Loop Scheduling for Real-Time DSPs with Minimum Switching Activities on Multiple-functional-unit Architectures *

Zili Shao[1], Qingfeng Zhuge[1], Meilin Liu[1], Edwin H.-M. Sha[1], and Bin Xiao[2]

[1] University of Texas at Dallas
Richardson, Texas 75083, USA
{zxs015000, qfzhuge, mxl024100, edsha}@utdallas.edu
[2] Hong Kong Polytechnic University
Hung Hom, Kowloon, Hong Kong
csbxiao@comp.polyu.edu.hk

Abstract. This paper studies the scheduling problem that minimizes both schedule length and switching activities for applications with loops on multiple-functional-unit architectures. We formally prove that to find a schedule that has the minimal switching activities among all minimum-latency schedules with or without resource constraints is NP-complete. An algorithm, SAMLS (Switching-Activity Minimization Loop Scheduling), is proposed to minimize both schedule length and switching activities. In SAMLS, the best schedule is selected from the ones generated from a given initial schedule by repeatedly rescheduling the nodes with schedule length and switching activities minimization based on rotation scheduling and bipartite matching. The experimental results show our algorithm can greatly reduce both schedule length and switching activities compared with the previous work.

1 Introduction

In many portable systems, such as wireless communication and image processing systems, the DSP processor core consumes a significant amount of power and time in highly computation-intensive applications. In such applications, loops are the most critical sections. An efficient loop scheduling scheme can help reduce the power consumption while still satisfying the timing constraint. Switching activities play a key role in the total power consumption [5], therefore, various techniques have been proposed to reduce power consumption by reducing switching activities [2, 3].

This paper focuses on reducing both switching activities and schedule length of an application on multiple-functional-unit architectures such as VLIW (Very-Long Instruction Word) processors. In a multiple-functional-unit architecture,

* This work is partially supported by TI University Program, NSF EIA-0103709, Texas ARP 009741-0028-2001, NSF CCR-0309461, USA, and HK POLYU A-PF86 and COMP 4-Z077, HK.

several instructions can be executed in parallel. The power consumption in a clock cycle, P_{cycle}, can be computed by:

$$P_{cycle} = P_{base} + \sum_{Inst_i} \{P_{Inst_i} + SP(i,j)\} \quad (1)$$

where P_{base} is the base power needed to support instruction execution, P_{Inst_i} is the basic power to execute an instruction I_i on a functional unit, and $SP(i,j)$ is the switching power caused by switching activities between $Inst_i$ (current instruction) and $Inst_j$ (last instruction) executed on the same functional unit (FU). Let S be a schedule for an application and L the schedule length of S. Then the energy E_S for Schedule S can be computed by

$$E_S = \sum_{k=1}^{L} P_{cycle}^{(k)} = L * P_{base} + \sum_{k=1}^{L} \sum_{Inst_i^{(k)}} P_{Inst_i^{(k)}} + \sum_{k=1}^{L} \sum_{Inst_i^{(k)}} SP^{(k)}(i,j) \quad (2)$$

$\sum\sum P$ is the summation of basic power consumptions for all instructions of an application. It does not change with different schedules. L and $\sum\sum SP(i,j)$ will change with different schedules though. Therefore, in order to minimize the energy consumption of an application, schedule length and switching activity both need to be considered in scheduling.

Low power scheduling to reduce switching activities has been extensively studied in high level synthesis (HLS) and compiler optimization. In HLS, a lot of approaches have been proposed to minimize switching activities based on single-FU architecture[4] or a fixed schedule [3]. In these techniques, schedule length is assumed to be fixed, so optimizing schedule length is not considered. In compiler optimization, various instruction-level scheduling techniques have been proposed to reduce power consumption[6]. Most of the techniques are based on DAG (Directed Acyclic Graph) Scheduling, in which an application is modeled as DAG and only the DAG parts of loops are considered. The loop pipelining techniques [7] can not be applied to optimize schedule length when loops are represented as DAGs.

Low power loop compilation optimization techniques have been proposed in [9, 10]. However, with the focus on reducing power variations of applications, they can not be directly applied to optimize the energy consumption. In HLS, based on operand sharing approach, a loop pipelining methodology to reduce both latency and power is first proposed in [2]. Using similar approach, a loop pipelining technique is proposed to first minimize power and then maximize throughput in [1]. These techniques are based on operand sharing and can not be directly used on multiple-functional-unit architectures. In [8], we show the loop scheduling problem with minimum latency and minimum switching activities is NP-complete without detailed proof and propose an algorithm to reduce both schedule length and switching activities base on greedy strategy. However, the greedy algorithm did not give very good results. Therefore, in this paper, we propose a better algorithm for multiple-functional-unit architectures to reduce both schedule length and switching activities for an application with loops. Our

scheme is constructed based on a general model and can be applied in either HLS or compiler optimization.

In the paper, we first analyze the complexity of the low power loop scheduling problem and formally prove that the loop scheduling problem with minimum latency and minimum switching activities is NP-complete with or without resource constraints. We then propose an algorithm, SAMLS (Switching-Activity Minimization Loop Scheduling), to minimize both schedule length and switching activities for loops. In SAMLS, the best schedule is selected from the ones generated from a given initial schedule by repeatedly rescheduling the nodes with schedule length and switching activities minimization based on rotation scheduling and bipartite matching. Finally, we conduct experiments on a VLIW simulator similar to TI C6000 DSP. The experimental results show significant reduction in switching activities and schedule length compared with the previous work.

In the next section, we introduce necessary background. Section 3 presents NP-completeness proofs. The algorithm is discussed in Section 4. Experimental results and concluding remarks are provided in Section 5 and 6, respectively.

2 Basic Concepts and Models

Data Flow Graph (DFG) is used to model loops and is defined as follows. A *Data Flow Graph (DFG)* $G = \langle V, E, OP, d, t \rangle$ is a node-weighted and edge-weighted directed graph, where V is the set of operation nodes, $E \subseteq V * V$ is the edge set that defines the precedence relations for all nodes in V, $OP(u)$ is a binary string associated with each node $u \in V$, $d(e)$ represents the number of delays for an edge e, and $t(u)$ is the computation time of node u. Nodes in V can be various operations, such as addition, subtraction, multiplication, logic operation, etc. $OP(u)$ is a binary string that denotes the state of signal associated with node u. It may represent different values in different optimization environment. For example, $OP(u)$ can be used to represent the operand of node u in optimizing switching activities in functional units [4], or it can be used to represent the binary code of node u in optimizing switching activities in instruction buses [6],

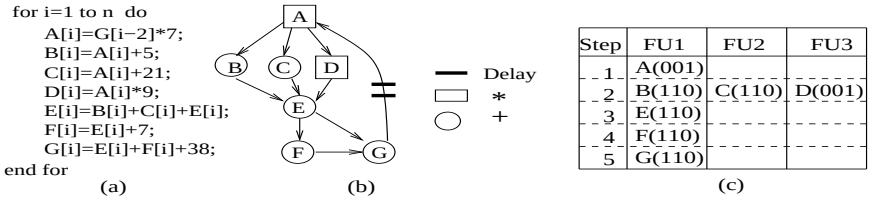


Fig. 1. (a) A loop. (b) The DFG. (c) The static schedule.

In our case, a DFG can contain cycles. The intra-iteration precedence relation is represented by the edge without delay and the inter-iteration precedence relation is represented by the edge with delays. The *cycle period* of a DFG corresponds to the minimum schedule length of one iteration of the loop when there are no resource constraints. An example is shown in Figure 1. The DFG in Figure 1(b) models the loop in Figure 1(a). In this example, there are two kinds of

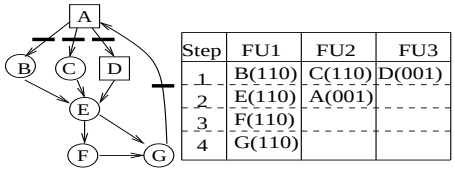
operations: multiplication and addition. They are denoted by the rectangle and circle as shown in Figure 1(b).

A *static* schedule of a cyclic DFG is a repeated pattern of an execution of the corresponding loop. In our work, a schedule implies both control step assignment, and functional unit allocation. A static schedule must obey the precedence relations of the *directed acyclic graph (DAG)* portion of the respective DFG. The DAG is obtained by removing all edges with delays in the DFG. Figure 1(c) shows the static schedule when there are three FUs and the computation time of each node is assumed to be one. The schedule is obtained by list scheduling. In the schedule, the binary string in the parenthesis beside each node denotes the states of signal associated with nodes. To make it simple, we assume that all multiplication operation nodes are associated with the same state of signal, 001, and all addition operation nodes are with the same state of signal, 110. These assumptions are only for demonstration purpose. In practice, nodes with the same operation may have different states of signal. We use $[i, j]$ to denote the location of a node in a schedule, where i is the row (control step) and j is the column (FU). For example, location $[2, 1]$ in the schedule refers to node B scheduled at control step 2 and assigned to FU_1 in Figure 1(c).

Retiming [11] can be used to optimize the cycle period of a DFG by evenly distributing the delays in it. Given a DFG $G = \langle V, E, OP, d, t\rangle$, retiming r of G is a function from V to integers. For a node $u \in V$, the value of $r(u)$ is the number of delays drawn from each of its incoming edges of node u and pushed to all of its outgoing edges. Let $G_r = \langle V, E, OP, d_r, t\rangle$ denote the retimed graph of G with retiming r, then $d_r(e) = d(e) + r(u) - r(v)$ for every edge $e(u \rightarrow v) \in V$ in G_r.

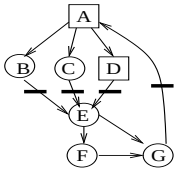
Rotation Scheduling [7] is a scheduling technique used to optimize a loop schedule with resource constraints. It transforms a schedule to a more compact one iteratively. In most cases, the minimal schedule length can be obtained in polynomial time by rotation scheduling. In each step of rotation, nodes in the first row of the schedule are rotated down. By doing so, the nodes in the first row are rescheduled to the earliest possible available locations. From retiming point of view, each node gets retimed once by drawing one delay from each of incoming edges of the node and adding one delay to each of its outgoing edges in the DFG. The new location of the node in the schedule must also obey the precedence relation in the new retimed graph. The retimed graphs and schedules after the first and second rotation are shown in Figure 2(a) and Figure 2(b) respectively, which is based on the original schedule in Figure 1(c). The minimal schedule length is obtained by the schedule in Figure 2(b).

Switching activity is used as the indicator of the power consumption in our work. The switching activity of node u bound to functional unit FU_i, Switch_Node(u,FU_i), is defined as the hamming distance between LAST_OP(FU_i) and $OP(u)$, where $OP(u)$ is the state of signal of u and LAST_OP(FU_i) is the state of signal of the node executed on FU_i before u. The switching activity of a static schedule for a DFG is defined as the summation of the switching activities of all nodes bound to FUs. Since the static schedule is repeatedly ex-



Step	FU1	FU2	FU3
1	B(110)	C(110)	D(001)
2	E(110)	A(001)	
3	F(110)		
4	G(110)		

(a)



Step	FU1	FU2	FU3
1	E(110)	A(001)	
2	F(110)	B(110)	C(110)
3	G(110)	D(001)	

(b)

Fig. 2. The retimed graphs and schedules after (a)the first rotation and (b)the second rotation.

ecuted for the loop, the initial value of LAST_OP(FU_i) is set as $OP(u)$ where u is the last node executed on FU_i in the previous iteration. For example, for the static schedule shown in Figure 1(c), the initial value of LAST_OP(FU_1) is 110($OP(G)$ of G). The switching activity is 6 for the static schedule shown in Figure 1(c), where the switching activities are 3+3+0+0+0=6 on FU_1 and 0 on FU_3 and FU_4. The switching activity remains 6 for both schedules in Figure 2(a) and Figure 2(b). In order to make it simple, we assume the state on a FU will not change with an empty slot. It may not be true for some optimization problems. For example, when the problem is to optimize switching activities on an instruction bus, an empty slot will represent a "NOP" instruction and will cause switching activities. Our algorithm is general and can be easily extended to deal all cases.

The problem we intend to solve is defined as follows. Given a cyclic DFG $G = \langle V, E, OP, d, t\rangle$ that models a loop and a set of FUs, find a static schedule S of G such that S has the minimum switching activities in all possible minimum-latency schedules. We call the problem as the min-latency-switching-activity scheduling problem.

3 NP Completeness

In this section, we prove the min-latency-switching-activity scheduling problem is NP-complete. We categorize the problem into three cases and give proofs as follows.

When the number of resources is greater than one but not infinite, it is known that the minimum latency loop scheduling is NP-complete[12]. So the min-latency-switching-activity scheduling problem is also NP-complete.

Theorem 1. *Let U be the number of resources, where $U > 1$ and $U < \infty$, min-latency-switching-activity scheduling problem is NP-complete.*

Proof. When $U > 1$ and $U < \infty$, the minimum latency loop scheduling problem is NP-complete[12]. Given an instance of the minimum latency loop scheduling problem, we can assigning all nodes with the same $OP(u)$ to get an instance of our problem. Thus, we transform the minimum latency loop scheduling problem to our problem in polynomial time.

When the number of resources equals to one, it is known that the minimum latency loop scheduling is trivially polynomial-time solvable. However, this is not the case when switching activities are considered as the second constraint.

Theorem 2. *Let U be the number of resources, when $U = 1$, min-latency-switching-activity scheduling problem is NP-complete.*

In order to prove Theorem 2, we first define the decision problem (DP1) of min-latency-switching-activity scheduling problem when $U = 1$.

DP1: Given a cyclic DFG $G = \langle V, E, OP, d, t\rangle$, one FU and two constants D and K, does there exist a static schedule that has the schedule length at most D and has the switching activity at most K?

In our proof, we will transform the L_1 Geometric Traveling Salesman Problem (GTSP) to our problem. GTSP is defined as follows[13].

The L_1 geometric traveling salesman problem (GTSP): Given a set S of integer coordinate points in the plane and a constant L, does there exist a circuit passing through all the points of S which, with edge length measured by L_1, has total length less than or equal to L?

Proof. It is obvious DP1 belongs to NP. Assume $S = \{[x_1, y_1], [x_2, y_2], \cdots, [x_n, y_n]\}$ is an instance of GTSP. Construct DFG $G = \langle V, E, OP, d, t\rangle$ as follows. $V = \langle v_1, v_2, \cdots, v_n\rangle$ where v_i corresponds to a point $[x_i, y_i]$ in S. $E = \emptyset$. Assume that $X = \max(x_i)$ and $Y = \max(y_i)$ for $1 \le i \le n$, then $OP(v_i) = (X - x_i)0's \bullet x_i1's \bullet (Y - y_i)0's \bullet y_i1's$ for each $v_i \in V$ $(1 \le i \le n)$, where "$\bullet$" denotes concatenation. For example, if $X = Y = 3$, $x_1 = 2$, and $y_1 = 1$, then $OP(v_1) =$ 011 001. Set $t(u) = 1$ for each node $u \in V$. Set $D = n$ and $K = L$. Since GTSP is NP-complete and the reduction can be done in polynomial time, DP1 is NP-Complete.

When there is no resource constraints, the minimum latency loop scheduling problem is polynomial-time solvable. Retiming[11] can be used to find an optimal solution. However, when switching activities are considered, the problem becomes NP-complete.

Theorem 3. *Let U be the number of resources, when $U = \infty$, min-latency-switching-activity scheduling problem is NP-complete.*

The decision problem (DP2) of min-latency-switching-activity scheduling problem when $U = \infty$ is similar to DP1 except that there is one FU in DP1 while no resource constraint in DP2. The proof of Theorem 2 is as follows.

Proof. It is obvious DP2 belongs to NP. Assume $S = \{[x_1, y_1], [x_2, y_2], \cdots, [x_n, y_n]\}$ is an instance of GTSP. Construct DFG $G = \langle V, E, OP, d, t\rangle$ as follows. $V = V^{(1)} \cup V^{(2)}$, where $V^{(1)} = \langle v_1^{(1)}, v_2^{(1)}, \cdots, v_n^{(1)}\rangle$ and $V^{(2)} = \langle v_1^{(2)}, v_2^{(2)}, \cdots, v_n^{(2)}\rangle$. The nodes in $V^{(1)}$ correspond to the points in S. Assume that $X = \max(x_i)$ and $Y = \max(y_i)$ for $1 \le i \le n$, then $OP(v_i^{(1)}) = (X + Y + 2)1's \bullet (X - x_i)0's \bullet x_i1's \bullet (Y - y_i)0's \bullet y_i1's$ for each node $v_i^{(1)} \in V^{(1)}$ $(1 \le i \le n)$. For example, if $X = Y = 3$, $x_1 = 2$, and $y_1 = 1$, then $OP(v_1^{(1)}) =$ 11111111 011 001. The nodes in $V^{(2)}$ construct a cycle. Set $OP(v_i^{(2)})$ all 0's for $1 \le i \le n$. Add edge $e(v_i^{(2)} \to v_{i+1}^{(2)})$ to E and set $d(e(v_i^{(2)} \to v_{i+1}^{(2)})) = 0$ for $1 \le i \le (n-1)$. Add edge $e(v_n^{(2)} \to v_1^{(2)})$ to E and set $d(e(v_n^{(2)} \to v_1^{(2)})) = 1$. Set $t(u) = 1$ for each $u \in V$. Set $D = n$ and $K = L$. Set the initial state of signal of each FU to all 0's.

With the construction of $V^{(2)}$, the assignment of nodes in $V^{(2)}$ does not introduce switching activities and the minimum schedule length equals to n. The construction of $V^{(1)}$ makes all nodes in $V^{(1)}$ be assigned to the same FU for minimizing switching activities. Since the reduction can be done in polynomial time, DP2 is NP-complete.

4 The SAMLS Algorithm

In this section, we propose an algorithm, SAMLS (Switching-Activity Minimization Loop Scheduling), to reduce both schedule length and switching activities for applications with loops. The basic idea is to obtain a better schedule by repeatedly rescheduling the nodes with schedule length and switching activities minimization based on Rotation Scheduling and bipartite matching. The SAMLS algorithm is shown in Figure 3.

In Algorithm SAMLS, we first generate N schedules based on a given initial schedule and then select the one with the minimum switching activities among all minimum-latency schedules, where N is an input integer to determine the rotation times. These N schedules are obtained by repeatedly rescheduling the nodes in the first row to new locations based on the rotation scheduling with schedule length and switching activities minimization. Two functions, BipartiteMatching_NodesSchedule() and RowByRow_BipartiteMatching(), are used to generate a new schedule.

BipartiteMatching_NodesSchedule() is used to reschedule the nodes in the first row to new locations to minimize schedule length and switching activities. Basically, in the algorithm, we construct a weighted bipartite graph between the nodes and the empty locations and reschedule the nodes based on the obtained minimum cost matching. The algorithm is shown in Figure 4.

RowByRow_BipartiteMatching() is then used to further minimize the switching activities of a schedule by performing a row-by-row scheduling. The algorithm is similar to the horizontal scheduling in[6]. However, two differences need to be considered. First, every row in the schedule can be regarded as the initial row in terms of minimizing switching activities, since we deal with cyclic DFG and the static schedule can be regarded as a repeatedly-executed cycle. Second, when processing the last row, we need to not only consider the second to the last row but also the first row in the next iteration, since both of them are fixed at that time. Because of the limited space, we omit the algorithm from the paper.

The complexity of SAMLS is $O(N * (|E| + |V|^2 \log |V|))$, where N is the rotation times, $|E|$ is the number of edges, and $|V|$ is the number of nodes. The detailed complexity analysis and the complete SAMLS algorithm can be found in [14].

5 Experiments

In this section, we experiment with the SAMLS algorithm on a set of benchmarks including 4-stage lattice filter, 8-stage lattice filter, differential equation solver, elliptic filter and voltera filter. The experiments are performed on a VLIW simulator with the similar architecture as TI C6000 DSP. The optimization problem

Input: DFG $G = \langle V, E, OP, d, t \rangle$, the retiming function r of G, an initial schedule S of G, the rotation times N.
Output: A new schedule S' and a new retiming function r'.
Algorithm:

1. **for** i=1 to N {
   (a) Put all nodes in the first row in S into a set R. Retiming each node $u \in R$ by $r(u) \leftarrow r(u) + 1$. Delete the first row from S and shift S up by one control step.
   (b) Reschedule the nodes in R by calling function BipartiteMatching_NodesSchedule(G,r,S,R) (Figure 4).
   (c) Minimize the switching activities of S row by row by calling function RowByRow_BipartiteMatching(S).
   (d) Store the obtained schedule and retiming function by $S_i \leftarrow S$ and $r_i \leftarrow r$.
   }
2. Select S_j from $S_1, S_2, \cdots, S_N$ such that S_j has the minimum switching activities among all minimum-latency schedules. Output results: $S' \leftarrow S_j$ and $r' \leftarrow r_j$.

Fig. 3. Algorithm SAMLS.

Input: DFG $G = \langle V, E, OP, d, t \rangle$, the retiming r of G, a schedule S, and a node set R.
Output: The revised schedule S.
Algorithm:

1. $Len \leftarrow$ the schedule length of S.
2. **while** (R is not empty) **do** {
   (a) Group all empty locations of S into blocks and let B be the set of all blocks. If B is empty, then let $Len \leftarrow Len + 1$; **Continue**.
   (b) Construct a weighted bipartite graph G_{BM} between node set R and block set B. $G_{BM} = \langle V_{BM}, E_{BM}, W \rangle$ in which: $V_{BM} = R \cup B$; for each $u \in R$ and $b_i \in B$, if u can be put into Block b_i, then $e(u, b_i)$ is added into E_{BM} with weight $W(e(u, b_i)) = Switch\_Block(u, b_i)$.
   (c) If E_{BM} is empty, then let $Len \leftarrow L + 1$; **Continue.**
   (d) Get the minimum cost maximum match M by calling function Min_Cost_Bipartite_Matching(G_{BM}).
   (e) Find edge $e(u, b_i)$ in M that has the minimal weight among all edges in M.
   (f) Assign u into the earliest possible location in Block b_i and remove u from set R.
   }

Fig. 4. Algorithm BipartiteMatching_NodesSchedule().

for reducing switching activities on the instruction bus is used in the experiments, and the real binary code of instructions from TI TMS320C6000 Instruction Set is used as $OP(u)$ for each node u.

Bench.	List		Rotation		PRRS [8]		SAMLS			
	SA	SL	SA	SL	SA	SL	SA	SA(%)	SL	SL(%)
4-Lattice	68	9	72	7	38	7	**34**	**50.0%**	**7**	**22.2%**
8-Lattice	108	17	118	11	68	11	**56**	**48.1%**	**11**	**35.3%**
DEQ	30	5	32	4	14	4	**12**	**60.0%**	**4**	**20.0%**
Elliptic	136	14	136	14	86	14	**72**	**47.1%**	**14**	**0.0%**
Voltera	70	12	68	12	38	12	**32**	**54.3%**	**12**	**0.0%**
Average Reduction (%) over List Scheduling							**51.9%**	–		**15.5%**

Table 1. The comparison of bus switching activities and schedule length for list scheduling, rotation scheduling, PRRS and SAMLS when FUs=4.

As shown in Table 1-2, we compare our results with those from list scheduling ("List"), the traditional rotation algorithm ("Rotation), the PRRS algorithm in [8] ("PRRS"), and the algorithms based on the approach in [6] (HV_Schedule). In the list scheduling, the priority of a node is set as the longest path from this node to a leaf node.

Bench.	HV_Schedule ([6])		SAMLS			
	SA	SL	SA	SA(%)	SL	SL(%)
4-Lattice	46	9	**34**	**26.1%**	**7**	**0.0%**
8-Lattice	64	17	**56**	**12.5%**	**11**	**17.6%**
DEQ	26	5	**12**	**53.8%**	**4**	**20.0%**
Elliptic	74	14	**72**	**2.7%**	**14**	**0.0%**
Voltera	42	12	**32**	**23.8%**	**12**	**0.0%**
Average Reduction (%)				**23.8%**	–	**15.5%**

Table 2. The comparison of bus switching activities and schedule length for SAMLS and the algorithms in[6] when FUs=4.

The experimental results for the list scheduling, the rotation scheduling, the PRRS algorithm from [8] and our SAMLS algorithm, are shown in Table 1 when the number of FUs is 4. Column "SA" presents the switching activity of the static schedule and Column "SL" presents the schedule length obtained from three different scheduling algorithms. Column "SL(%)" and "SA(%)" under "SAMLS" present the percentage of reduction in schedule length and switching activities respectively compared to the list scheduling algorithm. The average reduction is shown in the last row of the table. SAMLS shows an average 15.5% reduction in schedule length and 51.9% reduction in bus switching activities compared with the list scheduling.

To compare the performance between SAMLS and the algorithms in [6], we implement their horizontal scheduling and vertical scheduling and do experiments with window size 8. The experimental results for the various benchmarks are shown in Table 2 when the number of FUs is 4. In the table, "HV_Schedule" presents the algorithms in [6]. SAMLS shows an average 15.5% reduction in

schedule length and 23.8% reduction in bus switching activity compared with the algorithms in [6].

6 Conclusions

This paper studied low power loop scheduling problem and attempted to minimize both the schedule length and the power consumption for applications with loops on multiple-functional-unit architectures. We showed that to find a schedule that has the minimal switching activity among all minimum-latency schedules with or without resource constraints is NP-complete. An algorithm, SAMLS, was proposed. The algorithm minimizes both the switching activity and the schedule length based on rotation scheduling when performing the scheduling and allocation simultaneously. The experimental results show that our algorithm can greatly reduce switching activities and schedule length compared to the previous work.

References

1. Kim D., Shin D., Choi K.: Low Power of Linear Systems: A Common Operand Centric Approach. IEEE/ACM Int. Symp. on Low Power Electronics and Design (2001) 225–230
2. Yu T., Chen F., Sha E.: Loop Scheduling Algorithms for Power Reduction. IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (1998) 3073–3076
3. Raghunathan A., Jha N.: An ILP formulation for low power based on minimizing switched capacitance during data path allocation. IEEE Int. Symp. on Circuits & Systems (1995) 1069–1073
4. Musoll E.,Cortadella J.: Scheduling and resource binding for low power. IEEE Int. Symp. on System Synthesis (1995) 104–109
5. Chandrakasan A.,Sheng S.,Brodersen R.: Low-Power CMOS Digital Design. IEEE Journal of Solid-State Circuits 27(4) (1992)
6. Lee C., Lee J., Hwang T., Tsai S.: Compiler optimization on VLIW instruction scheduling for low power. ACM Transactions on Design Automation of Electronic Systems 8(2) (2003) 252–268
7. Chao L., LaPaugh A., Sha E.: Rotation Scheduling: A Loop Pipelining Algorithm. IEEE Trans. on Computer-Aided Design 16(3) (1997) 229–239
8. Shao Z., Zhuge Q., Sha E., Chantrapornchai C.: Loop Scheduling for Minimizing Schedule Length and Switching Activities. IEEE Int. Symp. on Circuits and Systems Vol. V (2003) 109–112
9. Yang H.,Gao G., Leung C.: On achieving balanced power consumption in software pipelined loops. Int. Conf. on Compilers, Architectures and Synthesis for Embedded Systems (2002) 210–217
10. Yun H., Kim J.: Power-aware modulo scheduling for high-performance VLIW processors. Int. Symp. on Low power electronics and design (2001) 40–45
11. Leiserson C., Saxe J.: Retiming Synchronous Circuitry. Algorithmica 6 (1991) 5–35
12. Garey M.,Johnson D.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company (1979)
13. Garey M.,Johnson D.: Some NP-complete Geometric Problems. ACM Symp. on Theory of Computing (1976) 10–22
14. Shao Z., Sha E.: Switching-Activity Minimization on Instruction-level Loop Scheduling for VLIW DSP Applications. Tech. Report (TR-0601-HSCL-UTD) University of Texas at Dallas (2004)