.4	11n
EDIV	63	8	5	0	2	20	15	3.2	2n+2n
MULF	78	6	9	4	3	23	12	3.4	(5n+5n+8)*
DIVF	56	6	3	1	3	16	10	3.5	2n

Table 3. Weighted Average Execution Time(in cycles) of PUMA

	Sequential	Local Compacted	Trace Scheduled	Tree Compacted	SRDAG Compacted	GDDG Compacted	ITSC Compacted	Hand Compacted
Floating Addition[1]	24.5+3*3n	21+3*2n	15.5+3*2n	15.5+3*2n	15.5+3*2n	16+3*2n	15+3n	13.5+3n
Floating Multiplication	33	22	14	16	14	17	14	14
Floating Division[2]	30+4.6n	22+4n	13+3.4n	18+3.4n	13+3.4n	18.5+3.4n	14.5+3n	13+3n
Normalization[3]	17.5+6n	15+2n	11.5+2n	10+2n	10+2n	10+2n	10+2n	10+2n

Note: (1) n = Average number of shift operation = 3.56
(2) n = Number of quotient bits = 48
(3) n = Average number of shift operation = 0.9

Table 4. Weighted average Execution Time (in cycles)of VAX_like Machine

	Sequential	Local Compacted	Trace Scheduling	Tree Compacted	SRDAG Compacted	GDDG Compacted	ITSC Compacted	Hand Compacted
ADDF2 & SUBF2	19.5	13.5	10.5	11.5	10.5	12	10.5	9.8
CALL (n=5)	73+6n	39.5+6n	35.5+6n	36.5+6n	36.5+6n	37+6n	34.5+5n	35.5+5n
EDIV (n=23)	37+2n	23.5+n	21.5+n	22.5+n	23+n	23+n	21.5+n	21.5+n
MULF (n=11.5)	38.5+3n	15.5+2n	14.5+2n	14.5+2n	14.5+2n	15.5+2n	14.5+2n	13.5+n
DIVF (n=23)	31+2n	16.5+n	13.5+n	13.5+n	12.5+n	14.5+n	13.5+n	12.5+n

Note: n = Average number of loop iterations.

Table 5. Space Requirement of PUMA

	Sequential	Local Compacted	Trace Scheduled	Tree Compacted	SRDAG Compacted	GDDG Compacted	ITSC Compacted	Hand Compacted
Floating Addition	41	33	29	27	27	27	27	26
Floating Multiplication	38	26	25	20	25	22	19	19
Floating Division	49	36	33	33	31	31	27	27
Normalization	32	25	22	19	20	19	20	19

Table 6. Space Requirement of VAX_like Machine

	Sequential	Local Compacted	Trace Scheduled	Tree Compacted	SRDAG Compacted	GDDG Compacted	ITSC Compacted	Hand Compacted
ADDF2 & SUBF2	75	46	44	43	41	45	41	38
CALL	95	56	53	55	55	53	52	48
EDIV	60	36	33	35	35	35	33	33
MULF	69	37	36	37	35	36	37	32
DIVF	48	33	29	29	28	30	29	28