

# Algorithms and Analysis of Scheduling for Low-Power High-Performance DSP on VLIW Processors

Zili Shao, Qingfeng Zhuge, Youtao Zhang, Edwin H.-M. Sha

Department of Computer Science

University of Texas at Dallas

Richardson, Texas 75083, USA

{zxs015000, qfzhuge, zhangyt, edsha}@utdallas.edu

**Abstract**—Switching activity and schedule length are the two most important factors that influence the energy consumption of an application executed on a VLIW (Very Long Instruction Word) processor. Considering these two factors together, we propose an instruction-level energy-minimization scheduling technique to reduce the energy consumption of applications on VLIW processors. We first formally prove that this problem is NP-complete. Then three heuristic algorithms, MSAS, MLMSA, and EMSA, are proposed. While switching activity and schedule length are given higher priority in MSAS and MLMSA, respectively, EMSA gives the best result considering both of them. The experimental results show that EMSA gives a 31.7% reduction in energy compared with the traditional list scheduling approach on average.

**Index Terms**—DSP, VLIW, Scheduling, low power.

## I. INTRODUCTION

In embedded systems, high performance digital signal processing (DSP) used in image processing, multimedia, wireless security, etc., needs to be processed not only with high data throughput but also with low power consumption. To satisfy the ever-growing performance requirement, a high-performance architecture with multiple functional units such as VLIW (Very Long Instruction Word) architectures, for example, TI TMS320C6K, is now commonly used. Since multiple functional units are executed simultaneously in these architectures, power consumption becomes one of the most important issues to be considered with the concern of performance. Therefore, an efficient scheduling scheme is needed to reduce the energy consumption of an application as well as guarantee the required timing performance. Recent research for various processors [1], [2] shows that the instruction sequence of an application plays an important role in its energy consumption. Thus, new research directions in power optimization have begun to address the issues of instruction-level scheduling for reducing energy consumption [3]–[5]. In this paper, we

analyze and design efficient algorithms for various types of *instruction-level energy-minimization scheduling problem*, i.e., given a Directed Acyclic Graph (DAG) that models an application, how to schedule it to a VLIW architecture so that the energy consumption can be minimized while the required timing performance is satisfied.

A VLIW processor executes a *very-long instruction word* (called *long instruction word*) during each clock cycle. A *long instruction word* is composed of several parallel *sub-instructions*. The power consumption to execute a *long instruction word* during a clock cycle,  $P_{\text{cycle}}$ , can be computed by:

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

where  $P_{\text{base}}$  is the base power needed to support instruction execution,  $P_{\text{Inst}_i}$  is the basic power to execute a sub-instruction  $I_i$  on a functional unit, and  $SP(i, j)$  is the switching power caused by switching activities between  $\text{Inst}_i$  (current sub-instruction) and  $\text{Inst}_j$  (last sub-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

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

$\sum \sum P$  is the summation of basic power consumptions for all sub-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.

A lot of research efforts have been put in this field. In [6], compiler techniques are proposed to reduce power variations, hot-spots and peak power in scheduling. Though these techniques can efficiently reduce power variations, they don't aim to minimize the total energy consumption. Low power resource allocation approach [3], [7], [8] attempts to find an allocation for a fixed schedule in such a way that the total switching activities can be reduced. While this approach is effectively applied in resource allocation in the high-level synthesis, it may give inferior results in solving *instruction-level energy-minimization scheduling problem* because scheduling and allocation are performed separately. In [5], a two-phase scheduling approach is proposed to optimize transition activity in the instruction bus on a VLIW architecture. Based on an initial schedule, this approach performs low power resource allocation (called horizontal scheduling) in the first phase and vertically schedule instructions in the second phase to further reduce the switching activities. Since the schedule length is fixed priori, this approach can not be directly applied to solve *instruction-level energy-minimization scheduling problem*. In [4], several revised list scheduling techniques are proposed to minimize energy based on the instruction-level energy models for the specific processors. Using similar energy models, in [9], several energy-oriented instruction scheduling approaches are presented and compared with performance-oriented scheduling. However, these techniques are not general enough to be applied to VLIW processors because of the specific architectures.

Several techniques [10]–[12] are proposed to minimize switching activities only. In [10], an instruction scheduling technique, called cold scheduling, is proposed to reduce the switching activities on the control path. In [11], an operand sharing scheduling technique is proposed to schedule the operation nodes with the same operands as closely as possible to reduce the switching activities on the functional units. In [12], a scheduling algorithm for optimizing coefficients of a FIR filter is proposed to minimize the switching activities on memory data bus and function units. Since schedule length is not considered, these techniques may give inferior results for energy minimization.

In recent works [13], [14], the power efficient scheduling problem is formulated as the Traveling Salesman's Problem (TSP) and solved by heuristics of TSP when there is one FU. However, formulating a problem as TSP does not necessarily mean that the problem is NP-complete. For example, the problem to sequence jobs that require common resources on a single machine [15] can be transferred to TSP but still polynomially solvable. In the literature, it is still unknown that the instruction-level scheduling problem with minimum switching activities (called *minimum-switching-activities scheduling problem*) is solvable in polynomial time or NP-complete.

In our work, we formally prove *minimum-switching-activities scheduling problem* is NP-complete no matter there are resource constraints or not. Based on this result, we further prove that *instruction-level energy-minimization problem* is NP-complete with or without resource constraints. While minimum latency scheduling problem is polynomial-time solvable if there is only one FU or no resource constraints, the problem becomes NP-complete when considering switching activities as the second constraint.

To solve the *instruction-level energy-minimization problem* on VLIW architectures, three algorithms are proposed. We first propose two algorithms, *MSAS* and *MLMSA*, to solve two special cases of the energy-minimization problem. *MSAS* algorithm is designed for the case when switching activity plays the most important role in total energy consumption. *MLMSA* is for the case when schedule length plays the most important role. In both *MSAS* and *MLMSA* algorithms, we consider all nodes in the ready list in each schedule step and use *weighted bipartite matching* to do scheduling and allocation simultaneously. Then an algorithm, *Energy Minimization Scheduling Algorithm (EMSA)*, is proposed to solve the general *instruction-level energy-minimization scheduling problem*. In *EMSA*, we start from an initial schedule and reschedule the nodes to reduce switching activities when relaxing the initial schedule to a given timing constraint. The best schedule is selected from all possible schedules up to point. The experimental results show significant reductions in energy consumption. When switching activity plays the most important role in total energy consumption, *MSAS* gives a reduction of 41.2% in switching activity over list scheduling on average. When schedule length plays the most important role in total energy consumption, *MLMSA* keeps the same schedule length as list scheduling and reduces 33.2% switching activities on average. For the general case, *EMSA* gives a 31.7% reduction in energy compared with list scheduling on average.

The remainder of the paper is organized as follows. Section II introduces basic concepts and energy model. Section III proves *instruction-level energy-minimization scheduling problem* is NP-complete. The algorithms are discussed in Section IV. Experimental results and concluding remarks are provided in Section V and VI, respectively.

## II. BASIC CONCEPTS AND MODELS

In this section, we introduce the basic concepts that will be used in later sections. A motivational example and the energy model are also introduced in this section.

### A. DAG and Bit-String( $u$ )

A *Directed Acyclic Graph (DAG)*,  $G = \langle V, E \rangle$ , is a node-weighted graph, where  $V$  is the set of nodes and each node represents a sub-instruction, and  $E \subseteq V * V$  is the edge

set and an edge between two nodes denotes a dependency relation. In this paper, the operation time of each node in the DAG is assumed to be one time unit.

Assembly code that consists of *long instruction words* generated by a compiler has to satisfy the dependency relations in the DAG. It consists of long instruction words. A long instruction word contains multiple sub-instructions. Each sub-instruction specifies on which functional unit it is going to be executed. The bit switches between the consecutive sub-instructions executed on the same functional unit cause the switching power consumption ( $SP(i, j)$  in equation 1) in the power model introduced in Section I (the detail discussions about the power model will be given in Section II-B). Therefore, we can capture the switching power consumption ( $SP(i, j)$ ) through the computation of switching activities.

Instruction Loop	The Core Current	Instruction Loop	The Core Current
ADD A, B	42.5mA	MPY #1, A ADD A, B ADD A, B	46.6 mA
SUB A, B	42.5mA		
MPY #1, A	46.3mA	MPY #1, A ADD A, B SUB A, B	48.2 mA
ADD A, B SUB A, B	43.7mA		

Fig. 1. Some experimental results for TI TMS320C5410 DSP processor

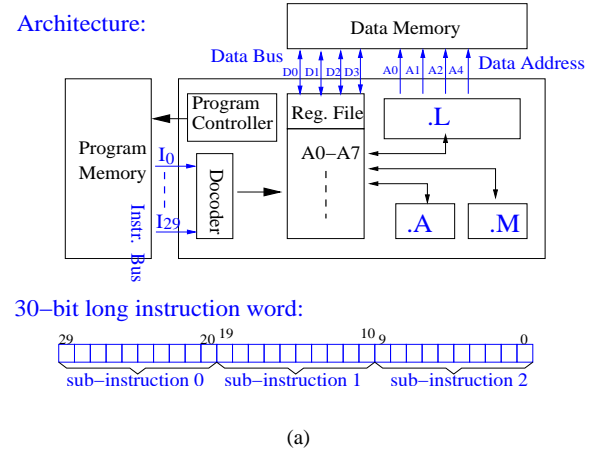
We define a function,  $Bit\_String(u)$ , to count the number of bit switches for each node  $u \in V$  in DAG  $G$ .  $Bit\_String(u)$  is a binary string that denotes the state of signal after  $u$  is executed. Without losing generality, it contains one or multiple of the following items:

- opcodes. When loading instructions from program memory, the state of instruction bus may be changed, which causes switching power consumption. Based on a series of experiments we did on TMS320VC5410 DSP processor, we collected the *core current* when executing the loops of different instructions and instruction combinations using similar method in [1]. The experimental results show that the core current increases correspondingly with the increase of the hamming distance of consecutive instructions. Some experimental results are shown in Figure 1. It confirms the correctness of our power model.
- addresses. To accessing operands, the states of data address bus may be changed and thus cause switching power consumption.
- operands. When the inputs are loaded from data memory and processed on functional units, the states of data bus, control paths in the functional units, etc., may be changed and cause switching power consumption. Since applications in embedded systems

are very specific, we can predict the possible values or patterns for the inputs priori. Note that an operand here is a possible state but not a real input.

- Don't Care. Don't Care means to keep the previous state. For example, if we execute an *add* instruction whose inputs are from two registers and output is put into a register, then the states of data bus and data address bus are not affected. Thus, these states should be kept.

A motivational example is shown in Figure 2-4. To schedule a Finite Impulse Response (FIR) filter on an experimental VLIW processor, we show how  $Bit\_String()$  can be used to denote the state of signal and how different schedules can cause different bit switches.



Instruction	Opcode (10 bits)
nop	0 0 0 0 0 0 0 0 0 0
ld (addr), reg	1 0 0 0 0 0 0 0 0 0 Address reg.
st reg, addr	1 0 1 0 0 0 0 0 0 0 reg. Address
add .A reg1, reg2	1 1 0 0 0 0 0 0 0 0 reg1 reg2
add .L reg1, reg2	1 1 0 1 0 0 0 0 0 0 reg1 reg2
add .M reg1, reg2	1 1 1 0 0 0 0 0 0 0 reg1 reg2
mpy reg1, reg2	1 1 1 1 0 0 0 0 0 0 reg1 reg2

(b)

Fig. 2. (a) A VLIW processor. (b) The instruction set.

The architecture of our experimental VLIW processor is shown in Figure 2(a). It is a simplified model and similar to one cluster of TI C6000 VLIW processors.

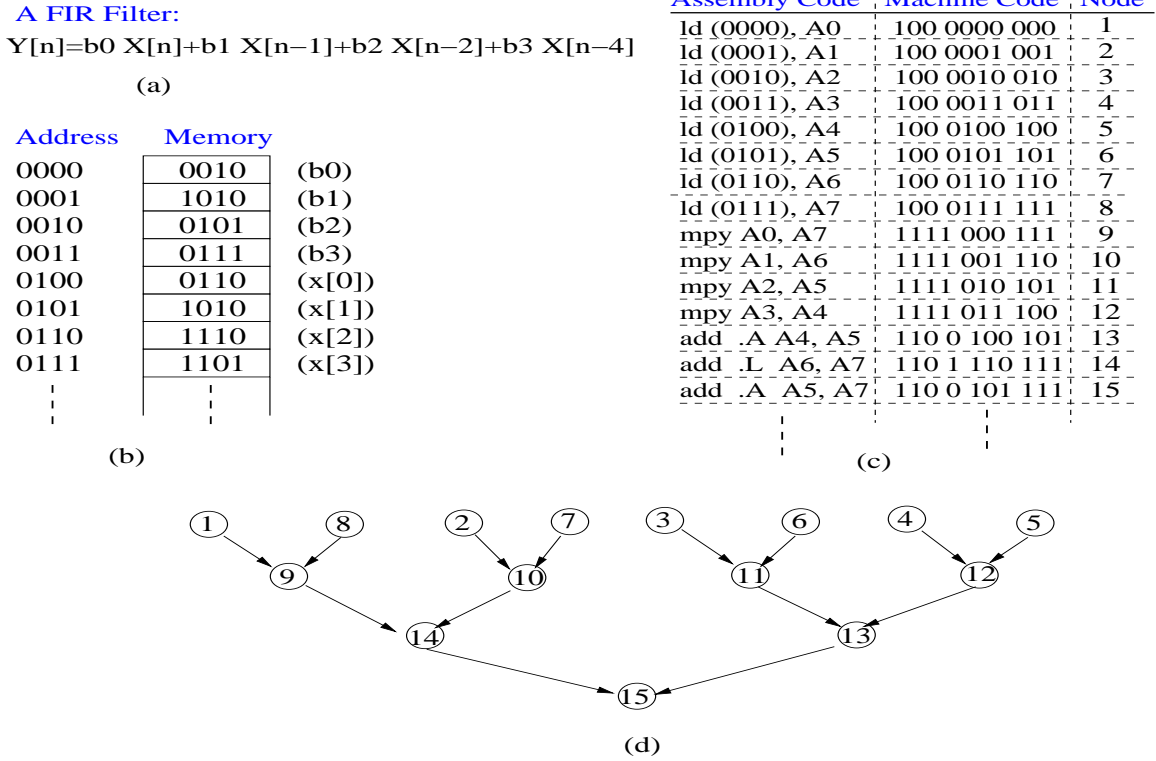


Fig. 3. (a) A FIR filter (b) The coefficients and inputs in data memory (c) The assembly program, opcodes and node id (d) The DAG representation

It uses Harvard architecture with separate memory space for program and data. For the illustration purpose, we assume that program memory has a 30-bit instruction bus and data memory has a 4-bit address bus and data bus. Thus, in each clock cycle, a 30-bit long instruction word composed of 3 sub-instructions is loaded to the decoder and the sub-instructions are redirected to corresponding functional units to be executed after decoded. There are 3 functional units: *.L*, *.M*, *.A*, where *.L* can perform load/store or addition operation, *.M* can perform multiplication or addition operation, and *.A* can only perform addition operation. According to the specifications of the functional units, a long instruction word has three possible combinations. One is to contain one load/store instruction, one multiplication instruction, and one addition instruction. Another is to contain two addition instructions and one load/store or multiplication instruction. The third one is to contain 3 addition instructions. The instructions and the corresponding opcodes are shown in Figure 2(b). We assume the operation time of each instruction is one time unit. Direct addressing is used by load/store instructions. Multiplication (*mpy*) and addition (*add*) operations are performed on general purpose registers  $A_0 \sim A_7$ . Their inputs are from two registers and the result is stored into the second register. For example, *add A<sub>1</sub>, A<sub>2</sub>* will store the result into  $A_2$ .

A FIR filter is shown in Figure 3(a). Figure 3(b) shows

the addresses and the values of its coefficients ( $b_0 \sim b_3$ ) and its first 4 inputs ( $X[0] \sim X[3]$ ). The assembly codes, the corresponding opcodes and node names are shown in Figure 3(c), in which the coefficients and inputs are first loaded into  $A_0 \sim A_7$  and then computed. The DAG representation for the assembly program is shown in Figure 3(d). Since bus capacitances are usually several orders of magnitude higher than those of the internal nodes of a circuit [16], a considerable amount of power can be saved by reducing the switching activities on the buses. Therefore, we will only consider reducing the transitions on the instruction bus, data address bus and data bus in this example. For each node  $u$ , *Bit\_String(u)* denotes the states of three buses after executing node  $u$ . It consists of three parts: the opcode on the instruction bus, the operand on the data bus, and the access address on the data address bus as shown in Figure 4(a). Usually we don't know the inputs at the compiling time. However, we can predict their possible values or patterns for a specific application. In this example, we assume our prediction for the operand part in *Bit\_String(u)* is totally correct. Since multiplication, addition and nop operations can not influence the states of data memory, we put "—" under column "data addr." and "data bus" to denote the state of "Don't Care". Two different schedules are shown in Figure 4(b) and (c), respectively. Schedule 1 is obtained by traditional list scheduling and Schedule 2 is obtained by our

Node	Bit_String( i )		
	Opcode	Data Addr.	Data Bus
1	100 0000 000	0000	0010
2	100 0001 001	0001	1010
3	100 0010 010	0010	0101
4	100 0011 011	0011	0111
5	100 0100 100	0100	0110
6	100 0101 101	0101	1010
7	100 0110 110	0110	1110
8	100 0111 111	0111	1101
9	1111 000 111	_____	_____
10	1111 001 110	_____	_____
11	1111 010 101	_____	_____
12	1111 011 100	_____	_____
13	110 0 100 101	_____	_____
14	110 1 110 111	_____	_____
15	110 0 101 111	_____	_____
nop	00000 00000	_____	_____

(a)

Sch. Step	Instruction Bus			Data Addr.	Data Bus
	Sub-instru. 0	Sub-instru. 1	Sub-instru. 2		
	0000000000	0000000000	0000000000	0000	0000
1	1 (1000000000)	nop	nop	0000	0010
2	8 (1000111111)	nop	nop	0111	1101
3	2 (1000001001)	9 (1111000111)	nop	0001	1010
4	7 (1000110110)	nop	nop	0110	1110
5	3 (1000010010)	10 (1111001110)	nop	0010	0101
6	6 (1000101101)	14 (1101110111)	nop	0101	1010
7	4 (1000011011)	11 (1111010101)	nop	0011	0111
8	5 (1000100100)	nop	nop	0100	0110
9	12 (1111011100)	nop	nop	_____	_____
10	13 (1100100101)	nop	nop	_____	_____
11	15 (1100101111)	nop	nop	_____	_____

(b)

Sch. Step	Instruction Bus			Data Addr.	Data Bus
	Sub-instru. 0	Sub-instru. 0	Sub-instru. 0		
	0000000000	0000000000	0000000000	0000	0000
1	1 (1000000000)	nop	nop	0000	0010
2	2 (1000001001)	nop	nop	0001	1010
3	6 (1000101101)	nop	nop	0101	1010
4	5 (1000100100)	nop	nop	0100	0110
5	7 (1000110110)	nop	nop	0110	1110
6	8 (1000111111)	10 (1111001110)	nop	0111	1101
7	4 (1000011011)	9 (1111000111)	nop	0011	0111
8	3 (1000010010)	12 (1111011100)	13 (1100100101)	0010	0101
9	nop	11 (1111010101)	nop	_____	_____
10	nop	14 (1101110111)	nop	_____	_____
11	nop	15 (1100101111)	nop	_____	_____

(c)

Fig. 4. (a) *Bit\_String()* (b) Schedule 1 (c) Schedule 2

MSAS algorithm. Assume the initial states of buses are all 0's. Then the total number of bit switches for each bus is the total number of transitions for a schedule. The number of transitions for Schedule 1 is 121 and that for Schedule 2 is 66. Since they have the same schedule length, Schedule 2 consumes less energy than Schedule 1 when executed on our experimental VLIW processor. The generated codes with long instruction words can be found in the column of instruction bus in Figure 4(b) and (c).

### B. Power Cost Function and Energy Model

Our energy model is based on the methodology for the instruction-level energy estimation framework for VLIW architecture proposed in [17]–[20]. In VLIW processors such as TI TMS320C620x and TMS320C670x DSP devices, there is no data cache and application programs are loaded into on-chip-memory before they are executed [21], therefore the influence of cache miss is not considered in our energy model.

As shown in equation 1, the power consumption to execute a *long instruction word* during a clock cycle,  $P_{cycle}$ , can be computed by:

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

In VLIW processors,  $P_{base}$  denotes the power to support basic pipeline execution even when a *long instruction word* contains only NOPs.  $P_{Inst_i}$  denotes the basic power for a functional unit to execute a sub-instruction on a specific datapath even when the inputs don't cause any transition.  $SP(i, j)$  denotes the switching power caused by switching activities between  $Inst_i$  (current sub-instruction) and  $Inst_j$  (last sub-instruction) executed on the same FU.

The switching power is proportional to the number of transitions. So

$$SP(i, j) = \alpha \cdot WHD(Bit\_String(Inst_i), Bit\_String(Inst_j)) \quad (3)$$

where  $\alpha$  is a power coefficient representing the consumed power per transition, and WHD (Weighted Hamming Distance) is a function used to compute the number of transitions between  $Inst_i$  and  $Inst_j$ . Let  $X=Bit\_String(Inst_i)$  and  $Y=Bit\_String(Inst_j)$ , we define  $WHD(X, Y)$  as:

$$WHD(X, Y) = \sum_{i=1}^K w_i \cdot (X[i] \oplus Y[i]) \quad (4)$$

where  $w_i$  is the weight of a transition.  $w_i$  is used to denote the weight for the power consumption caused by one transition on different units.

The energy consumption of an application is the summation of all its power consumption during each clock cycle. Let  $S$  be a schedule for an application and  $L$  the schedule length of  $S$ . Then the energy consumption of Schedule  $S$ ,

$E_S$ , can be computed by

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

$\sum \sum P$  is the summation of basic power consumptions for all sub-instructions of an application that does not change with different schedules.  $P_{base}$  is a constant and  $P_{base} * L$  varies with the schedule length for a specific VLIW processor.  $\sum \sum SP(i, j)$  is the switching power and changes with different schedules. Therefore, schedule length and switching activity need to be considered together in instruction-level scheduling techniques in order to minimize energy consumption of an application. One example is shown in Figure 5.

In Figure 5(a), a DAG is given, in which the binary string close to each node  $u$  is the value of  $Bit\_String(u)$ . Given 2 functional units, FU1 and FU2, three different schedules are shown in Figure 5(b)–(d). An empty slot denotes a NOP node. We assume a NOP node doesn't cause any transition. Schedule 1 and Schedule 2 have the same schedule length but Schedule 2 has less bit switches. Thus we can easily identify Schedule 2 has less energy consumption than Schedule 1 has. To compare the energy consumptions of Schedule 2 and Schedule 3, we need to consider the schedule length. Assume that the power coefficient  $\alpha$  in equation 3 is 1 and  $P_{base}$  is 1 in equation 2. The energy for schedule 3 is  $6+4+\sum \sum P = 10+\sum \sum P$  and that for Schedule 2 is  $5+8+\sum \sum P = 13+\sum \sum P$ . So Schedule 3 uses less energy than Schedule 2. However, when  $P_{base} = 5$ , the energy for Schedule 3 is  $6*5+4+\sum \sum P = 34+\sum \sum P$  and that for Schedule 2 is  $5*5+8+\sum \sum P = 33+\sum \sum P$ . Then Schedule 2 uses less energy. Thus, we have to consider both schedule length and switching activity when comparing the energy consumption of schedules.

### C. Weighted Bipartite Matching

In our algorithms, we use the weighted bipartite matching graph to help find the binding of nodes in a DAG and resources. A *Weighted Bipartite Graph*  $G_{BM} = \langle V, E, W \rangle$  is an edge-weighted undirected graph, where  $V$  can be partitioned into two sets  $V_1$  and  $V_2$  such that each edge  $e(u, v) \in E$  implies  $u \in V_1$  and  $v \in V_2$  or  $u \in V_2$  and  $v \in V_1$ , and  $W(e)$  represents the weight of edge  $e \in E$ .

Given a bipartite graph  $G_{BM} = \langle V, E, W \rangle$ , a *matching*  $M$  for  $G_{BM}$ , is a subset of  $E$  such that for each node  $v \in V$ , at most one edge of  $M$  is incident on  $v$ . Hence, the weight of a matching is the summation of the weight of all edges in  $M$ . A *maximum matching* is a matching of

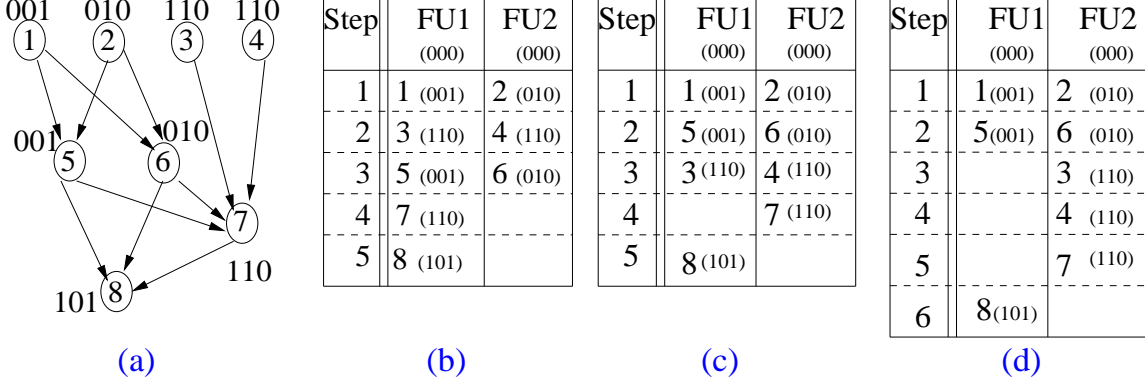


Fig. 5. (a) A given DAG (b) Schedule 1 with total bit switches 15 (c) Schedule 2 with total bit switches 8 (d) Schedule 3 with total bit switches 4

maximum cardinality, i.e., a matching  $M$  such that for any matching  $M'$ , we have  $|M| \geq |M'|$ , where  $|M|$  denotes the cardinality of  $M$ . A *min-cost maximum matching* is a maximum matching with a minimum weight.

#### D. The Optimization Problem

Our optimization problem is defined as follows: *Given a DAG  $G = \langle V, E \rangle$ , a function  $\text{Bit\_String}(u)$  for each node  $u \in V$ , and  $M$  functional units, find a schedule  $S$  of  $G$  that has minimum energy consumption based on energy model in Section II-B.*

### III. NP-COMPLETENESS

In this section, we prove that our optimization problem is NP-complete.

From the energy model in equation (2), we know the energy consumption of a schedule is related to schedule length and switching activity. To prove our optimization problem is NP-complete, we use the following method. We first consider two subproblems of our optimization problem: *minimum-latency scheduling problem* that only minimizes schedule length and *minimum-switching-activities scheduling problem* that only minimizes switching activities. If we can prove one of these two problems is NP-complete, we can further prove our optimization problem is NP-complete. It is well-known that *minimum-latency scheduling problem* is polynomial-time solvable when there is only one functional unit or no resource constraints. Therefore, the focus is put on *minimum-switching-activities scheduling problem*. As discussed in Section I, it is still unknown that *minimum-switching-activities scheduling problem* is solvable in polynomial time or NP-complete in the literature. Thus, we categorize the problem into two cases as shown in Theorem 3.1 and Theorem 3.2 and prove them as follows.

**Theorem 3.1:** Let  $U$  be the number of resources, when  $U = 1$ , min-switching-activities scheduling problem is NP-complete.

In order to prove Theorem 3.1, we first define two decision problems: one is the decision problem (*DPI*) of *minimum-switching-activities scheduling problem* when  $U = 1$  and the other is the  $L_1$  *Geometric Traveling Salesman Problem (GTSP)*. We will transfer GTSP to our problem in the proof.

**DP1:** Given a DAG  $G = \langle V, E \rangle$ , a function  $\text{Bit\_String}(u)$  for each node  $u \in V$ , one resource, and one constant  $H$ , does there exist a schedule that has switching activities at most  $H$ ?

**The  $L_1$  geometric traveling salesman problem (GTSP) [22]:** 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 *DPI* belongs to NP. Assume that  $S = \{[x_1, y_1], [x_2, y_2], \dots, [x_n, y_n]\}$  is an instance of *GTSP*. Construct DAG  $G = \langle V, E \rangle$  as follows:  $V = \langle v_1, v_2, \dots, v_n \rangle$  where  $v_i$  corresponds to a point  $[x_i, y_i]$  in  $S$  and  $E = \emptyset$ . Assume that  $X = \max(x_i)$  and  $Y = \max(y_i)$  for  $1 \leq i \leq n$ , then  $\text{Bit\_String}(v_i) = (X - x_i)0's \bullet x_i 1's \bullet (Y - y_i)0's \bullet y_i 1's$  for each  $v_i \in V (1 \leq i \leq n)$ , where “ $\bullet$ ” denotes concatenation. For example, if  $X = Y = 3$ ,  $x_1 = 2$ , and  $y_1 = 1$ , then  $\text{Bit\_String}(v_1) = 011 001$ . Set  $H = L$ . Since *GTSP* is NP-complete and the reduction can be done in polynomial time, *DPI* is NP-complete. ■

**Theorem 3.2:** Let  $U$  be the number of resources, when  $U > 1$ , min-switching-activities scheduling problem is NP-complete.

The decision problem (*DP2*) of *minimum-switching-activities scheduling problem* when  $U > 1$  is similar to *DPI* except the number of resource is greater than 1 in *DP2*. The proof of Theorem 3.2 is as follows.

*Proof:* It is obvious *DP2* belongs to NP. Assume that  $S = \{[x_1, y_1], [x_2, y_2], \dots, [x_n, y_n]\}$  is an instance of *GTSP*. Construct DAG  $G = \langle V, E \rangle$  as follows:  $V = \langle v_1, v_2, \dots, v_n \rangle$  where  $v_i$  corresponds to a point  $[x_i, y_i]$  in  $S$  and  $E = \emptyset$ . Assume that  $X = \max(x_i)$  and  $Y = \max(y_i)$  for  $1 \leq i \leq n$ , then  $\text{Bit\_String}(v_i) = (X + Y +$

2)1's • (X - x<sub>i</sub>)0's • x<sub>i</sub>1's • (Y - y<sub>i</sub>)0's • y<sub>i</sub>1's for each v<sub>i</sub> ∈ V (1 ≤ i ≤ n). For example, if X = Y = 3, x<sub>1</sub> = 2, and y<sub>1</sub> = 1, then Bit\_String(v<sub>1</sub>)=11111111 011 001. Set H = L. Set the initial Bit\_String() of each resource to all 0's.

The construction of V makes all nodes in V be assigned to the same resource for minimizing switching activities. Since the reduction can be done in polynomial time, DP2 is NP-complete. ■

From Theorem 3.1 and 3.2, we know *minimum-switching-activities scheduling problem* is NP-complete. Then we use it to prove our optimization problem is NP-complete.

*Theorem 3.3:* Our optimization problem for instruction-level energy-minimization scheduling is NP-complete.

*Proof:* Given an instance of *minimum-switching-activities scheduling problem*, it can be directly mapped as an instance of *our optimization problem* by setting P<sub>base</sub>=0 in the energy model. Since the reduction can be done in polynomial time, *our optimization problem* is NP-complete. ■

#### IV. THE ALGORITHMS

In this section, we present three algorithms to solve *instruction-level energy-minimization scheduling problem*. Two algorithms for two special cases are presented first and then another algorithm to solve general problem is proposed.

##### A. MSAS Algorithm

1) *The Algorithm:* Our first algorithm, Minimizing Switching Activities Scheduling (MSAS), is designed to solve a special case of *instruction-level energy-minimization scheduling problem*, i.e., the case when the switching activities play the most important role in energy consumption.

When P<sub>base</sub> is very small compared with α (in equation 3), the energy of a schedule depends mainly on switching activities. For example, when P<sub>base</sub> equals 0.1 in Figure 5, we need to reduce 10 control steps in schedule length to count one bit switch. Thus, we need an algorithm to minimize switching activities as much as possible. On the other side, considering the performance, we also want to minimize schedule length. Hence, MSAS algorithm minimizes switching activities in first priority and still considers schedule length. Since most previous works focus on one functional unit, we need algorithms that can take advantage of multiple functional units under VLIW architectures. MSAS algorithm is shown in Algorithm IV.1.

Due to the existence of the dependency in a DAG, we can only schedule a node after all its parent nodes have been scheduled. The scheduling problem with switching activities minimization is how to find a matching between functional units and ready nodes in such a way that the

---

##### Algorithm IV.1 MSAS (Min-Switching-Activities Scheduling)

---

**Input:** DAG G = ⟨V, E⟩, Bit\_String(u) for each u ∈ V, functional unit set FU\_SET

**Output:** A schedule with switching activities minimization

L<sub>RD</sub> ← All ready nodes in G;

**while** L<sub>RD</sub> ≠ ∅ **do**

Construct G<sub>BM</sub> = ⟨V<sub>BM</sub>, E, W⟩, where: V<sub>BM</sub> = FU\_SET ∪ L<sub>RD</sub>; E = {(F<sub>i</sub>, u) | ∀F<sub>i</sub> ∈ FU\_SET, u ∈ L<sub>RD</sub>} and W(F<sub>i</sub>, u) = WHD(Bit\_String(u), Bit\_String(F<sub>i</sub>)); M ← Min\_Cost\_Bipartite\_Matching(G<sub>BM</sub>);

**for all** e(F<sub>i</sub>, u) ∈ M **do**

Schedule Node u to functional unit F<sub>i</sub>

Bit\_String(F<sub>i</sub>) ← Bit\_String(u);

Remove u from L<sub>RD</sub>;

Check all adjacent nodes of u and put new ready nodes into L<sub>RD</sub>;

**end for**

**end while**

---

schedule based on this matching minimizes the total switching activities in every scheduling step. This is equivalent to the min-cost weighted bipartite matching problem. Thus, in MSAS algorithm, we repeatedly create a weighted bipartite graph G<sub>BM</sub> between the set of functional unit and the set of nodes in Ready\_List, and assign nodes based on the min-cost maximum bipartite matching M. In each scheduling step, the weighted bipartite graph, G<sub>BM</sub> = ⟨V<sub>BM</sub>, E<sub>BM</sub>, W⟩, is constructed as follows: V<sub>BM</sub> = FU\_SET ∪ L<sub>RD</sub> where FU\_SET = ⟨F<sub>1</sub>, F<sub>2</sub>, ..., F<sub>N</sub>⟩ is the set of all functional units and L<sub>RD</sub> is the set of all nodes in Ready\_List; for each FU F<sub>i</sub> ∈ FU\_SET and each node u ∈ L<sub>RD</sub>, an edge e(F<sub>i</sub>, u) is added into E<sub>BM</sub> and W(F<sub>i</sub>, u) = WHD(Bit\_String(u), Bit\_String(F<sub>i</sub>)), where WHD(X, Y) is weighted hamming distance function as defined in equation 4 and Bit\_String(F<sub>i</sub>) = Bit\_String(v) where v is the last node executed on F<sub>i</sub>.

Since bit switches are set as the weight of edge, a schedule based on this matching optimally minimizes the switch activities at every scheduling step. All nodes in M are scheduled at every scheduling step, so the schedule length can not be too big. After scheduling a node u to a FU F<sub>i</sub>, we update the state of signal on F<sub>i</sub> by setting Bit\_String(F<sub>i</sub>) = Bit\_String(u). Note that there may exist "Don't Care" in Bit\_String(u). In such case, we need to put the current state instead of "Don't Care" in order to correctly count bit switches on F<sub>i</sub>. An example is shown in Figure 6.

Given the DAG in Figure 5(a), the scheduling in the first step by MSAS algorithm is shown in Figure 6 when there are 2 FUs. Ready\_List of the DAG in the first step is shown in Figure 6(a). Assume that the initial states of FUs are 000 and w<sub>i</sub> = 1 in weighed hamming distance function WHD(.). A weighted bipartite graph based on the set of FUs and the set of nodes in Ready\_List is constructed in Figure 6(b). A min-cost maximum bipartite matching is shown in Figure 6(c). The schedule of the first step is shown

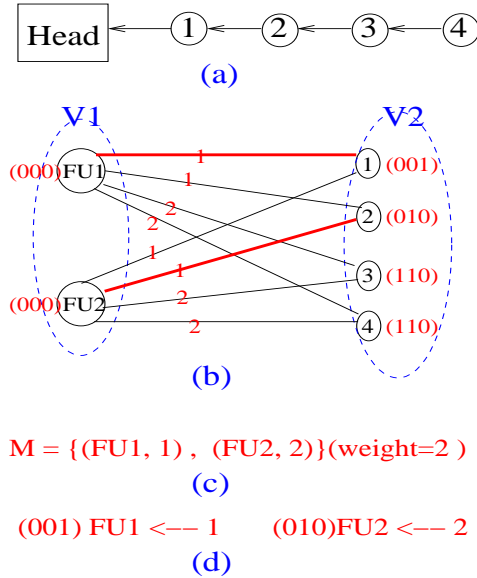


Fig. 6. (a) Ready\_List (b) Weighted Bipartite Graph (c) Min-cost Max Bipartite Matching (d) The schedule for first step

in Figure 6(d). After scheduling Node 1 to  $FU_1$  and Node 2 to  $FU_2$ , we change the states of the FUs. The final schedule generated by MSAS is shown in Figure 5(c). Compared with Schedule 1 in Figure 5(b) generated by traditional list scheduling, Schedule 2 has less bit switches.

It is known that finding a min-cost maximum bipartite matching takes  $O(n^3)$  by the Hungarian Method [23]. Let  $N$  be the number of functional units. In every scheduling step, we need at most  $O((N + |V|)^3)$  to find minimum weight maximum bipartite matching using Hungarian Method and the scheduling step is at most  $|V|$ . Thus, the complexity of MSAS is  $O(|V| * (N + |V|)^3)$ .

2) *Applying MSAS Algorithm to VLIW architecture:* Our generic algorithm can be easily modified to solve the scheduling problem on various VLIW architectures. In this section, we show how to apply MSAS to the VLIW architecture similar to TI C6000 VLIW processor. In such architecture, a sub-instruction can be put into any location in the instruction bus. It is the decoder that distributes the sub-instruction to a FU. So we need to change our generic algorithm a little bit to handle this case. The VLIW architecture consists of heterogeneous FUs with resource constraints. Thus, the number of sub-instructions of a type executing in a clock cycle can not exceed the resource bound. For example, since there is only one load/store unit in the experimental VLIW processor in Figure 2, we can not schedule more than one load/store instruction at the same schedule step. Assume that at most  $N$  sub-instructions of a type can be executed in a clock cycle. If there are more than  $N$  sub-instructions of this type in the bipartite matching in a schedule step in MSAS, we can sort the edges with this type of sub-instructions by an increasing order and schedule

these sub-instructions from the first  $N$  edges. Then we can remove all sub-instructions of this type from Ready\_List and find the best matching for the left FUs and left nodes till every FU has been assigned a node. In such a way, we can reduce switching activity as much as possible. We also need to add NOP nodes if the number of nodes is less than the number of FUs in each step. Using  $F_i$  to denote sub-instruction  $i$  in the instruction bus and applying above, Schedule 2 in Figure 4(c) is generated by MSAS. It reduces bit switches from 121 of Schedule 1 to 66.

### B. MLMSA Algorithm

Our second algorithm, Minimizing Latency with Minimal Switching Activities Scheduling (MLMSA), is designed to solve another special case of *instruction-level energy-minimization scheduling problem*, i.e., the case when  $P_{base}$  is very big compared with  $\alpha$  in equation 3 (the switching power caused per transition). In this case, the schedule length plays the most important role in energy consumption. Thus, MLMSA is designed to generate a schedule with minimum schedule length in the first priority and then reduce switching activities as much as possible.

MLMSA is similar to MSAS except that we use the different method to assign weights to the edges of weighted bipartite graph. In order to reduce the schedule length, we define  $Depth(u)$ , a priority function for each node  $u$  in a DAG, to be the longest path from  $u$  to a leaf node in a DAG. When constructing weighted bipartite graph  $G_{BM}$ , for each edge  $e(F_i, u) \in G_{BM}$ , its weight is set as  $W(u, F_i) = WHD(Bit\_String(u), Bit\_String(F_i)) - w_{max} \times Depth(u) \times Len$ , where  $Depth(u)$  is the priority value of node  $u$  as mentioned above, and  $Len$  is the bit length of  $Binary\_string(u)$ , and  $w_{max} = \max_i w_i$  for each  $w_i$  in equation 4.

Using this method, the priority of a node is dominant in edge weight in our bipartite graph  $G_{BM}$ . Thus, the node with the highest priority in list  $L_{RD}$  will be scheduled first. However, for the nodes with same priority, switching activities minimization is considered. Thus, we minimize the schedule length while considering switching activities.

### C. EMSA Algorithm

In this section, an algorithm, *Energy Minimization Scheduling Algorithm (EMSA)*, is proposed to solve the general *instruction-level energy-minimization scheduling problem*. The basic idea is to reschedule nodes to reduce switching activities when relaxing the schedule length of an initial schedule to a given timing constraint and then select the schedule with minimal energy consumption. EMSA is shown in Algorithm IV.2.

In EMSA, a function, *Relax\_Reschedule*, is called to generate a new schedule for each  $t$  from  $L$  to  $TC$ , where  $L$  is the schedule length of the given initial schedule

---

**Algorithm IV.2** EMSA (Energy Minimization Scheduling Algorithm)

---

**Input:** DAG  $G = \langle V, E \rangle$ ,  $Bit\_String(u)$  for each  $u \in V$ , functional unit set  $FU\_SET$ , an initial schedule  $S_{init}$ , a timing constraint  $TC$

**Output:** A schedule with minimum energy  
 $L \leftarrow$  the schedule length of  $S_{init}$ ;  $i \leftarrow 1$ ;  
**for all**  $t = L$  to  $TC$  **do**  
 $S_i \leftarrow Relax\_Reschedule(G, Bit\_String, FU\_SET, S_{init}, t)$ ;  
**end for**  
 Select the schedule with minimum energy among  $S_i (1 \leq i \leq (TC - L + 1))$ ;

---

and  $TC$  is the timing constraint. The schedule generated by  $Relax\_Reschedule$  has minimum switching activities because the switching activities are reduced as much as possible in  $Relax\_Reschedule$ . For each schedule, we compute its energy consumption based on equation 2. Then the schedule with minimum energy consumption is selected as the output.

Function  $Relax\_Reschedule$  reschedules nodes to reduce switching activities during relaxing the initial schedule as shown in Algorithm IV.3.  $Relax\_Reschedule$  works mostly like As-Late-As-Possible scheduling. In  $Relax\_Reschedule$ , we reschedule nodes to new locations based on an initial schedule and a timing constraint from the bottom up. Since the timing constraint is equal to or greater than the schedule length of the initial schedule, the nodes in the initial schedule may have some freedom to be moved to a new location such that the switching activities can be reduced. In order to obey the precedence relation, we associate two variables,  $Latest\_Step(u)$  and  $Post\_Order(u)$ , to each node  $u \in G$ , where  $Latest\_Step(u)$  keeps track of the latest schedule step, and  $Post\_Order(u)$  keeps track of the number of child nodes of  $u$  that have not been rescheduled yet. We put all nodes with  $Post\_Order = 0$  into set  $Ready\_Set$  and compute the best location of each node with two considerations: 1) it causes the greatest reduction in switching activity; 2) it is the closest to the bottom. Then we select the node with the greatest reduction in switching activity among all nodes in  $Ready\_Set$  and reschedule it to its best location. In this way, we can reduce switching activities as much as possible when rescheduling a node.

Since the schedule generated by EMSA is directly related to the initial schedule. It is very important to have a good initial schedule. Both MSAS and MLMSA greatly reduce switching activities, so we can use the schedules generated by them as the initial schedules of EMSA. For example, using Schedule 2 in Figure 5(c) as an initial schedule and assuming the timing constraint is 6 time units, Schedule 2 and Schedule 3 (Figure 5(d)) are obtained by EMSA. The best schedule is selected based on our energy model shown in equation 2. When the power coefficient  $\alpha$  (equation 3) is 1 and  $P_{base}$  is 1 in equation 2, Schedule 3

---

**Algorithm IV.3**  $Relax\_Reschedule()$

---

**Input:** DAG  $G = \langle V, E \rangle$ ,  $Bit\_String(u)$  for each  $u \in V$ , functional unit set  $FU\_SET$ , an initial schedule  $S_{init}$ , a timing constraint  $TC$

**Output:** A schedule with switching activities minimization  
**for all**  $u \in G$  **do**  
 $Latest\_Step(u) \leftarrow TC$ ;  
 $Post\_Order(u) \leftarrow$  the number of child nodes of  $u$ ;  
**end for**  
 $Ready\_Set \leftarrow$  all nodes with  $Post\_Order = 0$  in  $G$ ;  
**while**  $Ready\_Set \neq \emptyset$  **do**  
 $u \leftarrow$  the node with minimum  $Min\_SW(u)$  among all nodes in  $Ready\_Set$ ;  
 $Reschedule\ u\ to\ Best\_Loc(u)$ ;  
**for all** parent node  $v_p$  of  $u$  in  $G$  **do**  
 $Sch\_Step \leftarrow$  the schedule step of  $v$ ;  
**if**  $Latest\_Step(v_p) > Sch\_Step - 1$  **then**  
 $Latest\_Step(v_p) \leftarrow Sch\_Step - 1$ ;  
**end if**  
 $Post\_Order(v_p) - -$ ;  
**end for**  
 Remove  $u$  from  $Ready\_Set$  and put new ready nodes with  $Post\_Order = 0$  into  $Ready\_Set$ ;  
**end while**

---

is selected as the final schedule.

## V. EXPERIMENTS

We experimented with our algorithms on a set of benchmarks including 2-cascaded Biquad filter, 4-stage lattice filter, 8-stage lattice filter, differential equation solver, elliptic filter and voltera filter. When doing experiments with EMSA, the schedules generated by MSAS and MLMSA are used as the initial schedules, and the best results are selected. When computing the switching power, we assume  $\alpha = 1$  in equation 3 and  $w_i = 1$  for each  $i$  in equation 4. The results are compared with those of the traditional list scheduling algorithm. In our experiments, the opcodes for each node is obtained from TI TMS320C6000 Instruction Set [24].

The experimental results for MSAS and list scheduling are shown in Table I. Column “SA” lists the switching activities for list scheduling (field “ListS”) and MSAS (field “MSAS”) when the number of functional units varies from 3 to 6. Column “SL” lists the schedule length. Column “%” under MSAS lists the percentage of reduction

Bench.	FUs=3					FUs=4					FUs=5					FUs=6				
	ListS		MSAS			ListS		MSAS			ListS		MSAS			ListS		MSAS		
	SL	SA	SL	SA	%	SL	SA	SL	SA	%	SL	SA	SL	SA	%	SL	SA	SL	SA	%
2-Cas-Biq	7	54	8	27	50.0	6	40	7	33	17.5	6	45	7	28	37.8	6	45	7	28	37.8
4-Lat-IIR	9	64	12	35	45.3	9	69	11	35	49.3	9	68	10	37	45.6	9	62	10	34	45.2
8-Lat-IIR	17	96	23	43	55.2	17	96	22	43	55.2	17	102	20	43	57.8	17	103	19	45	56.3
DEQ	5	40	6	21	47.5	5	30	5	27	10.0	5	35	5	27	22.9	5	35	5	27	22.9
Elliptic	15	143	16	90	37.1	14	142	14	93	34.5	14	142	14	85	40.1	14	142	14	85	40.1
Voltera	12	60	15	31	48.3	12	57	14	29	49.1	12	56	13	32	42.9	12	57	13	34	40.4
Average Reduction %					47.2	-				35.9	-				41.2	-				40.4

TABLE I

THE COMPARISON ON SWITCHING ACTIVITY AND SCHEDULE LENGTH FOR MSAS AND LIST SCHEDULING WHEN THE NUMBER OF FUS VARIES FROM 3 TO 6.

Bench.	FUs=3					FUs=4					FUs=5					FUs=6				
	ListS		MSAS			ListS		MSAS			ListS		MSAS			ListS		MSAS		
	SL	SA	SL	SA	%	SL	SA	SL	SA	%	SL	SA	SL	SA	%	SL	SA	SL	SA	%
2-Cas-Biq	7	54	7	28	48.1	6	40	6	29	27.5	6	45	6	23	48.9	6	45	6	30	33.3
4-Lat-IIR	9	64	9	46	28.1	9	69	9	48	30.4	9	68	9	40	41.2	9	62	9	38	38.7
8-lat-iir	17	96	17	71	26.0	17	96	17	73	24.0	17	102	17	57	44.1	17	103	17	47	54.4
DEQ	5	40	5	31	22.5	5	30	5	27	10.0	5	35	5	27	22.9	5	35	5	27	22.9
Elliptic	15	143	15	101	29.4	14	142	14	93	34.5	14	142	14	85	40.1	14	142	14	85	40.1
Voltera	12	60	12	44	26.7	12	57	12	41	28.1	12	56	12	37	33.9	12	57	12	34	40.4
Average Reduction %					30.1	-				25.7	-				38.5	-				38.3

TABLE II

THE COMPARISON ON SWITCHING ACTIVITY AND SCHEDULE LENGTH FOR MLMSA AND LIST SCHEDULING WHEN THE NUMBER OF FUS VARIES FROM 3 TO 6.

in switching activity compared with list scheduling. The average reduction is shown on the last row. Similarly, the experimental results for MLMSA and list scheduling are shown in Table II.

The experimental results in Table III and Table IV show the comparison on energy for EMSA and list scheduling when  $P_{base} = 1$  and  $P_{base} = 2$ , respectively. Row ‘‘E.’’ lists the energy for various benchmarks. Row ‘‘SL’’ lists the schedule length obtained by list scheduling. Row ‘‘TC’’ lists the timing constraint used in EMSA. Each benchmark is scheduled with three timing constraints. Among them, the first one is the schedule length obtained by list scheduling. Row ‘‘%’’ in field ‘‘EMSA’’ lists the reduction on energy comparing EMSA with list scheduling (field ‘‘ListS’’). In each table, the results are shown when the number of FUs equals 3,4,and 5, respectively.

Through the experimental results from Table I-IV, we found our algorithms reduce energy consumption significantly compared with list scheduling. When switching activity plays the most important role in energy consumption, MSAS gives a reduction of 41.2% in switching activity on average compared with list scheduling. When schedule length plays the most important role in energy consumption, MLMSA gives the same schedule length as list scheduling and achieves a reduction of 33.2 % in switching activity on average at the same time. For general cases, EMSA gives a reduction of 31.2% in energy on average compared with

list scheduling.

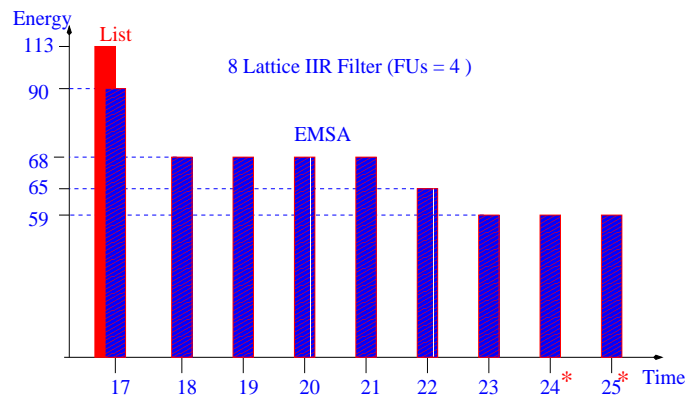


Fig. 7. The energy variation of 8-lattice IIR filter with the increase of the timing constraint when  $P_{base} = 1$ .

From the experimental results in Table III and Table IV, we can see the energy based on EMSA is reduced correspondingly with the increase of the timing constraint for each benchmark. To show the further energy variation, we show the results for 8-lattice IIR filter when the timing constraint varies from 17 to 25 in Figure 7. EMSA gradually reduces the energy when the timing constraint increases from 17 to 23. After that, the energy can not be reduced anymore. This shows that we can not gain

		2-Cas-Biq			4-Lat-IIR			8-Lat-IIR			DEQ			Elliptic			Voltera		
FU <sub>s</sub> = 3																			
ListS	SL	7			9			17			5			15			12		
	E.	61			73			113			45			158			72		
EMSA	TC	7	8	9	9	11	14	17	20	24	5	6	7	15	18	21	12	14	17
	E.	35	30	28	55	49	41	88	75	66	36	27	26	116	96	89	56	55	47
	%	<b>42.6</b>	<b>50.8</b>	<b>54.1</b>	<b>24.7</b>	<b>32.9</b>	<b>43.8</b>	<b>22.1</b>	<b>33.6</b>	<b>41.6</b>	<b>20.0</b>	<b>40.0</b>	<b>42.2</b>	<b>26.6</b>	<b>39.2</b>	<b>43.7</b>	<b>22.2</b>	<b>23.6</b>	<b>34.7</b>
FU <sub>s</sub> = 4																			
ListS	SL	6			9			17			5			14			12		
	E.	46			78			113			35			156			69		
EMSA	TC	6	7	8	9	11	14	17	20	24	5	6	7	14	18	21	12	14	17
	E.	35	31	29	57	46	39	90	68	59	32	28	28	107	79	74	53	43	43
	%	<b>23.9</b>	<b>32.6</b>	<b>37.0</b>	<b>26.9</b>	<b>41.0</b>	<b>50.0</b>	<b>20.4</b>	<b>39.8</b>	<b>47.8</b>	<b>8.6</b>	<b>20.0</b>	<b>20.0</b>	<b>31.4</b>	<b>49.4</b>	<b>52.6</b>	<b>23.2</b>	<b>37.7</b>	<b>37.7</b>
FU <sub>s</sub> = 5																			
ListS	SL	6			9			17			5			14			12		
	E.	51			77			119			40			156			68		
EMSA	TC	6	7	8	9	11	14	17	20	24	5	6	7	14	18	21	12	14	17
	E.	29	29	29	49	43	43	74	63	58	32	27	27	99	82	82	49	45	44
	%	<b>43.1</b>	<b>43.1</b>	<b>43.1</b>	<b>36.4</b>	<b>44.2</b>	<b>44.2</b>	<b>37.8</b>	<b>47.1</b>	<b>51.3</b>	<b>20.0</b>	<b>32.5</b>	<b>32.5</b>	<b>36.5</b>	<b>47.4</b>	<b>47.4</b>	<b>27.9</b>	<b>33.8</b>	<b>35.3</b>

TABLE III

THE COMPARISON ON ENERGY FOR EMSA AND LIST SCHEDULING ( $P_{base} = 1$ ) WHEN THE NUMBER OF FUS VARIES FROM 3 TO 5.

		2-Cas-Biq			4-Lat-IIR			8-Lat-IIR			DEQ			Elliptic			Voltera		
FU <sub>s</sub> = 3																			
ListS	SL	7			9			17			5			15			12		
	E.	68			82			130			50			173			84		
EMSA	TC	7	8	9	9	11	14	17	20	24	5	6	7	15	18	21	12	14	17
	E.	42	38	37	64	59	54	105	94	89	41	33	32	131	114	108	68	68	61
	%	<b>38.2</b>	<b>44.1</b>	<b>45.6</b>	<b>22.0</b>	<b>28.0</b>	<b>34.1</b>	<b>19.2</b>	<b>27.7</b>	<b>31.5</b>	<b>18.0</b>	<b>34.0</b>	<b>36.0</b>	<b>24.3</b>	<b>34.1</b>	<b>37.6</b>	<b>19.0</b>	<b>19.0</b>	<b>27.4</b>
FU <sub>s</sub> = 4																			
ListS	SL	6			9			17			5			14			12		
	E.	52			87			130			40			170			81		
EMSA	TC	6	7	8	9	11	14	17	20	24	5	6	7	14	18	21	12	14	17
	E.	41	38	37	66	57	51	107	90	82	37	34	34	121	95	92	65	57	57
	%	<b>21.2</b>	<b>26.9</b>	<b>28.8</b>	<b>24.1</b>	<b>34.5</b>	<b>41.4</b>	<b>17.7</b>	<b>30.8</b>	<b>36.9</b>	<b>7.5</b>	<b>15.0</b>	<b>15.0</b>	<b>28.8</b>	<b>44.1</b>	<b>45.9</b>	<b>19.8</b>	<b>29.6</b>	<b>29.6</b>
FU <sub>s</sub> = 5																			
ListS	SL	6			9			17			5			14			12		
	E.	57			86			136			45			170			80		
EMSA	TC	6	7	8	9	11	14	17	20	24	5	6	7	14	18	21	12	14	17
	E.	35	35	35	58	53	53	91	81	79	37	33	33	113	98	98	61	59	59
	%	<b>38.6</b>	<b>38.6</b>	<b>38.6</b>	<b>32.6</b>	<b>38.4</b>	<b>38.4</b>	<b>33.1</b>	<b>40.4</b>	<b>41.9</b>	<b>17.8</b>	<b>26.7</b>	<b>26.7</b>	<b>33.5</b>	<b>42.4</b>	<b>42.4</b>	<b>23.8</b>	<b>26.3</b>	<b>26.3</b>

TABLE IV

THE COMPARISON ON ENERGY FOR EMSA AND LIST SCHEDULING ( $P_{base} = 2$ ) WHEN THE NUMBER OF FUS VARIES FROM 3 TO 5.

further energy reduction when the timing constraint reaches a certain point. At this point, EMAS gives a reduction of 47.8% compared with list scheduling when the timing constraint equals 23.

## VI. CONCLUSION

In this paper, we presented instruction level scheduling techniques to minimize the energy consumption of applications executed on VLIW processors. We studied the two most important factors, switching activity and schedule length, that influence the energy consumption of an application on a VLIW processor during scheduling. We proved that *instruction-level energy-minimization scheduling problem* is NP-complete with or without resource constraints.

We proposed three heuristic algorithms, MSAS, MLMSA, and EMSA, to solve the problem. While switching activity and schedule length are given higher priority in MSAS and MLMSA respectively, EMSA gives the best result considering both of them. The experimental results show our algorithms can significantly reduce energy consumption compared with the standard list scheduling.

## REFERENCES

- [1] V. Tiwari, S. Malik, and M. Fujita, "Power analysis of embedded software: A first step towards software power minimization," in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, Nov. 1994, pp. 110–115.
- [2] N. Chang, K. Kim, and H. G. Lee, "Cycle-accurate energy measurement and characterization with a case study of the ARM7TDMI,"

- IEEE Trans. on VLSI Systems*, vol. 10, no. 2, pp. 146–154, Apr. 2002.
- [3] J. Chang and M. Pedram, “Register allocation and binding for low power,” in *Proceedings of the IEEE/ACM Design Automation Conference*, June 1995, pp. 29–35.
- [4] M. T.-C. Lee, V. Tiwari, S. Malik, and M. Fujita, “Power analysis and low-power scheduling techniques for embedded dsp software,” in *Proceedings of the IEEE International Symposium on System Synthesis*, Sept. 1995, pp. 110–115.
- [5] C. Lee, J.-K. Lee, and T. Hwang, “Compiler optimization on instruction scheduling for low power,” in *Proceedings of the IEEE International Symposium on System Synthesis*, Sept. 2000, pp. 55–60.
- [6] H. Yang, G. R. Gao, and C. Leung, “On achieving balanced power consumption in software pipelined loops,” in *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems*, 2002, pp. 210–217.
- [7] A. Raghunathan and N. K. Jha, “An ILP formulation for low power based on minimizing switched capacitance during data path allocation,” in *Proceedings of the IEEE International Symposium on Circuits and Systems*, May 1995, pp. 1069–1073.
- [8] L. Kruse, E. Schmidt, G. Jochens, A. Stammermann, A. Schulz, E. Macii, and W. Nebel, “Estimation of lower and upper bounds on the power consumption from scheduled data flow graphs,” *IEEE Trans. on VLSI Systems*, vol. 9, no. 1, pp. 3–14, Feb. 2001.
- [9] A. Parikh, M. Kandemir, N. Vijaykrishnan, and M. J. Irwin, “Instruction scheduling based on energy and performance constraints,” in *IEEE Computer Society Annual Workshop on VLSI*, Apr. 2000, pp. 37–42.
- [10] C.-L. Su, C.-Y. Tsui, and A. M. Despain, “Saving power in the control path of embedded processors,” *IEEE Design & Test of Computers*, vol. 11, no. 4, pp. 24–30, Winter 1994.
- [11] E. Musoll and J. Cortadella, “Scheduling and resource binding for low power,” in *Proceedings of the IEEE International Symposium on System Synthesis*, Apr. 1995, pp. 104–109.
- [12] M. Mehendale, S. Sherlekar, and G. Venkatesh, “Coefficient optimization for low power realization of fir filters,” in *IEEE Workshop on VLSI Signal Processing*, 1995, pp. 352–361.
- [13] K. Masselos, S. Theoharis, P. K. Merakos, T. Stouraitis, and C. E. Goutis, “Low power synthesis of sum-of-products computation,” in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*, July 2000, pp. 234–237.
- [14] K. Choi and A. Chatterjee, “Efficient instruction-level optimization methodology for low-power embedded systems,” in *Proceedings of the IEEE International Symposium on System Synthesis*, Oct. 2001, pp. 147–152.
- [15] J. V. D. Veen and S. Z. G. J. Woeginger, “Sequencing jobs that require common resources on a single machine: a solvable case of the tsp,” *Mathematical Programming*, pp. 235–254, 1998.
- [16] E. Macii, M. Pedram, and F. Somenzi, “High-level power modeling, estimation and optimization,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, pp. 1061–1079, November 1998.
- [17] L. Benini, D. Bruni, M. Chinosi, C. Silvano, V. Zaccaria, and R. Zafalon, “A power modeling and estimation framework for VLIW-based embedded systems,” in *Proceedings of the International Workshop-Power and Timing Modeling, Optimization and Simulation, PATMOS’01*, 2001, pp. 26–28.
- [18] A. Bona, M. Sami, D. Sciuto, C. Silvano, V. Zaccaria, and R. Zafalon, “Energy estimation and optimization of embedded VLIW processors based on instruction clustering,” in *Proceedings of the IEEE/ACM Design Automation Conference*, 2002, pp. 886–891.
- [19] M. Sami, D. Sciuto, C. Silvano, and V. Zaccaria, “Instruction level power estimation for embedded VLIW cores,” in *Proceedings of the International Workshop on Hardware/Software Codesign*, May 2000, pp. 34–38.
- [20] —, “Power exploration for embedded VLIW architecture,” in *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, Nov. 2000, pp. 498–503.
- [21] *TMS320C6000 Peripherals Reference Guide (Rev. D)*, Texas Instruments, Inc., 2001.
- [22] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [23] H. Saip and C. L. Lucchesi, “Matching algorithm for bipartite graphs,” Tech. Rep. DCC-93-03(Departamento de Cincia da Computao, Universidade Estadual de Campinas), March 1994, [http://www.dee.unicamp.br/ic\\_trftp/ALL/Abstrace.html](http://www.dee.unicamp.br/ic_trftp/ALL/Abstrace.html).
- [24] *TMS320C6000 CPU and Instruction Set Reference Guide*, Texas Instruments, Inc., 2000.