

ANALYSIS AND ALGORITHMS FOR SCHEDULING WITH MINIMAL SWITCHING ACTIVITIES

Zili Shao, Qingfeng Zhuge, Edwin H.-M. Sha *

Department of Computer Science
University of Texas at Dallas
Richardson, Texas 75083, USA
{ zxs015000, qfzhuge, edsha }@utdallas.edu

Chantana Chantrapornchai

Department of Mathematics
Silpakorn University
Nakorn Pathom, Thailand 73000
ctana@su.ac.th

ABSTRACT

Switching activities are one of the important factors in power minimization. This paper studies the scheduling problem which minimizes switching activities. We show that to find the schedule which minimizes switching activities only or minimizes latency and switching activities with or without resource constraints is NP-complete. Two heuristic scheduling algorithms to minimize switching activities are presented. The algorithms use the weighted bipartite matching to find a good schedule. The experimental results show that our algorithms can produce the near-optimal schedule which uses averagely the switching activities 35% less compared to the traditional list scheduling.

1. INTRODUCTION

Low power is becoming the critical design issue and performance metric due to wide use of the portable devices, especially those powered by batteries. To reduce the switching activities of a software application is one of most important power optimization methods when the hardware is built up. Low power scheduling can efficiently reduce switching activities such that the power consumption of an application is minimized.

Much research effort has been put in this field. Musoll et al. in [1] develop a scheduling technique such that the operation nodes sharing the same operands are scheduled as close as possible. Su et al. in [2] suggest a scheduling technique to reduce the switching activities of address access when scheduling. Kim et al. in [3] present some techniques to combine retiming with operand sharing scheduling. In these work, list scheduling is used. Since list scheduling only considers the node with highest priority in ready list, we can't get an optimal solution when the number of functional units is greater than 1 in every scheduling step.

In recent work [4], the power efficient scheduling problem is formulated as a Traveling Salesman's Problem (TSP) for one functional unit and the low power scheduling algorithm of the sum-of-products computation is solved by heuristics of TSP. However, a problem can be transferred to TSP doesn't necessarily mean it is NP-complete. For example, the problem to sequence jobs that require common resources on a single machine [5] can be transferred to TSP but still polynomially solvable. In the literature, it is still unknown that

the scheduling problem with minimizing switching activities is solvable in polynomial time or NP-complete.

To solve these problems, we generalize the scheduling problems in above work as *the scheduling problem of Data Flow Graph (DFG) with minimal switching activities*. In our work, the application program is modeled as DFG, where the nodes of DFG denote the operations while the edges denote the precedence relation. Every node of DFG has a opcode associated with the operation decided by the instruction set of functional unit. Switching activities are considered as the hamming distance between two consecutive opcodes bound to the same functional unit. Using different methods to set the opcode of the node, the different power cost can be obtained such that all problems in above work can be transferred to the scheduling problem of DFG with minimal switching activities. For example, if the opcode is set as the operand, then it is operand sharing scheduling in [1]. We consider both control step assignment and functional unit binding when scheduling in our work.

We discuss the generalized scheduling problem by dividing it further into two sub-problems: 1). the minimum-latency scheduling problem with minimal switching activities. 2). the minimum-switching-activities scheduling problem. The complexity of two problems are analyzed. We show that the minimum-latency scheduling problem with minimum switching activities is NP-complete no matter with or without resource constraints. However, it is known that the minimum-latency scheduling problem is trivially polynomial-time solvable when the number of resource equals to 1 and is polynomial-time solvable without resource constraints. Our study also shows that the minimum-switching-activities scheduling problem with or without resource constraints is NP-complete.

Two efficient heuristic algorithms, MSAM and MLMSA, are presented to solve both problems. We use the weighted bipartite matching to select the resource assignment for operations to minimize the switching activities while performing scheduling. The experimental results display the great reduction in switching activities while keeping the same schedule performance. For example, for the elliptic filter benchmark, our methods result in switching activities of 71 while the traditional list scheduling results in 113. This is 37.2% reduction while the resulting schedule length is the same. On average, our algorithms give an improvement of 35% over the list scheduling.

In the next section, we introduce necessary backgrounds and Section 3 presents the complexity analysis of our scheduling problems. The MSAM and MLMSA are discussed in Section 4. Experimental results and concluding remarks are provided in Section 5 and 6 respectively.

This work is partially supported by TI University Program, NSF EIA-0103709 and Texas ARP 009741-0028-2001, USA.

This work is partially supported by NECTEC, NT-B-06-4D-16-509, Thailand.

2. BASIC CONCEPTS AND MODELS

In this section, we introduce some basic concepts, which will be used in complexity analysis and algorithm design in Section 3 and Section 4.

We define a *Data Flow Graph (DFG)* $G_D = \langle V, E, OP \rangle$ as a node-weighted directed acyclic 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 , $OP(u)$ represents the opcode for node $u \in V$. Nodes in V can be various operations, such as addition, subtraction, multiplication, logic operation, etc.. $OP(u)$ is a function from V to a set of opcodes, representing the opcode of node u . The opcode is determined by the instruction set of the functional unit. We assume that all functional units are identical and general-purposed. Note that the nodes with the same operation in a DFG may have the different opcodes, since opcodes may vary depending the operand type and addressing mode of the instruction. The operation time of each node in the DFG is assumed to be one time unit.

In our technique, we use the weighted bipartite matching graph to help find the binding of nodes in the DFG 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 $e(u \rightarrow v) \in E$ 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 for 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 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.

Switching activities are considered as the hamming distance between the two consecutive opcodes bound to the same functional unit in our work. The switching activities of node u bound to functional unit F_i are defined as the hamming distance between $OP(u)$ and $LAST_OP(F_i)$, where $OP(u)$ is the opcode of u and $LAST_OP(F_i)$ is the opcode of the last node executing on functional unit F_i . Note that the initial value of $LAST_OP(F_i)$ is set to all 0's. We use a function, called $Switch_Node(u, F_i)$, to compute it:

$$Switch_Node(u, F_i) = HD(OP(u), LAST_OP(F_i))$$

where $HD(x, y)$ represents the hamming distance of opcode x and y , which is the number of all non-matching bits in all the corresponding locations of x and y .

The total switching activities of a schedule of a DFG are the summation of the switching activities of all nodes bound to functional units based on that schedule. In this paper, a schedule implies both control step assignment and functional unit binding. Thus, for functional units F_1, F_2, \dots, F_m , and a schedule of $G_D = \langle V, E, OP \rangle$, the switching activities of the schedule S of G_D ,

$$Switch_Act(S) = \sum_{u \in V} Switch_Node(u, F_i),$$

where u is assigned to F_i in the schedule.

Consider an example shown in Figure 2. Figure 2(a) shows the given DFG. In the figure, $OP(u)$ is shown next to each node. There are 2 functional units: FU1 and FU2. Figure 2(b) shows a schedule of which switching activities=20. The

switching activities for FU1 are $1+3+3+3=10$ and for FU2 are $1+3+3+3=10$. Hence $Switch_Act(S_1) = 20$. For schedule S_2 in Figure 2(c), $Switch_Act(S_2) = 4$. In S_2 we can see that in step 2 in FU1, rather than selecting node 5, node 3 is assigned after node 1. By doing so, the total switching activities are reduced. This is because node 1 and node 3 have the same opcode. Hence the hamming distance is zero while the hamming distance between node 1 and node 5 is three. The example shows that different schedules can result in different switching activities, therefore affecting the power consumption.

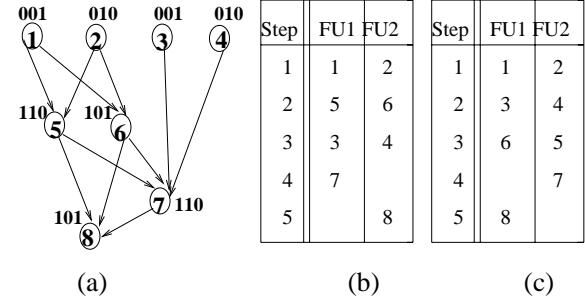


Figure 1: (a) DFG (b) S1: Switch_Act=20
(c) S2: Switch_Act=4

In this paper, the problem we intend to solve is described as follows: Given a DFG $G_D = \langle V, E, OP \rangle$,

1. Find a schedule S of G_D such that $Switch_Act(S)$ are minimum. We call the problem as min-switching-activities scheduling problem.
2. Find a schedule S of G_D whose schedule length is minimum and $Switch_Act(S)$ are minimum in all schedules with the same minimum length. We call the problem, min-latency-switching-activities scheduling problem.

Note that Problem 1 and Problem 2 are different. In Problem 1, we want to find a schedule that has the minimum switching activities in all possible schedules while in Problem 2 we only want to find a schedule that has the minimum scheduling activities in all possible minimum-latency schedules.

3. COMPLEXITY ANALYSIS

In this section, we analyze the complexity of the scheduling problem with switching activities minimization. We first discuss the min-latency-switching-activities scheduling problem and then min-switching-activities scheduling problem as defined in Section 2.

The min-latency-switching-activities scheduling problem can be categorized into three cases(Theorem 3.1, Theorem 3.2, Theorem 3.3).

When the number of resources is greater than one but not infinite, it is known that the minimum-latency scheduling is NP-complete, so the min-latency-switching-activities scheduling problem is also NP-complete.

Theorem 3.1. *Let M be the number of resources, where $M > 1$ and $M < \infty$, min-latency-switching-activities scheduling problem is NP-complete.*

Proof. When $M > 1$ and $M < \infty$, the minimum latency scheduling problem is NP-complete [6]. Assigning all nodes in the schedule with the same opcode, it can be reduced to the min-latency-switching-activities scheduling problem. \square

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

Theorem 3.2. *Let M be the number of resources, when $M = 1$, min-latency-switching-activities scheduling problem is NP-complete.*

Proof. Geometric Traveling Salesman Problem can be reduced to this problem by transforming the coordinates of a city to the opcode of the corresponding node. \square

When the number of resources is infinite, the minimum latency scheduling problem is polynomial-time solvable. The As-Soon-As-Possible scheduling can be used to find an optimal solution. However, when switching activities are considered as a second constraint, the problem becomes NP-complete.

Theorem 3.3. *Let M be the number of resources, when $M = \infty$, min-latency-switching-activities scheduling problem is NP-complete.*

Proof. Geometric Traveling Salesman Problem can be reduced to this problem by transforming the coordinates of a city to the opcode of the corresponding node and constructing a graph such that all nodes are scheduled to one functional unit. \square

The min-switching-activities scheduling problem only minimizes the switching activities. However, it is still NP-complete.

Theorem 3.4. *Let M be any positive integer representing the number of resources, min-switching-activities scheduling problem is NP-complete.*

As we see that the min-switching-activities scheduling problem and min-latency-switching-activities scheduling problem are NP-complete. In the next section, we propose efficient algorithms to solve these problems.

4. SCHEDULING WITH SWITCHING ACTIVITIES MINIMIZATION

From Section 3, we know min-switching-activities scheduling problem and min-latency-switching-activities scheduling problem are NP-complete. In this section, we propose two efficient heuristics to solve them.

Because of the dependency in a DFG, we can only schedule a node when all its parent nodes have been scheduled. Thus, the scheduling problem with switching activities minimization is how to find a matching between functional units and ready nodes such that the 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. It was shown that the problem can be solved by the Hungarian Method in $O(n^3)$ or ILP(Integer Linear Programming) in polynomial time [7]. In our technique, we construct a weighted bipartite graph based on the functional units and a set of ready nodes in every scheduling step. Then the solution based on the minimum weight maximum bipartite matching is used in the scheduling. Based on this strategy, two heuristic scheduling algorithms are designed in the following.

4.1. Minimizing Switching Activities

The first algorithm, MSAS (Minimum Switching Activities Scheduling) is designed to solve the min-switching-activities scheduling problem. It is shown in Algorithm 4.1.

Algorithm 4.1 MSAS (Min-Switching-Activities Scheduling)

Input: DFG $G_D = \langle V, E, OP \rangle$, functional unit set FU_SET
Output: A schedule with switching activities minimization
 $L_{RD} \leftarrow$ All ready nodes in G_D ;
while $L_{RD} \neq \phi$ **do**
 Construct $G_{BM} = \langle V_{BM}, E, W \rangle$;
 where $V_{BM} = FU_SET \cup L_{RD}$
 $E = \{ \langle v, F_i \rangle \mid v \in FU_SET, v \in L_{RD} \}$ and
 $W(u, F_i) = Switch_Node(u, F_i)$
 $Min_Cost_Bipartite(G_{BM})$;
 for all $e(F_i, u) \in$ min-weight max. bipartite matching **do**
 Schedule Node u to functional unit F_i
 $LAST_OP(F_i) \leftarrow OP(u)$;
 Remove u from L_{RD} ;
 Check all adjacent nodes of u and put new ready nodes into L_{RD} ;
 end for
end while

In this algorithm, we create a weighted bipartite graph in which the weight $W(e)$, where $e : u \rightarrow F_i$, is defined as $Switch_Node(u, F_i)$, as in Section 2. After constructing G_{BM} , we call function $Min_Cost_Bipartite$ by inputting G_{BM} . The function performs a minimum weight maximum bipartite matching. This matching results in a one-to-one mapping from FUs to nodes in L_{RD} which has the minimum weight among all possible matching. Because we assign the switch activities as the weight of edge, the minimum weight means the minimum switch activities. Thus, a schedule based on this matching minimizes the switch activities in every scheduling step. The loop is repeated until all nodes are scheduled.

The complexity of MSAS algorithm depends on the number of FUs and the number of nodes in a DFG. Let n be the number of nodes in G_D and m is the number of FUs. In every scheduling step, assuming k is the number of nodes in L_{RD} , we need $O((k+m)^3)$ to find minimum weight maximum bipartite matching using Hungarian Method. Thus, the complexity of MSAS is $O(n * (n+m)^3)$.

4.2. Minimizing Latency and Switching Activities

In MSAS algorithm, we do not consider optimizing schedule length. Thus, its schedule length may be longer than that of the original min-latency scheduling algorithm. In the following, MLMSA (Minimum Latency Scheduling Algorithm with Minimum Switching Activities) is presented and used to solve the min-latency-switching-activities scheduling problem, in which minimizing schedule length is the first priority and minimizing switching activities the second.

To reduce the schedule length, we define a priority function for node u of DFG, $Depth(u)$, which is the longest path from u to a leaf node in DFG.

MLMSA is similar to MSAS except that we use the different method to assign weights to the edges of weighted bipartite graph. For each edge $e(F_i \rightarrow u) \in G_{BM}$, its weight is $W(u, F_i) = Switch_Node(u, F_i) - Depth(u) * Len(OP)$, where $Depth(u)$ is the priority value of node u as mentioned and $Len(OP)$ is the length of opcode of node u .

Using this weighted method, the priority of the node is dominant in the weights of edge. Thus, the high-priority nodes in list L_{RD} will be scheduled as soon as possible. However, for the nodes with same priority, switching activities minimization is considered. Thus, we minimize the schedule length considering switching activities. In other words, MLMSA produces the same schedule as ASAP Scheduling Algorithm with Switching Activities Minimization does when the number of FUs is big enough.

Theorem 4.1. *MLMSA algorithm without resource constraints produces the same schedule as ASAP scheduling with switching activities without resource constraints does.*

5. EXPERIMENTS

We have experimented our algorithms against a set of well-known benchmarks including 4-stage lattice filter, 8-stage lattice filter, differential equation, elliptic filter and voltera filter. We compare our results with the traditional list scheduling algorithm. In our experiments, the opcodes for each node is obtained from TI TMS320C6000 Instruction Set.

Bench.	ListS		MSAS			MLMSA		
	SA	SL	SA	SL	%	SA	SL	%
4-Lattice	79	9	37	11	53	56	9	29
8-Lattice	111	17	58	21	48	70	17	37
DEQ	40	5	26	6	54	36	5	10
Elliptic	113	14	71	14	37	71	14	37
Voltera	59	12	48	15	19	45	12	24

Table 1: Comparison of switching activities and the schedule length when FUs=4.

Table 1 shows the experimental results when the number of functional units is 4. Column “SA” presents the total switching activities obtained from each scheduling approach: list scheduling (ListS), MSAS, MLMSA. Column “SL” shows the schedule length for each algorithm. Column “%” under MSAS and MLMSA present the percentage of improvement on the switching activities compared to the list scheduling approach. The average improvement for MSAS is 42.08% and for MLMSA is 27.38%. While MSAS obtains this improvement with the cost of the extra scheduling steps, MLMSA achieves it without increasing the schedule length.

	FU=1		FU=2		FU=3		FU=4	
	SA	SL	SA	SL	SA	SL	SA	SL
MSAS	58	42	58	27	56	23	58	21
MLMSA	122	42	90	21	78	17	70	17
ListS	166	42	116	21	109	17	111	17

Table 2: Switching activities and schedule length for MSAS, MLMSA and List Scheduling of 8-Lattice for FUs=1 to 4.

	FU=5		FU=6		FU=7	
	SA	SL	SA	SL	SA	SL
MSAS	56	20	55	19	65	19
MLMSA	62	17	54	17	54	17
ListS	109	17	108	17	103	17

Table 3: Switching activities and schedule length for MSAS, MLMSA and List Scheduling of 8-Lattice for FUs=5 to 7.

To show the influence of the number of FUs, Tables 2–3 show the switching activities, schedule length for 8-stage Lattice Filter for different approaches. The number of functional units varies from 1 to 8. From Tables 2–3, MSAS has greater reduction of switching activities than MLMSA when the number of functional units is small. With the increasing of number of FUs, the switching activities of MLMSA are close to that of MSAS while having the shorter schedule length. Thus, MSAS is more suitable to the system with small number of functional units and MLMSA performs well when many functional units are available. Compared with list scheduling, the switching activities are reduced by MSAS and MLMSA in all conditions. Thus, our algorithms are also efficient with increasing of the number of FUs.

6. CONCLUSION

In this paper, the scheduling problem with switching activities minimization is addressed and its complexity is analyzed. We show that min-switching-activities scheduling problem and min-latency-switching-activities scheduling problem are NP-complete. We proposed polynomial-time scheduling algorithms, MSAS and MLMSA to find the schedule which optimizes switching activities. In our approach, a weighted bipartite graph is constructed to find a minimum weight maximum bipartite matching between functional units and ready nodes in a DFG in every scheduling step. This matching is used to find the best functional unit assignment for nodes. Using the different weight assignment approaches, MSAS minimizes switching activities only while MLMSA minimizes schedule length and switching activities. We experiment our algorithms against the well-known benchmarks. The experimental results show scheduling algorithms can greatly reduce switching activities compared with the traditional list scheduling algorithm.

7. REFERENCES

- [1] 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.
- [2] 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.
- [3] D. Kim, D. Shin, and K. Choi, “Low power pipelining of linear systems: a common operand centric approach,” in *Proceedings of the IEEE/ACM International Symposium on Low Power Electronics and Design*, August 2001, pp. 225–230.
- [4] 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.
- [5] J.A.A. V. D. Veen and S. Zhang 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.
- [6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.
- [7] H.A.B. Saip and C. L. Lucchesi, “Matching algorithm for bipartite graphs,” Tech. Rep., March 1994, http://www.dee.unicamp.br/ic_trftp/ALL/Abstrace.html.