Instruction Scheduling and Register Relabeling Algorithms for Reducing Switching Activity between Instructions

Kun-Yi Wu and Shiann-Rong Kuang
Department of Computer Science Engineering, National Sun Yat-sen University
Kaohsiung, Taiwan, ROC
Email: d963040012@student.nsysu.edu.tw and srkuang@cse.nsysu.edu.tw

Abstract- *In this paper, we present instruction scheduling and register relabeling algorithms for ARM processor to reduce switching activity between instructions. Given the original assembly code and machine code produced by compiler, the proposed algorithm first builds the corresponding DAG (Directed Acyclic Graph) and DDG (Data Dependence Graph) of the assembly code. Then we reorder the sequence of instructions in DAG by our proposed list scheduling algorithm (move_ahead) and re-allocate registers into variables of DDG by tabu search to decrease the switching activity. Experimental results show that our proposed algorithms can achieve 5 % to 25% decrement in switching activity without sacrificing any program performance.*

Keywords: Instruction scheduling, register relabeling, tabu search, switching activity.

1. Introduction

In the past years the development of VLSI implementation emphasizes on performance and miniaturization. Recently with the significant growth in cost of packaging, cooling, digital noise immunity, and device reliability, the demands of multimedia and digital signal processing (DSP) applications have concentrated attention on the low power design under tardy progress of power support devices. Therefore, it is crucial to develop a low power system with constraints of performance and area.

It can be undertaken at many abstract levels of the design process for minimizing power and energy consumptions. However, the low energy implementation achieved by software could have less overhead costs than by hardware. Compiler is one of software based techniques for program code optimization. For architecture with the various characteristics, compiler will be flexible to execute optimization techniques to bring these features into full play. This is why we want to explore low energy technique in compiler.

Instruction scheduling and register allocation are common static optimal techniques in the code generator phase of compiler to achieve high performance and reduce spill cost and power consumption. Instruction scheduling reorders the sequence of instructions to arrive the target of low power. The work in [1, 2] evaluates several instruction scheduling algorithms while considering the energy dissipation with performance constraints. In [3] Sinevriotis et al. present a list-scheduling algorithm for minimizing the total inter-instruction effect cost. Another approach in [4] adopts two ways, Gary code addressing and Cold scheduling, for decreasing switching activity. Moreover, horizontal scheduling and vertical scheduling algorithms were proposed in [5] for optimizing transition activities in instruction bus of VLIW architectures. Vertical scheduling recombines these microinstructions of original instructions into new instructions with performance constraint. However, this approach doesn't reschedule the first instruction and likes Top_Down approach in [1][2] when it is used for ARM processors. On the other hand, the work of traditional register allocation is assigning variables of IR (Intermediate Representation) generated in most modem compiler into the limit number of registers in the specific architecture. There are many researches [6]–[8] using graph coloring approach to solve spilling problem. HEA (hybrid evolutionary algorithm) approach [9] efficiently solves this problem by combining specialized heuristic with EA.

In this paper, we propose an efficient list-based scheduling algorithm and apply the tabu search [10] to solve the low-power instruction scheduling and register relabeling problems, respectively. Given the assembly code and machine code generated by compiler, the proposed approach will construct DAG and then perform move_ahead scheduling in

DAG to reduce the switching activity between instructions. Since the different instruction orders will make different DDG, thus DDG will be generated based on the sequence of instructions after instruction scheduling. Finally, we utilize tabu search to re-label (rename) the registers in DDG to further decrease the switching activity between instructions. Experimental results show that our proposed approaches can efficiently and quickly reduce the switching activity without sacrificing any program performance.

The remainder of this paper is organized as follows. Section 2 gives the problem description and definition. Section 3 introduces the proposed instruction scheduling method in detail. We will discuss the proposed register relabeling method in Section 4. Section 5 shows the experimental results. Finally, a concluding remark is given in Section 6.

2. Problem Description and Definition

Instruction scheduling and register relabeling are the main methods adopted in this paper to reduce switching activity. Before constructing DAG for scheduling and DDG for relabeling, we must distinguish the source and destination registers of all instructions beforehand. The DAG and DDG are explained as follows.

Table 1. The assembly code and machine code of DIFF example

No	Assembly code	Machine code
1	ldr r5 , [sp, #24]	18509de5
2	cmp r5 , r0	000055e1
3	mov r7 , r0	0070a0e1
4	mov r8 , r1	0180a0e1
5	mov r6 , r2	0260a0e1
6	mov r4 , r3	0340a0e1
7	ldr lr , [sp, #28]	1ce09de5
8	ldr ip , [sp, #32]	20c09de5
9	bge L6	140000aa
10	mul r2 , lr, r8	9e0802e0
11	mul r1 , r5, r2	950201e0
12	mul r0 , ip, r6	9c0600e0
13	mul r2 , r4, r1	940102e0
14	mul r1 , r4, r0	940001e0
15	add r3 , r5, r4	043085e0
16	rsb r2 , r2, lr	0e2062e0
17	mla ip , r4, lr, ip	94ce2ce0
18	mov r5 , r3	0350a0e1
19	cmp r3 , r7	070053e1
20	rsb lr , r1, r2	02e061e0
21	blt L11	080000ba
22	mov r0 , ip	0c00a0e1

2.1. Directed Acyclic Graph (DAG)

Each node of DAG denotes one instruction. There is an edge from node I to node J in DAG if instruction I writes a new value to its destination register R and instruction J reads or writes the same register R. In this case, we call node J is a successor of node I and node I is one predecessor of node J. In addition, the live range of a destination register R is the interval between the instruction I that writes a new value to it and the last successor of instruction I. For instance, Figure1 is the DAG of DIFF example shown in Table 1. There are three basic blocks BB0, BB1, and BB2 in the DAG. Node 10 is a predecessor of node 11 and node 11 is a successor of node 10.

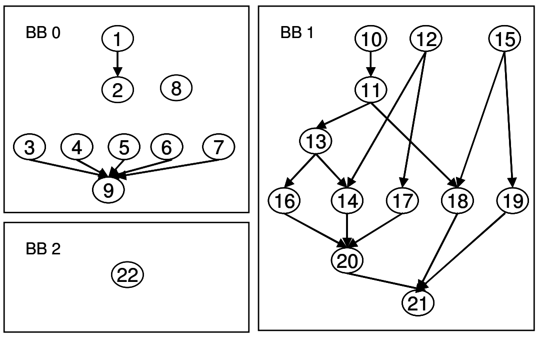


Figure 1. The DAG of DIFF example

2.2. Data Dependence Graph (DDG)

Each node of DDG is a destination register of the instruction in DAG. These nodes must be merged by data flow analysis cross basic blocks and deleted by the limits of specific architecture to simplify the DDG. Then, the live ranges of these remnant nodes in DDG must be found. An edge between two nodes is added if their live ranges are overlapping each others.

Not all destination registers could be a node in DDG. For example, some special purpose registers and initial reserved registers that ARM processor will pre-fetch in the beginning of function procedure call will limit the number of available registers. Figure 2 shows the DDG of DIFF example.

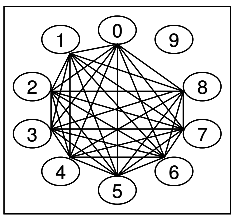


Figure 2. The DDG of DIFF

3. Instruction Scheduling- move_ahead

To generate assembly code quickly, list scheduling is often adopted to schedule instruction in compiler. The most important key point of developing a list scheduling is how to efficiently select the next instruction to schedule and break tie in ready list so that local optimization can be avoided.

Top_Down [1, 2] is a common and simple list scheduling method. It decides the next instruction to schedule by greedy method. However, we discover that if we can move the current selected instruction which should be scheduled at position K into previous position L which had already scheduled one instruction, and then move the instructions which have be scheduled at position L ~ K−1 to their next position, the efficiency of Top_Down scheduling can be improved. We call the new list scheduling method as move_ahead.

```
move_ahead()
{
  Initialize ready list, done list, solution, SW, count,
      basic block index n=0;
  While ready list is not empty {
    For each instruction I in ready list of $BB_n$ {
        Calculate Object(I, previous position, count)
          with all possible positions until position
          avoid the dependence constraints;
        Select instruction K and location L with
          smallest switching activity;
    }
    Add K into done list and update solution, ready
        list and SW. If all instructions in basic
        block $BB_n$ have been scheduled; n++;
  }
  Move instruction I into previous position if this
    movement will make better solution in the
    backward direction. This step is called *rebound.*
}
Object(I, pp, count)
{
  SW += $\alpha$(done[pp-1], I) + $\alpha$(I, done[pp]) −
    $\alpha$(done[pp-1], done[pp]) − $\alpha$(done[count-1], I);
  return SW;
}
```

Figure 3. The algorithm of move_ahead

Besides, after all instructions have been scheduled by move_ahead method, we will check and move instruction into previous position again in the backward direction if this movement will reduce switching activity. This step is called rebound. When a tie occurs, tie breaker will consider the much later instructions will be scheduled in the future until this tie has been broken. Scheduling the first instruction of each basic block is regarded as a tie happened. The main steps of move_ahead are listed as follows, where $\alpha(X, Y)$ denotes the switching activity between instruction X and Y. Moreover, count is the original scheduled location, and SW denotes the switching activity.

Here we illustrate move_ahead with DIFF example shown in Table 1. Table 2 is the result of performing move_ahead step by step. The column "I" in Table 2 mean the selected instruction to schedule in current step. The column "L" is location moved by move_ahead. Instruction 17 would be moved to order 11 instead of order 19. Then we move instruction 10 to order 10 by rebound step. The final scheduled order is {8, 7, 1, 2, 4, 6, 5, 3, 9, 10, 12, 17, 11, 13, 14, 15, 18, 19, 16, 20, 21, 22}.

Table 2. A trace of performing move_ahead step by step before rebound step

order	I	L	ready list
-	-	-	1,3,4,5,6,7,8
1	8	-	1,3,4,5,6,7
2	7	-	1,3,4,5,6
3	1	-	2,3,4,5,6
4	2	-	3,4,5,6
5	4	-	3,5,6
6	6	-	3,5
7	5	-	3
8	3	-	9
9	9	-	10,12,15
10	12	-	10,15,17
11	10	-	11,15,17
12	11	-	13,15,17
13	13	-	14,15,16,17
14	14	-	15,16,17
15	15	-	16,17,18,19
16	18	-	16,17,19
17	19	-	16,17
18	16	-	17
19	17	11	20
20	20	-	21
21	21	-	22
22	22	-	-

4. Register Relabeling - Tabu Search

Tabu search [9] is one heuristic search method to escape the local optimal solution. Tabu search has been used to solve many problems efficiently, e.g. integer programming problem, scheduling, routing, traveling salesman [11-13]. In this paper, we apply tabu search to solve the register relabeling problem. For the DIFF example, it can further reduce switching activity from 18.99% to

25.69%.

Given a DDG as shown in Figure 2, the flow chart of tabu search to solve the register relabeling problem is shown in Figure 4. The essence of tabu search is using memory structure (e.g. tabu list and intensification etc.) to avoid generating solutions that we had searched recently. The main steps are described in detail as follows.

- **Initial solution**
  Each node of DDG, which represents a destination register filed of one instruction in DAG, must be assigned one register, and a feasible register allocation for each node of DDG is a solution of register relabeling. The initial solution is the register allocation for each node of DDG generated by compiler.

- **Move**
  All the steps inside the circle of Figure 4 are called one move. The move is repeated until the stop rule is satisfied.

- **Neighbor**
  Neighbor is known as a solution in tabu search. All neighbors are generated according to the optimal solution with the least switch activity (SA) in last move by changing the allocated register for each node of DDG.

- **Tabu list**
  Tabu list is a short memory structure that records these nodes used for generating the optimal solutions of past moves. The number of nodes recorded in tabu list is called tabu length. If the node is found in tabu list while generating new neighbors, we will not generate neighbors according to it. That is, the solutions in tabu list are locked. Therefore tabu list will help us to escape from the local optimization.

- **Aspiration criterion**
  Sometimes it is possible that these locked solutions will generate much better solution than the current best solution in all past moves. Aspiration criterion will sometimes allow the locked solution to generate new neighbors.

- **Intensification**
  Intensification will keep all local optimal solutions of the past moves and make sure never repeat the same solution in the later moves.

- **Stop rule**
  If the number of moves in tabu search reaches an appointed number, then we stop the tabu search.

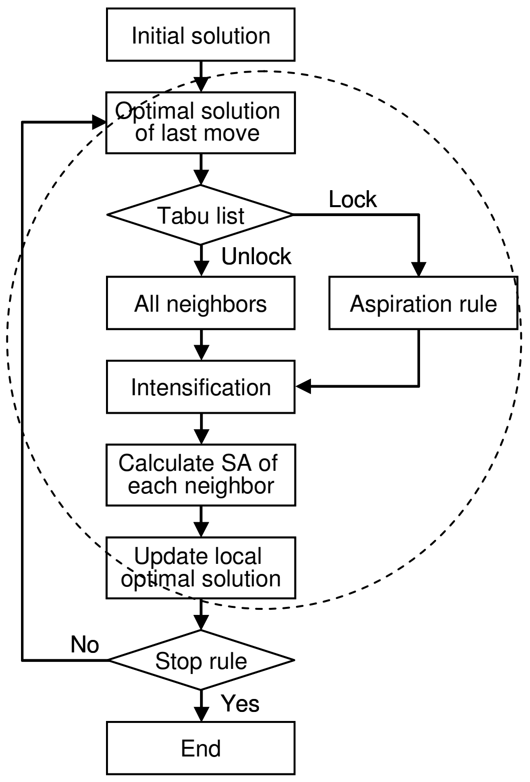
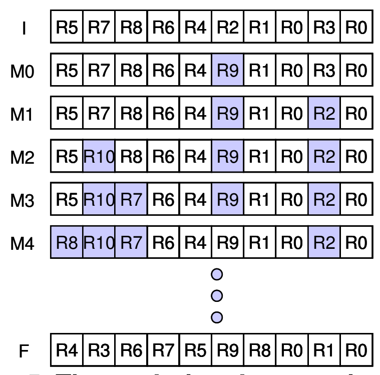


Figure 4. The flow chart of tabu search

We illustrate tabu search in Figure 5 with DIFF example of Figure 2. In Figure 5, "I" is the initial solution. MK is the optimal solution of the K_{th} move. The optimal solution of move 0 is changing R2 to R9 in node5. We color node 5 gray. That means we lock node 5 in tabu list. When the number of nodes in tabu list is over tabu length 4, we unlock node 5 with white color in move 4. We repeat above steps until the stop rule is satisfied. "F" denotes the final solution generated by tabu search.



I	R5	R7	R8	R6	R4	R2	R1	R0	R3	R0
M0	R5	R7	R8	R6	R4	R9	R1	R0	R3	R0
M1	R5	R7	R8	R6	R4	R9	R1	R0	R2	R0
M2	R5	R10	R8	R6	R4	R9	R1	R0	R2	R0
M3	R5	R10	R7	R6	R4	R9	R1	R0	R2	R0
M4	R8	R10	R7	R6	R4	R9	R1	R0	R2	R0
						○ ○ ○				
F	R4	R3	R6	R7	R5	R9	R8	R0	R1	R0

Figure 5. The evolutional generations of performing tabu search with tabu length=4

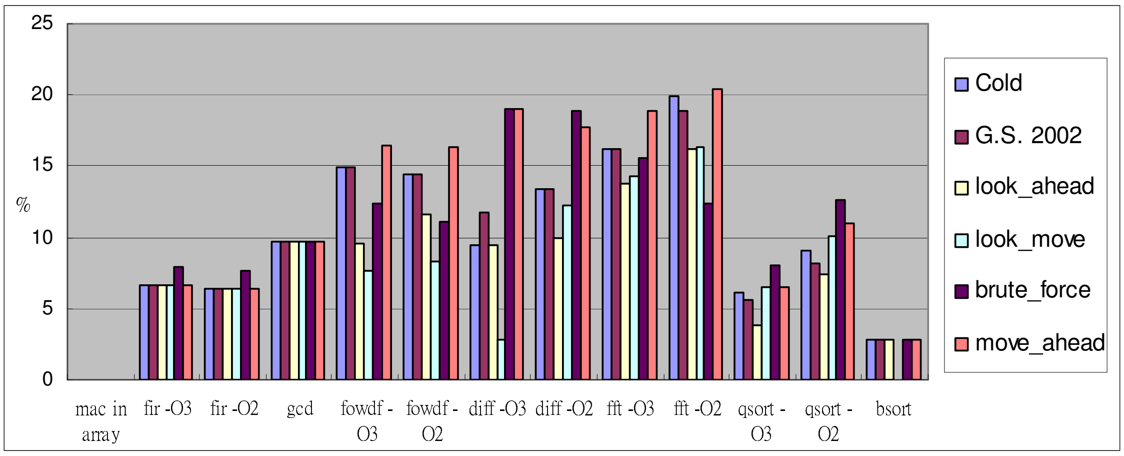


Figure 6. The saving of switching activity for different approaches

5. Experimental Results

In this section we will demonstrate the efficiency of our proposed approaches. Experiments are performed on the experimental environment using AMD Opteron™ 248 CPU, 1.96GB Memory, Gentoo with kernel 2.6.14-gentoo-r5 as OS. -O2 and -O3 in the following figures represent the different optimal level for generating ARM assembly and machine codes of real benchmark by compiler.

First we implemented five different approaches for comparing with the proposed move_ahead approach. These approaches include Cold scheduling [4], G.S.2002 [3], look_ahead [1], look_move, and brute_force. look_move is the combination of look_ahead and move_ahead. brute_force will generate all possible schedules and pick the best one until over 24 hours. Figure 8 shows the cost time of executing ten thousand times for different approaches in this paper. Except for brute_force, all other approaches can obtain the solutions within 0.01 seconds for all benchmarks in Figure 8.

The results in Figure 6 show that move_ahead is much better than other list scheduling approaches especially for complex cases. Figure 7 shows the further saving on switching activity by using register relabeling after instruction schedule. The result demonstrates that tabu search is very efficient in register relabeling for reducing switching activity. It only spends no more than 0.5 second to obtain the result for the benchmark with 200 instructions. “The best in 24 hours” in Figure 7 means the result is obtained by finding all possible allocations until arriving 24 hours and picking the best one as the solution.

6. Conclusion

An efficient approach explores instructions scheduling by our novel move_ahead and register relabeling by famous tabu search method has been proposed in this paper. The most different view compared to other papers is we propose a specific list schedule can change schedule order that already done while scheduling next one. In the experimental results move_ahead is much better than other list schedule, and even nearly closes the optimal solution. It also shows that an excellent performance and high quality solution can be obtained by tabu search for register relabeling. Consequently we prove our proposed approach can efficiently reduce switching activity in ARM.

7. Acknowledgment

This work was supported in part by the National Science Council, Taiwan, under Grant NSC 96-2220-E-110-007.

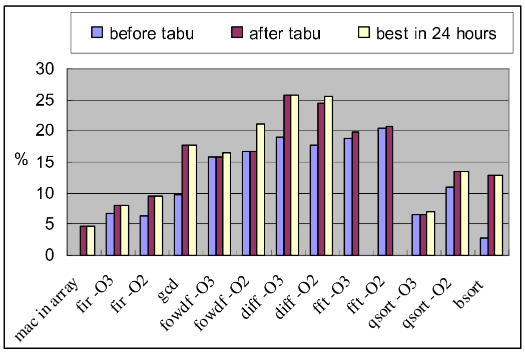


Figure 7. The saving of switching activity after executing tabu search

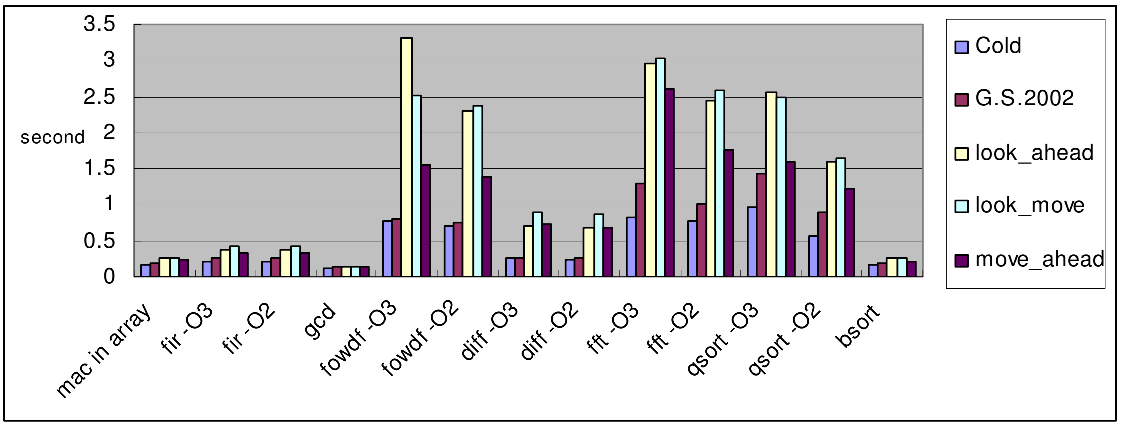


Figure 8. The cost time of executing 10000 times for different approaches

References

[1] A. Parikh, M. Kandemir, N. Vijaykrishnan, and M.J. Irwin, "Instruction Scheduling based on Energy and Performance Constraints," *IEEE Computers Society Annual Workshop on VLSI*, pp. 37-42, April 2000.

[2] A. Parikh, M. Kandemir, N. Vijaykrishnan, and M.J. Irwin, "VLIW scheduling for energy and performance," *IEEE Computer Society Workshop on VLSI*, pp. 111-117, April 2001.

[3] G. Sinevriotis and T. Stouraitis, "A novel list-scheduling algorithm for the low-energy program execution," *IEEE International Symposium on Circuits and Systems*, Vol. 4, pp. IV-97 - IV-100, May 2002.

[4] C.-L. Su, C.-Y, Tsui, and A.M. Despain, "Low power architecture design and compilation techniques for high-performance processors," *IEEE COMPCON*, pp. 489-498, March 1994.

[5] C.-G Lee, J.-K Lee, and T.-T Hwang, "Compiler Optimization on VLIW Instruction Scheduling for Low Power," *ACM Transactions on Design Automation of Electronic Systems* (TODAES), Vol. 8, No. 2, pp. 252-268, 2003.

[6] G. Chaitin, "Register allocation and spilling via graph coloring," *ACM SIGPLAN Symposium on Compiler Construction*, pp. 98-105, 1982.

[7] P. Briggs, K.D. Cooper, K. Kennedy, and L. Torczon, "Coloring heuristics for register allocation," *ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 275-284, June 1989

[8] F.M.Q. Pereira and J. Palsberg, "Register allocation via coloring of chordal graphs," *Asian Symposium on Programming Languages and Systems*, pp. 315-329, November 2005.

[9] H.P. Topcuoglu, B. Demiroz, and M. Kandemir, "Solving the Register Allocation Problem for Embedded Systems Using a Hybrid Evolutionary Algorithm," *IEEE Transactions on Evolutionary Computation*, Vol. 11, No. 5, pp. 620-634, 2007.

[10] F. Glover and M. Laguna, Tabu Search, Kluwer Academic Publishers, 1997.

[11] S. Porto and C. Ribeiro, "A Tabu search Approach to Task Scheduling on Heterogeneous Processors under Precedence Constraints," *International Journal of High Speed Computing*, Vol. 7, No. 1, pp. 45-71, 1995.

[12] M.K. Dhodhi and I. Ahmad, "Task tree scheduling onto linear arrays using tabu search," *IEE Proceedings - Computers and Digital Techniques*, Vol. 144, No. 5 , pp. 317-323, 1997.

[13] C. Dzongang, P. Galinier, and S. Pierre, "A tabu search heuristic for the routing and wavelength assignment problem in optical networks," *IEEE Communications Letters*, Vol. 9, No. 5, pp. 426-428, 2005.