

LOOP SCHEDULING FOR MINIMIZING SCHEDULE LENGTH AND 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 activity is one of the most important factors in power dissipation. This paper studies the scheduling problem that minimizes both schedule length and switching activities for applications with loops. We show that to find a schedule that has the minimal switching activities among all minimum-latency schedules with or without resource constraints is NP-complete. Although the minimum latency scheduling problem is polynomial-time solvable if there is no resource constraints or only one function unit, the problem becomes NP-complete when considering switching activities as the second constraint. An algorithm, Power Reduction Rotation Scheduling (PRRS), is proposed. The algorithm attempts to minimize both switching activities and schedule length while performing scheduling and allocation simultaneously. The experimental results show that our algorithm produces effective schedules that give the switching activities 58.5% less compared to the traditional rotation scheduling on average. Our algorithm also shows better performance than the approach that considers scheduling and allocation at the separate phases.

1. INTRODUCTION

In many portable systems, such as wireless communication and image processing, the DSP processor core consumes a significant amount of power and time due to the 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. Since switching activities play a key role in the total power consumption [1], our low power scheduling algorithm attempts to construct a schedule in such a way that the total switching activities are minimized.

Different low power scheduling and allocation techniques have been studied. The low power resource allocation approach is proposed in [7]. This approach attempts to find an allocation for a fixed schedule such that the total switching activities are reduced. However, the approach can give inferior results since scheduling and allocation are performed separately. Based on operand sharing approach, a loop pipelining methodology to reduce both latency and power is first proposed in [12]. Using similar approach, a loop pipelining technique is proposed to first minimize power and then maximize throughput in [6]. However, in the literature, it

is still unknown that the loop scheduling for minimizing switching activities is polynomial-time solvable or NP-complete.

In our work, the complexity of the low power loop scheduling problem is analyzed. We formally prove that the loop scheduling problem with minimum latency and minimum switching activities is NP-complete with or without resource constraints. While the minimum latency loop scheduling problem is polynomial-time solvable if there is only one functional unit (FU) or no resource constraints, the problem becomes NP-complete when considering switching activities as the second constraint.

We design an algorithm, Power Reduction Rotation Scheduling (PRRS), to minimize both switching activities and schedule length for loop applications. Our approach performs scheduling and allocation simultaneously to produce a good schedule. The experimental results show significant reduction in switching activities while keeping the same schedule performance. For example, for 8 lattice filter benchmark, PRRS results in switching activities of 20 while the traditional rotation scheduling results in 106 when there are 10 FUs. This is a reduction of 81.8% while the resulting schedule length is the same. On average, our algorithm gives a reduction of 58.5% over the traditional rotation scheduling for various benchmarks. The experimental results also show PRRS has better performance in switching activities reduction than the algorithm based on the approach that considers low power allocation with a fixed schedule [7].

In the next section, we introduce necessary background. Section 3 presents complexity analysis of our scheduling problem. 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

In this section, we introduce some basic concepts which will be used in the later sections.

A *Data Flow Graph (DFG)* $G = \langle V, E, OP, d \rangle$ 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 , $OP(u)$ represents the opcode for each node $u \in V$, $d(e)$ represents the number of delays for an edge e . 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. The computation time of each node is assumed to be 1 time unit. 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

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.

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. 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 in Figure 1(b) respectively .

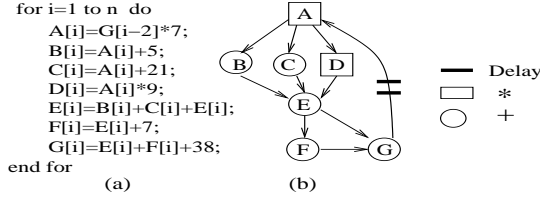


Figure 1: A loop and its corresponding DFG.

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. For example, Figure 2 shows a static schedule for the DFG in Figure 1(b) when there are three FUs. The schedule is obtained by list scheduling. In the schedule, the binary string in the parenthesis beside each node denotes the opcodes of nodes. We assume that the opcodes of multiplication and addition operation are 001 and 110, respectively. We use $[i, j]$ to denote the location of a node in a schedule, where i is the row (which control step) and j is the column (which FU). For example, location $[2, 1]$ in the schedule refers to node B scheduled at control step 2 and assigned to FU₁ in Figure 2.

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

Figure 2: The static schedule for the DFG in Figure 1(b).

Retiming [8] can be used to optimize the cycle period of a DFG by evenly distributing the delays in it. Given a DFG $G = \langle V, E, OP, d \rangle$, 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 incoming edges of node u and pushed to all of the outgoing edges. Let $G_r = \langle V, E, OP, d_r \rangle$ 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 presented in [2] 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 first and second rotation are shown in Figure 3(a) and Figure 3(b) respectively, which is based on the original schedule in Figure 2. The minimal schedule length is obtained by the schedule in Figure 3(b).

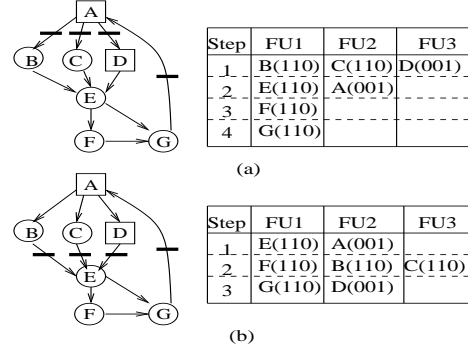


Figure 3: (a) The retimed graph and the schedule after first rotation. (b) The retimed graph and the schedule after second rotation.

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 , called $Switch_Node(u, FU_i)$, is defined as the hamming distance between $LAST_OP(FU_i)$ and $OP(u)$, where $OP(u)$ is the opcode of u and $LAST_OP(FU_i)$ is the opcode 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 executed for the loop, the initial value of $LAST_OP(FU_i)$ is set as the opcode of the last node executed on FU_i in the previous iteration. For example, for the static schedule shown in Figure 2, the initial value of $LAST_OP(FU_1)$ is 110(the opcode of node G) and the initial value of $LAST_OP(FU_2)$ is 110(the opcode of node C) and the initial value of $LAST_OP(FU_3)$ is 001 (the opcode of node D). For a static schedule S , $Switch_Act(S)$ is used to denote its switching activity, where:

$$Switch_Act(S) = \sum_{FU_i} \sum_{u \text{ assigned to } FU_i} Switch_Node(u, FU_i).$$

For example, $Switch_Act(S)=6$ for the static schedule S in Figure 2, 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 3(a) and Figure 3(b).

The problem we intend to solve is defined as follows. Given a cyclic DFG $G = \langle V, E, OP, d \rangle$ 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. COMPLEXITY ANALYSIS

In previous work such as [3,9],the power efficient scheduling problem is formulated as the Traveling Salesman Problem (TSP) and solved by heuristics of TSP when there is one FU. However, a problem can be transformed 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 [11] can be transformed to TSP but still is polynomial-time solvable. In this section, we formally prove that the min-latency-switching-activity scheduling problem is NP-complete with or without the resource constraints. Note that the minimum latency loop scheduling problem is polynomial-time solvable if there is only one FU or no resource constraints. However, we show that it becomes NP-complete when considering switching activities as the second constraint. We categorize the problem into three cases as shown in Theorem 3.1, Theorem 3.2 and Theorem 3.3.

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

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

Proof. When $U > 1$ and $U < \infty$, the minimum latency scheduling problem is NP-complete [5]. By assigning all nodes with the same opcode in an instance of the minimum latency scheduling problem, we can get an instance of our 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 U be the number of resources, when $U = 1$, the min-latency-switching-activity scheduling problem is NP-complete.*

Proof. An instance of the L_1 Geometric Traveling Salesman Problem [4] can be reduced to an instance of our 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. Retiming [8] can be used to find an optimal solution. However, when switching activities are considered as the second constraint, the problem becomes NP-complete.

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

Proof. An instance of the L_1 Geometric Traveling Salesman Problem can be reduced to an instance of our problem by transforming the coordinates of a city to the opcode of the corresponding node and constructing a cyclic DFG such that all nodes corresponding to the cities are scheduled to one FU. \square

4. THE PRRS ALGORITHM

In this section, an algorithm, Power Reduction Rotation Scheduling (PRRS), is designed to solve the min-latency-switching-activity scheduling problem based on rotation scheduling. It is shown in Algorithm 4.1.

In this algorithm, we first put all nodes in the first row of S into set R . Then we delete the first row of S and shift S up by one control step. Variable L is used to record the schedule length of S . After that, we retime each node $u \in R$ such that $r(u) \leftarrow r(u) + 1$. Then based on the precedence relation in the retimed graph G_r , we rotate each node $u \in R$ by putting u into the location with the minimum switching activities among all available empty locations in T , where T is the set containing all available locations of u . For a location $[i, j] \in T$, we define a function, $\text{Switch_Location}(u, [i, j])$, to compute the switching activities if u is assigned to location $[i, j]$. Assume that u' is the node in the first non-empty location above $[i, j]$ and u'' is the node in the first non-empty location below $[i, j]$ both in column j of S , then $\text{Switch_Location}(u, [i, j]) = \text{HD}(\text{OP}(u'), \text{OP}(u)) + \text{HD}(\text{OP}(u), \text{OP}(u'')) - \text{HD}(\text{OP}(u'), \text{OP}(u''))$, where $\text{HD}(x, y)$ represents the hamming distance of opcode x and y . When computing T , the available locations from row 1 to row L are considered first. If there is no available locations in this field, we assign

Algorithm 4.1 Power-Reduction-Rotation-Scheduling (PRRS)

Input: DFG $G = \langle V, E, \text{OP}, d \rangle$, the retiming r of G , an initial schedule S of G , the rotation times N

Output: A schedule S and the retiming r

for all $k=1$ to N **do**

$R \leftarrow$ All nodes in the first row in S ;

Delete the first row from S ;

Shift S up by 1 control step;

for all $u \in R$ **do**

$r(u) \leftarrow r(u) + 1$;;

end for

for all $u \in R$ **do**

$T \leftarrow$ All available locations of u from Row 1 to Row L in S based on the precedence relation in G_r ;

if $E = \emptyset$ **then**

$T \leftarrow$ All available locations of u in Row $L + 1$ in S ;

end if

$[a, b] \leftarrow$ The location with the minimum switching activities among all locations in T ;

Put u into $[a, b]$;

end for

$S_k \leftarrow S$; $r_k \leftarrow r$;

end for

Select S_j from S_1, S_2, \dots, S_N such that S_j has the minimum switching activities among all minimum-latency schedules;

Output S_j and r_j ;

the node to the locations in row $L + 1$. Using this strategy, the schedule length is minimized in the first priority. After all nodes in R are scheduled, the schedule S and the retiming r are recorded. PRRS will repeat the above procedure N times, where N is a user specified amount. The best schedule is selected from the generated N schedules, which has the minimum switching activities among all min-latency schedules.

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)	D(001)
3	G(110)	C(110)	

(b)

Figure 4: The schedules generated by PRRS algorithm both with the switching activity of 0. (a) The schedule after first rotation. (b) The schedule after second rotation.

The schedules generated by PRRS after first and second rotation of Figure 1 is shown in Figure 4. We assume the schedule in Figure 2 is the initial schedule. Note that the switching activity is 0 for both schedules while it is 6 for both schedules in Figure 3 generated by the traditional rotation scheduling. This shows that our PRRS can significantly reduce switching activities compared to the traditional rotation scheduling.

Let M be the number of functional units and n be the number of nodes in G . Then the number of nodes in a row in a schedule is at most M and the total number of empty locations is at most $M * (n - 1)$. Considering the rotation times N , the complexity of PRRS algorithm is $O(N * M * M * (n - 1)) = O(N * M^2 * n)$.

5. EXPERIMENTS

In this section, we show the effectiveness of PRRS algorithm on a set of well-known benchmarks including 4-stage lattice filter, 8-stage lattice filter, differential equation solver, elliptic filter and

voltera filter. We compare our results with the traditional rotation algorithm and the low power allocation approach in [7]. In that approach, the schedule is fixed and the allocation is performed to reduce switching activities. We implement an algorithm, LP_Allocation, based this approach. LP_Allocation uses the schedule generated by traditional rotation scheduling and performs the allocation by bipartite matching. In our experiments, the opcode for each node is obtained from TI TMS320C6000 Instruction Set [10].

Table 1 shows the experimental results when the number of FUs is 5. Column "SA" presents the switching activity of the static schedule and Column "SL" presents the schedule length obtained from four different scheduling algorithms: the list scheduling (Field "List"), the traditional rotation scheduling (Field "Rotation"), the LP_Allocation (Field "LP_Alloc"), and our PRRS algorithm (Field "PRRS"). Column "%" under "PRRS" presents the percentage of reduction on switching activities compared to the traditional rotation algorithm. Our PRRS algorithm reduces switching activities by 58.5% on average.

Bench.	List		Rotation		LP_Alloc		PRRS		
	SA	SL	SA	SL	SA	SL	SA	SL	%
4-Lattice	74	9	80	6	58	6	32	6	60.0%
8-Lattice	106	17	112	9	82	9	68	9	39.3%
DEQ	30	5	36	4	16	4	10	4	72.2%
Elliptic	136	14	136	14	68	14	58	14	57.4%
Voltera	72	12	72	12	40	12	26	12	63.9%
Average Reduction (%)									58.5%

Table 1: Comparison of switching activities and schedule length when the number of FUs equals to 5.

To demonstrate the influence of the number of FUs, Table 2 show the switching activity and schedule length for 8-stage Lattice filter for different scheduling algorithms when the number of FUs varies from 3 to 12. The experimental results show that when we increase the number of FUs, the percentage of reduction on switching activities increases correspondingly.

FUs	List		Rotation		LP_Alloc		PRRS		
	SA	SL	SA	SL	SA	SL	SA	SL	%
3	106	17	118	14	90	14	86	14	27.1%
4	108	17	118	11	94	11	68	11	42.4%
5	106	17	112	9	82	9	68	9	39.3%
6	104	17	116	7	76	7	68	7	41.4%
7	96	17	120	6	58	6	58	6	51.7%
8	110	17	120	6	58	6	30	6	75.0%
9	110	17	120	5	84	5	38	5	68.3%
10	114	17	110	5	66	5	20	5	81.8%
11	112	17	120	4	44	4	30	4	75.0%
12	102	17	106	4	76	4	26	4	75.5%
Average Reduction (%)									57.7%

Table 2: Comparison of switching activities and schedule length for 8 lattice filter when the number of FUs varies from 3 to 12.

Through the experimental results from Tables 1- 2, we found that the list scheduling shows the inferior performance in both schedule length and switching activities. The traditional rotation scheduling can effectively reduce schedule length but not switching activities. The LP_Allocation algorithm can only reduce switching activities for a fixed schedule. Our PRRS can reduce both of them. Furthermore, it yields greater reduction on switching activities than the LP_Allocation algorithm especially when the number of FUs is large.

6. CONCLUSION

This paper studies the scheduling problem which attempts to minimize both the schedule length and the power consumption for ap-

plications with loops. We show 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, Power Reduction Rotation Scheduling, is proposed. The algorithm attempts to minimize 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 produces effective schedule with the great reduction in switching activities and the same schedule performance compared to the existing approaches.

7. REFERENCES

- [1] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. M. Rabaey, and R. W. Brodersen. Optimizing power using transformation. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 14(1):12–31, Jan. 1995.
- [2] L.-F. Chao, A. S. LaPaugh, and E. H.-M. Sha. Rotation scheduling: A loop pipelining algorithm. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 16(3):229–239, March 1997.
- [3] K. Choi and A. Chatterjee. Efficient instruction-level optimization methodology for low-power embedded systems. In *Proc. of the IEEE International Symposium on System Synthesis*, pages 147–152, Oct. 2001.
- [4] M. R. Garey and D. S. Johnson. Some NP-complete geometric problems. In *Proc. of the ACM Symposium on Theory of Computing*, pages 10–22, May 1976.
- [5] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.
- [6] D. Kim, D. Shin, and K. Choi. Low power of linear systems: A common operand centric approach. In *Proc. of the IEEE/ACM International Symposium on Low Power Design*, pages 225–230, August 2001.
- [7] 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 schedule data flow graphs. *IEEE Trans. on VLSI Systems*, 9(1):3–14, Feb. 2001.
- [8] C. E. Leiserson and J. B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.
- [9] K. Masselos, S. Theoharis, P. K. Merakos, T. Stouraitis, and C. E. Goutis. Low power synthesis of sum-of-products computation. In *Proc. of the IEEE/ACM International Symposium on Low Power Electronics and Design*, pages 234–237, July 2000.
- [10] Texas Instruments, Inc. *TMS320C6000 CPU and Instruction Set Reference Guide*, 2000.
- [11] 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*, pages 235–254, 1998.
- [12] T. Z. Yu, F. Chen, and E. H.-M. Sha. Loop scheduling algorithms for power reduction. In *Proc. of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pages 3073–3076, May 1998.