Loop Scheduling Algorithms for Power Reduction*

Ted Zhihong Yu, Fei Chen and Edwin H.-M. Sha
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN 46556

ABSTRACT

The increasing demand for portable computing has elevated power consumption to be one of the most critical parameters for execution of loops which constitute most of the computation of scientific applications. The reduction of a schedule length is usually considered to be opposite to the reduction of power. This paper presents a novel loop pipelining approach to reduce power consumption while reducing the schedule length. Power consumption is measured by transition activity between operands of successive operations. Both initial scheduling and loop scheduling across iterations try to reduce the transition activity at the inputs to the functional units. A series of experiments show that our method achieves considerable power dissipation and schedule length reduction.

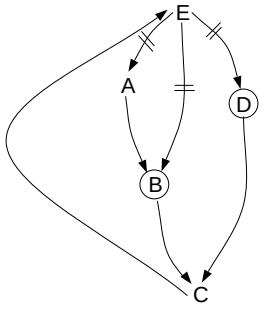

Figure 1. Original Data Flow Graph

1 INTRODUCTION

The time critical sections of computation intensive applications, such as multimedia applications and DSP systems, usually consist of loops of instructions. The strict limitation on power dissipation which portability imposes, must be met by the system while still satisfying computational requirements.

This paper presents a *loop pipelining* methodology that targets at reducing power dissipation of the execution of the loop schedule. Loop pipelining explores scheduling opportunities by arranging operations across iteration boundaries. *Rotation scheduling* is a loop pipelining algorithm whose effect is characterized by retiming data flow graph (DFG) of the loop body. The rotation scheduling, however, does not consider the power dissipation. And it was not clear how we can reduce a schedule length and the power dissipation at the same time by loop pipelining. Our loop scheduling approach integrates power dissipation reduction into rotation scheduling. Since the exponential growth of scheduling space with the number of operations in the loop prohibits a complete search through the problem space, rotation scheduling exhibits both efficiency and flexibility in minimizing the transition activity of the inputs to the functional units. After a sequence of rotations, we get the final schedule that both satisfies a predefined iteration bound and reduces power dissipation.

Studies concerning low power design have proposed several different solutions. Irwin, et. al, [3] study the effect of several standard compiler techniques such as loop unrolling, software pipelining, and recursion elimination on power consumption. They show that *software pipelining* decreases the number of stalls by fetching instructions from different iterations. Hence, total energy consumption reduces due to reduction in stalls and program takes fewer cycles to finish (therefore average energy per cycle is greater). However, the studied techniques themselves don't target at power minimization.

Anand and Jha [2] proposed an allocation method in behavioral synthesis that selects a sequence of operations (variables) for a module (register) such that the transition activity is reduced. They used a composite weight associated with each edge in the *Compatibility Graph*, involving both *capacitance weight* and *transition activity weight*, to guide mapping the two variables (operations) to the same register (module).

Chandrakasan, et. al, [1] presented a high-level synthesis system, HYPER-LP, for minimizing power consumption using a variety of architectural and computational transformations. The synthesis environment consists of high-level estimation of power consumption, a library of transformation primitives, and heuristic/probabilistic optimization search mechanisms for fast and efficient scanning of the design space.

This paper deals with uniform loops represented by data flow graphs (DFGs) where nodes represent computations and edges represent data dependencies between the nodes. We use the following loop from a digital filter to illustrate the effect of rotation scheduling for power reduction.

```
1 for i = 2 to N do
2     A[i] = E[i − 2] + 9
3     B[i] = A[i] ∗ E[i − 2]
4     D[i] = E[i − 2] ∗ 30
5     C[i] = B[i] + D[i]
6     E[i] = C[i] + 5
7 od
```

The corresponding DFG is given in Figure 1. Each statement in the code is represented by the name of destination array. Circled nodes represent multiplications and the other nodes are additions.

*This work was partially supported by the NFS under grant number MIP 95-01006.

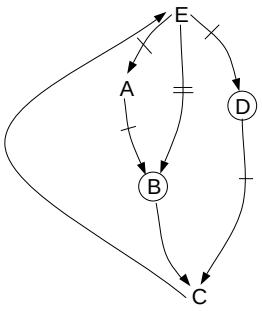

Figure 2. Data Flow Graph retimed once

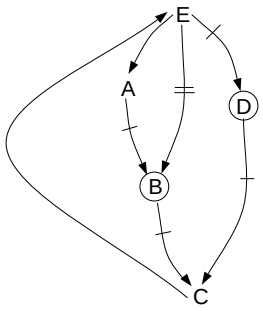

Figure 3. Data Flow Graph retimed twice

Before scheduling, a group of sample initial values are used to obtain the statistics of bit transitions between each pair of array entries that could be applied to the same input of the corresponding functional unit. This statistics is averaged over the total number of iterations run by the simulator.

We use one adder and one multiplier for the schedules in this Section. Both functional units have unit computation time for one operation. The initial schedule is shown in the table below with average transition activity of 80.324 (bits) per iteration. The subscripts on the same side of the operations denote the operands applied to the same input of the respective functional unit. For example, array entry B and constants 9 and 5 are fed to the same input of the adder.

Step	*Adder*	*Mult*
1	${}_{9}\mathrm{A}_{E}$	${}_{30}\mathrm{D}_{E}$
2		${}_{A}\mathrm{B}_{E}$
3	${}_{B}\mathrm{C}_{D}$	
4	${}_{5}\mathrm{E}_{C}$	

We then retime the graph by pushing one delay through node A and one delay through node D, obtaining a new graph in Figure 2. This retiming effectively rotates [6] nodes A and D in the initial schedule down. The new schedule is given in the second table, with average transition activity 96.7239. The retimed nodes carry apostrophes to signify that they come from the next iteration. Although the amount of transition increases, we would keep on rotating this schedule.

Step	*Adder*	*Mult*
1	${}_{E}\mathrm{A'}_{9}$	${}_{A}\mathrm{B}_{E}$
2	${}_{B}\mathrm{C}_{D}$	${}_{30}\mathrm{D'}_{E}$
3	${}_{C}\mathrm{E}_{5}$	

Nodes A' and B are rotated once more. The DFG in Figure 3 is produced and its schedule is given in the third table with transition activity 44.162. The power dissipation, as measured by transition activity, is reduced to **55%** of that of the initial schedule. It is noteworthy that the schedule length is also shortened by 1 control step.

Step	*Adder*	*Mult*
1	${}_{B}\mathrm{C}_{D}$	${}_{30}\mathrm{D'}_{E}$
2	${}_{C}\mathrm{E}_{5}$	${}_{A}\mathrm{B'}_{E}$
3	${}_{E}\mathrm{A''}_{9}$	

To the author's knowledge, this is the first paper that combines loop scheduling technique with power dissipation reduction. The proposed algorithm reduces power consumption as well as shortens schedule length. In addition, no hardware modification is required. The experiments show that power reduction is, on average, 34.17% of the original power dissipation.

The paper is organized as follows. We illustrate the fundamentals of power consumption sources, retiming technique and rotation scheduling in Section 2. Section 3 describes our approach to rotating the available schedule. A series of experiments are described in Section 4 that demonstrate the effectiveness of our method. Section 5 concludes the paper.

2 Modeling Power Dissipation

Sources of Power Dissipation

The primary sources of power dissipation in CMOS circuits are switching power, short-circuit currents and leakage currents, of which switching power is the dominant part. The *leakage current* consists of reverse bias current in the parasitic diodes formed between source and drain diffusions and the bulk region in a MOS transistor. The *short-circuit* (rush-through) current is due to the DC path between the supply rails during output transitions. The *switching power* originates from the charging and discharging of capacitive loads during logic changes.

The average switching power consumed by a CMOS gate is given by $\frac{1}{2}C_L V_{dd}^2 \frac{N}{T}$, where C_L is the gate output capacitance, V_{dd} is the supply voltage, and N is the number of gate output transitions during the period of operation T. The formula points to three avenues for reducing power dissipation, namely, reducing capacitance (C_L), reducing the supply voltage (V_{dd}), and reducing the transition activity ($\frac{N}{T}$). The quadratic dependence on supply voltage has been verified for a number of logic functions and logic styles. A side-effect of decreasing V_{dd} [1, 4], however, is that the delay of the circuit increases. The product of the physical capacitance, C_L, and the transition activity, N, is called the *switched capacitance*.

We aim at data-dominated circumstances, common in digital and image processing applications, for which control logic power dissipation is typically a minor portion of the total power consumption. Hence, our primary goal is to reduce power dissipation in data transfer which is a reflection of the data dependences in the uniform loops.

Retiming and Rotation Scheduling

The uniform loops we deal with can be represented by data flow graphs. A *data-flow graph (DFG)* is a directed node-weighted

edge-weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, d, t)$ where $\mathcal{V}$ is the set of computation nodes, $\mathcal{E}$ is the set of edges which define the precedence relations from nodes in $\mathcal{V}$ to nodes in $\mathcal{V}$, $d(e)$ is the number of delays (registers) for an edge $e \in \mathcal{E}$, and $t(v)$ is the computation time of node $v \in \mathcal{V}$. Example DFGs can be found in Figures 1, 2 and 3.

The notation $u \stackrel{e}{\rightarrow} v$ indicates that e is an edge from node u to node v. For any iteration j, an edge e from u to v with delay $d(e)$ conveys that the computation of node v at iteration j depends on the execution of node u at iteration $j - d(e)$.

A retiming r [5] of node u is a function from $\mathcal{V}$ to the integers. The value of this function is the number of delays taken from all incoming edges of u and moved to each of its outgoing edges. In this paper, delays are represented as bar lines over the graph edges. For any edge $u \stackrel{e}{\rightarrow} v$, we have $d_r(e) = d(e) + r(u) - r(v)$. Let us look at an example of retiming. After applying the retiming r of $r(A) = r(D) = 1$ on the graph in Figure 1, one delay is pushed though node A, another delay is pushed through node D. The graph 2 is obtained thereafter.

The *down-rotation* [6] of a set $X \subset V$ pushes one delay from each of the incoming edges of X into each of its outgoing edges, transforming the DFG G into G_X. As illustrated in Figures 1 and 2, the set of nodes $\{A, D\}$ are intuitively rotated down.

We say that a set X is *down-rotatable* if retiming X is a legal retiming for G. In other words, a set X is *down-rotatable* only when every edge from $V - X$ to X contains at least one delay. For example, in Figure 2, the sets $\{A\}$, $\{A, D\}$ and $\{A, B, D\}$ are down-rotatable sets, but $\{C\}$ and $\{C, E\}$ are not. The composite of two retimings is $r_1 \circ r_2(v) = r_1(v) + r_2(v)$. The composite of a sequence of down-rotations is the composite of the retimings of the down-rotation sets. In the realm of our research, we consider only *normalized* retiming functions, i.e. those which have $min_v r(v) = 0$. This means that not every node has been rotated during the scheduling process.

3 LOW POWER LOOP PIPELINING

A Measure for Power Dissipation

Consider a functional unit fu_i in an RTL circuit that performs two consecutive operations. The transition activity at the inputs of fu_i is determined by the number of bit flips between the input values. Our scheduling method selects a sequence of input values for a functional unit such that the transition activity is reduced. As the basis for our scheduling heuristic, we first gather the average transitions between every pair of array entries (or constants) that could possibly be consecutively fed to the same functional unit. We run the uniform loop several hundred times over random initial values of normal distribution for the arrays in the code.

This pairwise transition quantity, stored in a matrix called the *transition count matrix*, is categorized into two groups, one group for addition/subtraction, the other for multiplication. The matrix entries are averaged over all the iterations conducted. Each array entry appears, with different indices, the number of times equal to one plus the maximum of delay sums among all the cycles in the DFG. Take Figure 1 for example, each array entry appears 3 times as there are two delays along each cycle in the graph. The three appearances of array E are: E[i-2], E[i-1], E[i]. The submatrix corresponding to the initial schedule in Section 1 is given in the following two tables, the top for multiplication and the bottom for addition.

	E[i-2]	30
A[i]	5.479	15.707
E[i-2]	0	16.204

	E[i-2]	C[i]	5	9
B[i]	17.231	15.866	16.067	16.069
D[i]	16.212	16.269	17.238	16.74
E[i-2]	0	16.905	16.204	16.326

The actual transition count submatrix for the initial schedule is larger than shown here. It is divided into two tables for clearer view. Though the whole iteration space are considered to be pipelined, we prove the size of the transition count matrix is only within a fixed limit. The following theorem shows that considering so many iterations in the transition count matrix is adequate for the whole scheduling process.

Theorem 3.1 *The maximum number of spanning iterations to be included for the consideration of loop pipelining in the transition count matrix is:* $1 + max_{cycle\ l}\ d(l)$.

Proof: The number of delays on path $u \stackrel{p}{\rightsquigarrow} v$ represents the number of iterations computations u and v are apart. Since retiming maintains the number of delays on each cycle in the DFG constant, computation u can be retimed at most $max_{cycle\ l}\ d(l)$ times in which cycle l contains u. That is, u in iteration i is correlated to other nodes from iteration i to iteration $i + max_{cycle\ l}\ d(l)$. The above conclusion is reached considering each computation in the DFG.

Rotation for Reducing Transitions

In power reduction arena, the rotation/push-up process takes power dissipation into account. In reducing power consumption, the position for the rotated node that has least transition activity compared to other candidate empty schedule cells is chosen in the push-up phase of Rotation Scheduling.

We rotate the first row in the current schedule table each time. Each node in the first row increments its *retiming counter* which records the number of times the node has been retimed. Each submatrix in the transition count matrix has the same number of rows and columns equal to the number of different array references plus the number of constants in the original DFG. The retiming counter for a node is used as an index into the corresponding submatrix. For example, in Figure 2, A is retimed once, and A's reference to E becomes E[i-1]. Therefore, the transition count between E[i-1] and say, B[i] referenced by C[i], is stored in second submatrix in the first stripe of submatrices. Basically, the submatrix for the transition count concerned is located by the tuple of retiming counters of the two array references. The submatrix for the original DFG is given tuple (0, 0) since no reference is retimed yet. Second, the count is indexed by relative orders of the two references in the located submatrix.

Each schedule table cell contains two elements: the node allocated in the cell and its corresponding retiming counter. The counter is used in subsequent transition activity and operand ordering determination. The Rotation Scheduling method that reduces transition activity is outlined in the following algorithm:

Algorithm Power_Reduction_Rotation
Input: DFG $\mathcal{G} = (\mathcal{V}, \mathcal{E}, d, t)$
Output: schedule table, total transition activity, operand order for each node

```
1 for i = 1 to N do
2     for each u ∈ first row do
3         increment retiming counter for u
```

```
4       retime u once
5     od
6     increment the length of loop prologue
7     for each u ∈ first row do
8       deltaPwr ← infinity
9       for each empty schedule cell x do
10        currDelta ← new trans − old trans
11        if (currDelta < deltaPwr) then
12          deltaPwr ← currDelta
13          destCell ← x
14        fi
15      od
16      schedule u in schedule cell destCell
17      update total transition
18    od
19    output current schedule and total transition
20  od
```

In this algorithm, the potential schedule cell with the largest transition activity reduction for each rotated node is determined by the nested loop from line 7 to line 18. First we assign an infinite value to transition activity reduction variable $deltaPwr$. Later on, each available schedule cell x is searched, in the loop from line 9 to line 15, to see if the insertion of the rotated node reduces transition activity by an amount greater than the absolute value of current $deltaPwr$. Here, four possible combinations of operand ordering are considered against the transition activity between scheduled nodes. The currently best schedule cell is recorded in $destCell$. Once the cell with greatest transition activity reduction is found, the rotated node is scheduled there and total transition activity is updated as shown in line 17 of the algorithm.

In line 19, the resultant schedule table, after rotating one row, and its associated total transition activity are output. At this point, the specific operand order for each operation is recorded in the schedule table. This rotation procedure is repeated N times, which is a user specified amount. Because rotation would continuously seek to reduce the total transition, we use N as the count for the optimization process. The best schedule is selected among all the generated schedules.

Under this framework, we write a function, `minTrans`, that returns the smaller transition activity between one node (with retiming counter updated) and a specific schedule cell. The first argument is a node pointer rather than a schedule cell pointer because at the time of call to `minTrans`, the destination schedule cell for the rotated node is unknown. `minTrans` considers two possible operand orderings between the two nodes.

4 EXPERIMENTS

Loop	Nodes	Trans0	Trans1	%impr	Len0	Len1
Filter1[6]	5	80.324	44.162	**45**%	5	3
Median[9]	14	227.164	163.706	**28**%	12	9
WDF[7]	12	175.283	118.473	**32**%	10	7
IIR[7]	15	233.197	148.725	**36.2**%	15	11
Filter2[10]	21	337.162	226.672	**32.8**%	16	12
Filter3[8]	33	532.389	369.274	**31**%	23	17

We use the Median Filter for image processing [9], the Infinite-Extent Impulse Response filter [7], the Wave Digital Filter [7] for Transmission Line Simulation, the filter [6] from Section 1. Filter 3[8] has relatively many computations. Filter 2 comes from [10]. The Nodes column lists the total number of nodes in the graph. The final transition activity (Trans1) is compared against initial transition (Trans0), giving the percentage reduction in the %impr column. The initial schedule length (Len0) is compared with final schedule length (Len1). We use one adder of unit computation time and one multiplier of two unit computation time as the hardware configuration. The total transition activity is reduced, on average, by **34.17**%. We can see that this method is capable of dealing with various uniform loops containing relatively many computations. The achievement of rotation is actually two fold: schedule length is shortened and power consumption of the execution for the loop is reduced.

5 CONCLUSION

This paper presents a loop pipelining approach, *Rotation Scheduling*, to reduce power dissipation of the execution of uniform loops. Power dissipation is measured in terms of *transition activity* between successive computations allocated on the same functional unit. This information is recorded in *transition count matrix*. The Rotation process selects the available schedule cell that reduces transition activity most among consecutive operations. The experimental data on several digital filters show the effectiveness of this approach in both schedule length and power dissipation reduction. To the author's knowledge, this is the first paper that integrates power consumption reduction into loop pipelining.

References

[1] Chandrakasan, Mehra, Rabaey, et. al, "Optimizing Power Using Transformations", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 14, No. 1*, Jan. 1995, pp. 12-30.

[2] Anand Raghunathan and Niraj K. Jha, "Behavioral Synthesis for Low Power", *International Conference on Computer Design*, 1994

[3] H. Mehta, Robert M. Owens, M. Irwin, R. Chen and D. Ghosh, "Techniques for Low Energy Software", *International Symposium on Low Power Design*, 1997, pp. 72-75.

[4] Mark C. Johnson and Kaushik Roy, "Scheduling and Optimal Voltage Selection for Low Power Multi-Voltage DSP Datapaths", *Technical Report, School of Electrical and Computer Engineering*, 1997

[5] Charles E. Leiserson, and James B. Saxe, "Retiming Synchronous Circuitry", *Algorithmica*, 1991, 6:5-35

[6] L.-F. Chao, E. H.-M. Sha and A. LaPaugh, "Rotation Scheduling: A Loop Pipelining Algorithm", (regular paper) in IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 16, No. 3, March 1997, pp. 229-239

[7] N. Passos and E. H.-M. Sha, "Achieving Full Parallelism using Multi-Dimensional Retiming", (regular paper) in IEEE Transactions on Parallel and Distributed Systems, Vol. 7, No. 11, November, 1996, pp. 1150-1163

[8] N. Passos and E. H.-M. Sha, "Push-Up Scheduling: Optimal Polynomial-Time Resource Constrained Scheduling for Multi-Dimensional Applications", in Proc. IEEE/ACM International Conference on Computer-Aided Design, San Jose, CA, November 1995, pp. 588-591

[9] Zhizhong Tang, Bogong Su, Stanley Habib, et al, "GPMB–Software Pipelining Branch-Intensive Loops", *26th International Symposium on Microarchitecture*, 1993, pp. 21-29.

[10] T. Kim, N. Yonezawa, W. S. Liu, and C. L. Liu, "A Scheduling Algorithm for Conditional Resource Sharing – A Hierarchical Reduction Approach". *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Systems* , Vol. 13, No. 4, April 1994, pp. 425-438.