Performance Analysis and Optimization of Schedules for Conditional and Loop-Intensive Specifications

Subhrajit Bhattacharya *
Dept. of Computer Science
Duke University
Durham, NC 27706

Sujit Dey
C&C Research Labs
NEC USA
Princeton, NJ 08540

Franc Brglez †
CBL, Dept. of ECE
North Carolina State Univ
Raleigh, NC 27695

Abstract - This paper presents a new method, based on Markov chain analysis, to evaluate the performance of schedules of behavioral specifications. The proposed performance measure is the expected number of clock cycles required by the schedule for a complete execution of the behavioral specification for any distribution of inputs. The measure considers both the repetition of operations (due to loops) and their conditional execution (due to conditional branches). We propose an efficient technique to calculate the metric. We introduce a loop-directed scheduling algorithm (LDS). The algorithm produces schedules such that the expected number of clock cycles, required by the schedule for a complete execution of the behavioral specification, is minimized. Experimental results on several conditional and loop-intensive specifications demonstrate the relevance and effectiveness of both the performance measure and the scheduling algorithm.

I. INTRODUCTION

Several existing scheduling algorithms specifically target behavioral specifications containing a large number of mutually exclusive paths due to the presence of conditionals [1, 2, 3, 4, 5, 6]. The following metrics are used to evaluate the performance of the schedules generated by these algorithms: (1) the number of clock cycles required to execute the longest and shortest paths (LP/SP), and (2) the number of clock cycles required, averaged over all paths in the description. These metrics consider *simple* paths, that is paths without repetition of operations. Moreover, it is assumed that the branches of an if-then-else have equal probability of execution.

In this paper, we focus on behavioral specifications with nested conditionals and loops. The number of times a loop is executed is not fixed and in general depends upon the inputs. Also, branch probabilities of conditionals, like if-then-else and case statements, may be unequal. For such specifications, given a schedule, the number of clock cycles required by the schedule for a complete execution of the behavioral specification depends on the inputs. As a metric to measure the performance of a schedule, we propose the expected number of clock cycles required by a schedule to execute the behavioral specification for any distribution of inputs. We present an efficient and effective method for calculating this metric based on Markov chain modeling of schedules [7]. We also introduce a loop-directed scheduling algorithm (LDS), which produces schedules such that the expected number of clock cycles required by the schedule to execute the behavioral description is minimized.

To evaluate the proposed performance analysis technique and the new scheduling algorithm, we schedule a number of descriptions using the path-based scheduling algorithm [2] and the proposed LDS algorithm. When the performance of the two schedules are compared using the LP/SP metrics, both perform equally well. However, the proposed metric of expected number of clock cycles, as well as actual simulation results, reveal a large variance in the number of clock cycles required by the two schedules. This demonstrates, (a) the relevance and accuracy of the proposed performance measure, and (b) the effectiveness of LDS in scheduling to minimize the expected number of clock cycles.

In Section II we motivate the need for a new metric to estimate the performance of schedules for designs with conditional loops and conditional branches. In Section III, we define the metric and propose an effective procedure to calculate it. A new loop-directed scheduling algorithm, LDS, is presented in Section IV, followed by experimental results in Section V.

II. MOTIVATION

We consider behavioral descriptions in which the control flow between the sequence of operations is explicitly specified, and derive a corresponding control flow graph (CFG). An example control flow graph for the *send* process, which is part of the X.25 communications protocol [8], is shown in Figure 1. In the CFG, each operation of the behavioral description is represented as a node. Arcs between nodes represent the flow of control specified by the behavioral description.

Two possible schedules for the *send* process of Figure 1 are shown in Figure 2(a) and Figure 2(b). We derive the schedule in Figure 2(a) using the path-based scheduling algorithm presented in [2]. The schedule in Figure 2(b) has been derived using LDS, a new loop-directed scheduling algorithm explained in Section IV.

Scheduling a CFG consists of clustering the operations of the CFG into states. The control flow of the operations in a state of the schedule forms a rooted and acyclic CFG which is a subgraph of the CFG corresponding to the behavioral description. For example, the operations in state s_4 of the schedule in Figure 2(a) include operations 13, 14, 15, 16, 17, 18, and 19, with operation 13 as the root. For both the schedules shown, the *start* state is s_0. There is an arc from state s_i to state s_j if there is an arc, in the CFG of the behavioral description, from any operation in state s_i to the root operation in state s_j. A state may have more than one outgoing arc, for example state s_4 in Figure 2(a). However, the arcs themselves are conditional, and only one of the conditions is TRUE at any time. Given that the state

* S. Bhattacharya was supported by a grant from CCRL, NEC USA.
† F. Brglez was supported in part by a grant from SRC.

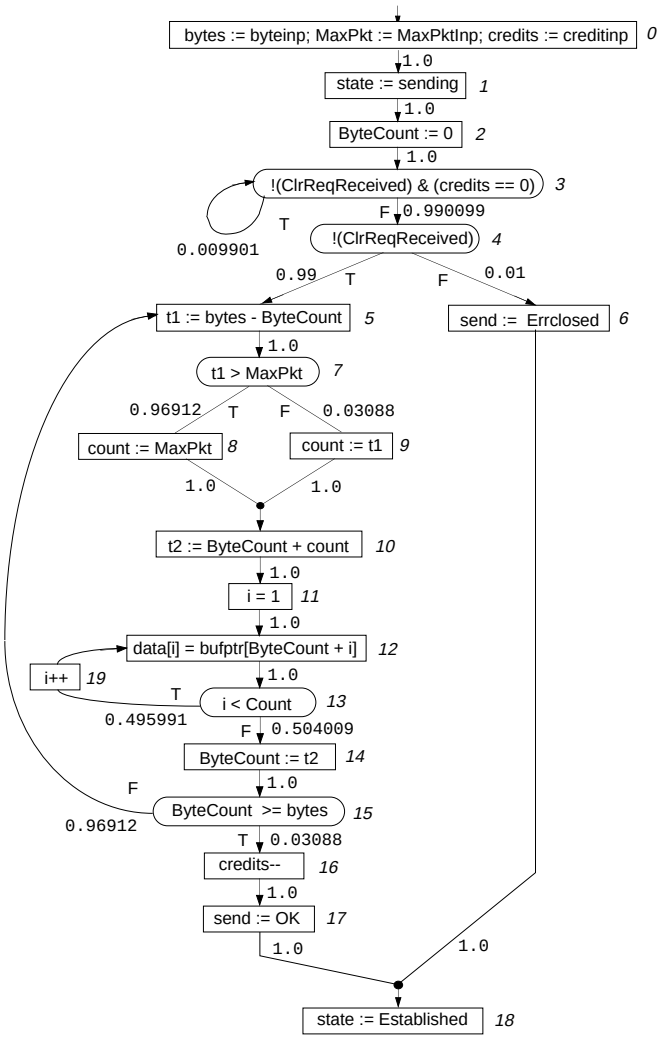

Figure 1: The CFG for the *send* process of the X.25 communications protocol. Branch probabilities are indicated on the branches.

being executed in the current clock cycle has multiple outgoing arcs, the state at the tail of the arc whose condition evaluates to TRUE is executed in the next clock cycle. Executing the schedule is equivalent to executing the behavioral specification. We assume that each state of a schedule requires one clock cycle for execution.

A path in a CFG is a sequence of CFG operations, such that if op_i and op_{i+1} are successive operations in the sequence, then there is an arc from op_i to op_{i+1} in the CFG. Consider execution of one of the longest paths (LP) in the CFG of Figure 1, $\langle 0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 14, 15, 16, 17, 18\rangle$. The schedule in Figure 2(a) goes through the states $\langle s_0, s_2, s_3, s_4\rangle$ to execute this sequence. The schedule in Figure 2(b) goes through states $\langle s_0, s_1, s_4, s_6\rangle$ to execute the same sequence. Hence both schedules require 4 clock cycles to execute the longest path. Similarly, both schedules can execute the shortest path (SP), $\langle 0, 1, 2, 3, 4, 6, 18\rangle$, in one clock cycle by executing the state s_0. Thus, the SP/LP measure predicts that both schedules have the same performance.

Note that both the paths, LP and SP, are *simple paths*, that is, they do not contain repeated operations. For the CFG of Figure 1, for inputs ($byteinp = 3, MaxPktInp = 2, credits = 1, ClrReqReceived = 0$), the **complete execution sequence** which arises from a **complete execution** of the behavioral description follows: $\langle 0, 1, 2, 3, 4, 5, 7, 8, 10, 11, 12, 13, 19, 12, 13, 14, 15, 5, 7, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18\rangle$. Note that operations are repeated due to the loops present. To execute the above specified sequence of operations, the schedule in Figure 2(a) executes the states $\langle s_0, s_2, s_3, s_4, s_2, s_4, s_5, s_2, s_3, s_4\rangle$, requiring 10 clock cycles. The schedule in Figure 2(b) executes the states $\langle s_0, s_1, s_4, s_3, s_4, s_5, s_1, s_4, s_6\rangle$, requiring 9 clock cycles. For inputs ($byteinp = 7, MaxPktInp = 2, credits = 1, ClrReqReceived = 0$), it can be shown that the schedule in Figure 2(a) requires 21 clock cycles while the schedule in Figure 2(b) requires 18 clock cycles.

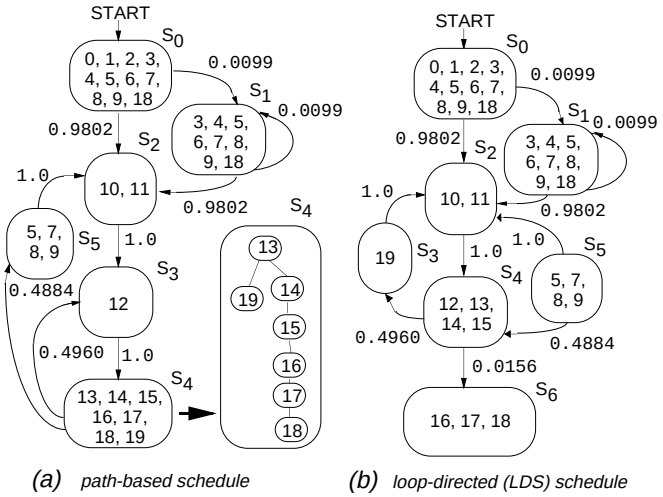

Figure 2: Two possible schedules for the *send* process. Transition probabilities are indicated on each state transition arc.

Assume that the two input combinations specified above are the only ones that can occur, and that they occur with probability $p = 0.1$, and $p = 0.9$ respectively. Then the expected number of clock cycles required by the first schedule to execute the behavioral description is $(10 \times 0.1 + 21 \times 0.9) = 18.9$. The expected number for the second schedule is $(9 \times 0.1 + 18 \times 0.9) = 16.2$.

The above example demonstrates the need for considering the execution of all possible complete execution sequences of the CFG, with repetition of operations, and not only simple paths where operations are not repeated. Such cases arise while evaluating the performance of a schedule of behavioral descriptions that have conditional loops. Also, we observe that the performance of schedules can vary significantly depending upon how loops are scheduled. This motivates us to propose a loop directed scheduler which produces schedules to minimize the number of clock cycles required to execute the behavioral description.

III. Performance Analysis: The Expected Number of Clock Cycles of a Schedule

This section defines the new metric proposed to measure the performance of a schedule, which is the expected number of clock cycles required by the schedule to execute the behavioral description. We propose an effective procedure to evaluate the metric for any schedule. We use the example of the *send* process of the X.25 communications protocol [8] to illustrate the procedure.

A. The Performance Metric and its Computation

Expected Number of Clock Cycles of a Schedule. Let the states of the schedule be s_0 to s_n. We assume that each state requires one clock cycle for execution. Let the random variable X_i, the state execution count, be the expected number of times the state s_i is executed or visited during a execution of the behavioral specification. The expected number of clock cycles required by the schedule to execute the behavioral description, denoted by X_{sch} is:

$$X_{sch} = \sum_{i=0}^{n} X_i \quad (1)$$

A behavioral description modeled as a finite, discrete-time, homogeneous Markov chain [7] allows a simple formulation which can be used for calculating the state execution counts X_i, and hence X_{sch}. The model assumes that the probability of going from operation op_i to operation op_j of the behavior is independent of the operations that have been executed before op_i. A schedule can be modeled as a homogeneous Markov chain under similar assumptions about transitions between states. The assumption under the homogeneous Markov chain model is not true in general, but as will be validated by simulation results, the modelling is very accurate.

Let s_i and s_j be any two states in a schedule such that there is a state transition arc from state s_i to state s_j. The *state transition probability*, $p_S(i,j)$, is the probability of executing s_j immediately after s_i. If s_0 is the initial state of the schedule, then the following relationship holds between the state execution counts and the state transition probabilities:

$$X_0 = 1 \quad (2)$$

$$X_i = \sum (X_j \times p_S(j,i)) \quad \forall j, such\ that\ s_j\ has\ a\ transition\ arc\ to\ s_i \quad (3)$$

X_0 is set equal to one, since for any execution of the behavioral description, the initial state is executed only once. Also, since state s_i can be reached from state s_j with probability $p_S(j,i)$, the number of times s_i is executed immediately after s_j is given by $(X_j \times p_S(i,j))$. Such a product term exists for every state that has an arc to s_i. The summation of the product terms equals X_i, the expected number of times s_i is executed. A solution to the above set of linear equations exists if the Markov chain model of the schedule is irreducible and aperiodic [7]. Equivalently, a solution exists if none of the loops in the behavioral description are executed forever.

The state transition probabilities can be formulated in terms of the path probabilities of the behavioral description. The path probabilities themselves can be formulated in terms of the branch probabilities.

Calculating the Behavioral Branch Probabilities. The *branch probabilities* of a behavioral description, $p_B(i,j)$, is the probability of executing operation op_j immediately after operation op_i. In the control flow graph corresponding to the *send* process shown in Figure 1, the branch probability $p_B(7,8) = 0.96912$, while $p_B(7,9) = 0.03088$. Since the behavioral description can be modeled as a homogeneous Markov chain [7], the probability of going from operation op_i immediately to operation op_j, is independent of the operations that have been executed before op_i. We use this assumption to estimate the branch probabilities by simulating the given behavioral description with different sets of inputs. The inputs can follow any specified probability distribution. The range of inputs can vary too. For example, for the *send* behavioral description, if the input *bytes* can be any positive 7 bit number, then its range is from 0 to $(2^7-1) = 127$. During the simulation of the behavioral description, every time the program executes some op_j after op_i, we increment the execution counts of both the operation op_i and the branch ($op_i \rightarrow op_j$), $C[i]$ and $B[i,j]$ respectively. Finally, after all the simulation runs have been completed, $p_B(i,j)$ is set equal to $B[i,j]/C[i]$.

Calculating the Schedule's State Transition Probabilities. Let $P = \langle op_i, op_{i+1}, op_{i+2}, \ldots, op_{k-1}, op_k \rangle$ be any path in a behavioral description. If we assume a homogeneous Markov chain model for the behavioral description, the probability of executing all the operations on the path in succession, given that op_i is the statement of the behavior that has been most recently executed, is given by:

$$Prob(P) = p_B(i,i+1) \times p_B(i+1,i+2) \times \ldots \times p_B(k-1,k) \quad (4)$$

Consider paths where the first operation is the root of state s_i, the last operation is the root of state s_j, and the remaining operations have been scheduled in state s_i. Let the set of paths satisfying the above property be $\{P_k, P_{k+1}, \ldots, P_m\}$. The probability of a state transition from state s_i to state s_j is:

$$p_S(i,j) = Prob(P_k) + Prob(P_{k+1}) + \ldots + Prob(P_m) \quad (5)$$

Equation (5) is valid since all the paths are mutually exclusive, that is, they will never be executed at the same time. In Figure 2(a) and Figure 2(b), two schedules for the *send* process are shown along with the state transition probabilities, calculated using the branch probabilities of the CFG in Figure 1. For the schedule in Figure 2(a), the state transition probability $p_S(4,3) = 0.4960$, and $p_S(4,5) = 0.4884$.

B. An Example: The Send Process

In this section we illustrate the procedure for estimating the expected number of clock cycles required by the path-based schedule [2] for the *send* process, using the steps detailed in Section III-A.

The Branch Probabilities. We simulate the behavioral description of the *send* process to calculate its branch probabilities. In general, the input pattern used for simulation could follow any distribution, and have a user-specified range. For illustration we assume the inputs *MaxPktInp* and *credits* were set to 2 and 1 respectively. The input *byteinp* is assumed to follow a uniform distribution in the range 0 to 127 and was generated using the UNIX uniform random number generator *rand()*. The input *ClrReqReceived* was generated to be 0 or 1 with probability 0.99 and 0.01 respectively. A representative set of branch probabilities from our experiments on the *send* process is shown in Figure 1.

Schedule's State Transition Probabilities. Consider the path-based schedule of the *send* process shown in Figure 2(a). We want to calculate $p_S(0,2)$. State s_2 can be reached from state s_0 along two possible paths: $P1 = \langle 0,1,2,3,4,5,7,8,10 \rangle$, and $P2 = \langle 0,1,2,3,4,5,7,9,10 \rangle$. The probability that path *P1* is executed to reach state s_2 from s_0, using (4) and the branch probabilities of Figure 1 is, Prob(P1)$= p_B(0,1) \times p_B(1,2) \times \ldots \times p_B(8,10) = 0.95$. Similarly, Prob(P2) $= 0.03$. Hence, the probability of reaching state s_2 from state s_0 using (5) is, $p_S(0,2)$ = Prob(P1) + Prob(P2) $= 0.95 + 0.03 = 0.98$. The remaining transition probabilities shown on the edges of the schedule in Figure 2(a) can be calculated similarly.

Expected number of Clock Cycles. We use the state transition probabilities of the path-based schedule in Figure 2(a) to illustrate the process of calculating the expected number of clock cycles required by the schedule. Using (2) and (4), we get:

$X_0 = 1$

$X_1 = 0.01X_0 + 0.01X_1$

$X_2 = 0.98X_0 + 0.98X_1 + 1.0X_5$
$X_3 = 1.0X_2 + 0.49X_4$
$X_4 = 1.0X_3$
$X_5 = 0.49X_4$
The solution is, $\{X_0, X_1, X_2, X_3, X_4, X_5\}$ = { 1, 0.01, 31.98, 63.46, 63.46, 30.99 }. The expected number of clock cycles required, by the schedule, to execute the complete behavioral description, from (1) is, $X_{sch} = \sum_{i=0}^{5} X_i = 1 + 0.01 + 31.98 + 63.46 + 63.46 + 30.99 = 190.9$. The expected number of clock cycles required by the LDS schedule, based on the branch probabilities of Figure 1, is 159.91.

An alternative method for estimating the proposed performance measure is to simulate the schedule. For applications which require repeated use of the performance measure to explore different scheduling alternatives, the method presented above is much faster. Simulating the schedule every time it changes is a time consuming process. Using the technique presented, a simulation is performed only once on the behavioral description. The performance for each alternative schedule can then be rapidly estimated using the fast probabilistic method.

IV. Scheduling To Minimize the Expected Number of Clock Cycles

The loop directed scheduling algorithm (LDS) produces schedules such that the expected number of clock cycles required by the schedule, to execute the behavioral description, is minimal. It is an extension of the path-based scheduling algorithm [2]. Path-based scheduling considers only simple paths without repeated operations. It can be shown that LDS implicitly considers all valid execution sequences, which includes sequences with repetition of operations, by going across loops. The main differences between LDS and path-based scheduling are explained in the following sections. The complete LDS algorithm is explained in Section IV-C.

A. Going Back Across Loops

In Figure 3(a), we show a fragment of the CFG from the GCD example [9]. The operations in the CFG of Figure 3 have to be scheduled, subject to the constraint that no two data dependent operations can be in the same state, that is, no data chaining is allowed. Two operations in a sequence of operations are data dependent when the output of one of the operations, op_i, is an input of the other, op_j, where op_i precedes op_j in the sequence.

The path-based algorithm "breaks" all the loops of the CFG by deleting feedback arcs before scheduling the operations. The resultant CFG consists of only simple paths, without repetition of operations. For example, in the case of the CFG in Figure 3(a), the feedback arc $\langle 1, 0 \rangle$ is deleted, and the only path considered by the path-based algorithm is $\langle 0, 1 \rangle$. Since operation 1 was using the output *x* of operation 0, scheduling them in the same state, with operation 1 executed after operation 0, would violate the constraint of no data chaining. Hence operation 1 had to be put in a different state than operation 0. The schedule produced by path-based scheduling is shown in Figure 3(b).

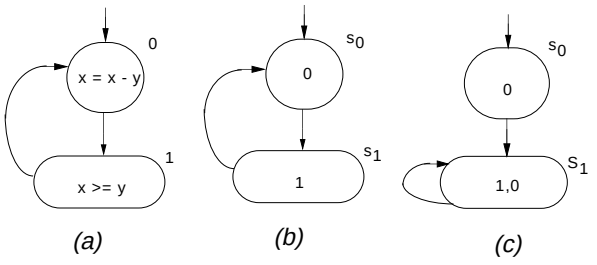

Figure 3: (a)A fragment of the CFG of the GCD example. Two schedules, (b) path-based and (c) loop directed, under constraint that data dependent operations not allowed in same state.

In our approach, we do not delete feedback arcs, and consider the repetition of operations. Due to the feedback arc from op_1 to op_0 in the CFG of Figure 3, there is another path consisting of the sequence $\langle 1, 0 \rangle$. In this sequence, operation op_0 is not data dependent on operation op_1. Hence, both the operations in the sequence $\langle 1, 0 \rangle$ are scheduled in the same state without violating the no data chaining constraint, giving the loop-directed schedule shown in Figure 3(c).

The schedule in Figure 3(b) requires two clock cycles to execute the loop. The schedule in Figure 3(c) requires only one clock cycle, except for the first iteration of the loop when it requires two clock cycles. Thus, going back across the loop allows producing a schedule which requires a smaller number of clock cycles to execute the loop, than the number required if the feedback arc had been deleted.

B. Moving Control Steps Outside Loops

It may be possible to reduce the number of control steps required to execute loops at the expense of increasing the total number of states in the schedule. As an illustration, consider the CFG shown in Figure 4(a). It has been extracted from the CFG of the *send* process in Figure 1. It has to be scheduled under resource constraints of two comparators and one ALU unit which supports the $+$ and $-$ operations. A path-based schedule and a LDS schedule are given in Figure 4(b) and Figure 4(c) respectively.

Note that since there is only 1 ALU, a control step has to be introduced between operation 3 and operation 4 of the CFG in Figure 4(a). Hence, operation 4 starts a new state. Consider the two paths starting at operation 4 in Figure 4(a), $P1 = \langle 4, 5, 8 \rangle$ and $P2 = \langle 4, 5, 6, 7 \rangle$. Since in path *P1*, operations 4 and 8 have to be implemented on the single available ALU, a control step has to be introduced between operations 4 and 8. Similarly along *P2*, a control step needs to be introduced between operations 4 and 7.

The primary objective of the path-based scheduling algorithm is to introduce control steps in such a way that executing *simple* paths takes a minimum number of control steps. However, if the control steps can be introduced in more than one place and still satisfy the

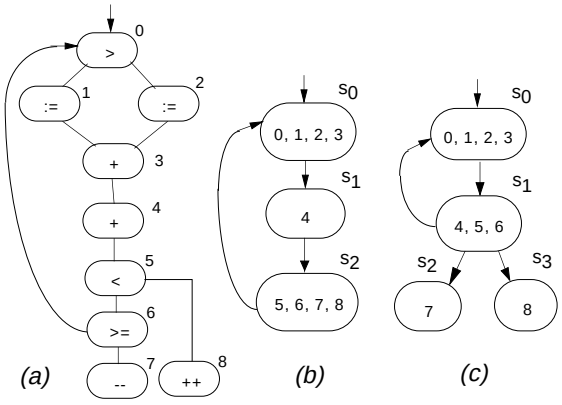

Figure 4: (a)A fragment of the CFG extracted from the *send* process in Figure 1. Two schedules, (b) path-based and (c) loop directed, under resource constraint of 2 comparators and 1 $(+/-)$ ALU.

primary objective, the path-based scheduling algorithm introduces the control steps so as to minimize the total number of states in the schedule. To schedule the operations in the paths $P1$ and $P2$ with minimum number of states, the path-based algorithm will introduce a control step between operations 4 and 5 in both paths, $P1$ and $P2$. Operation 4 is scheduled into state s_1 as shown in Figure 4(b). The remaining operations, operations $\langle 5, 8\rangle$ of $P1$ and $\langle 5, 6, 7\rangle$ of $P2$ can be scheduled into a single state s_2. They can be scheduled into the same state since both the subsequences $\langle 5, 8\rangle$ and $\langle 5, 6, 7\rangle$ satisfy the resource requirements, they belong to mutually exclusive paths, and they have the same first operation, operation 5. To execute each simple path from operation 0 to operations 7 or 8, the path-based schedule requires the minimum number of clock cycles. Also, the total number of states in the schedule, which is three, is minimum.

The LDS scheduling algorithm minimizes the number of control steps required to execute any sequence of repeated operations due to loops, to minimize the expected number of clock cycles required by the schedule to execute a behavioral description. Consider scheduling the CFG shown in Figure 4(a) under the same resource constraint of 1 ALU and 2 comparators. To satisfy the resource constraints, the loop-directed scheduler introduces a control step between operations 5 and 8 along $P1$, and a control step between operations 6 and 7 along $P2$. The corresponding loop-directed schedule is shown in Figure 4(c). We show that when paths with repeated operations are considered, as opposed to simple paths without repeated operations, the loop-directed schedule requires less clock cycles than the path-based schedule.

Consider execution of the sequence $\langle$0, 1, 3, 4, 5, 6, 0, 2, 3, 4, 5, 6$\rangle$ of the CFG of Figure 4(a), where the sequence has repeated operations. The LDS schedule executes the sequence by executing the states $\langle s_0, s_1, s_0, s_1\rangle$, which requires four clock cycles. To execute the same sequence, the path-based schedule executes the states $\langle s_0, s_1, s_2, s_0, s_1, s_2\rangle$, requiring 6 clock cycles.

The loop-directed schedule of Figure 4(c) has four states as compared to the three states of the path-based schedule in Figure 4(b). Experimental results reported in Section V show that loop-directed schedules sometimes have less states than the path-based schedules. Even when the loop-directed schedules require more states, the area or delay penalty of designs synthesized using loop-directed scheduling as compared to those synthesized using path-based scheduling is nominal.

C. The Loop Directed Scheduling Algorithm (LDS)

We briefly describe the loop-directed scheduling algorithm (LDS) using the pseudo-code given below and the CFG shown in Figure 1. A detailed description is given in [10]. The CFG has to be scheduled under the resource constraints of one $(+/-)$ ALU and 2 comparators. The schedule produced by LDS is shown in Figure 2(b).

The start state of the schedule should have the start operation of the CFG as the root operation. Hence, *root_ops* is initialized to include op_0, the start operation of the CFG in Figure 1. Procedure *create_state* uses depth-first search in the CFG, starting at the operation op_i selected in line 2 of the code. The depth-first search along a path is halted either if (a) a constraint is violated or (b) if an operation is reached which already belongs to the path. Consider state s_0 of the schedule in Figure 2(b) with operation op_0 as the root operation. The operations along four paths in the CFG of Figure 1 are scheduled into state s_0, where every path started with the root operation op_0. The four paths scheduled into s_0 are: $P1 = \langle 0, 1, 2, 3\rangle$, $P2 = \langle 0, 1, 2, 3, 4, 5, 7, 8\rangle$, $P3 = \langle 0, 1, 2, 3, 4, 5, 7, 9\rangle$, and $P4 = \langle 0, 1, 2, 3, 4, 6, 18\rangle$. Note from Figure 1 that along $P1$, operation 3 is a successor of itself because of the feedback arc present. It can not be included in $P1$ since it has already been included once and including it would violate condition (b) above. Similarly, along $P4$, operation 0 can not be included since it already belongs to the path. Including operation 10 along path $P1$ or $P2$ is not possible since from condition (a) above, including it violates the resource constraint of one ALU.

The set of possible new_root_ops identified as a result of creating a state are the operations at which the depth-first search terminated. For s_0, they are 0, 3 and 10. Since state s_0 has already been created with operation 0 as the root operation, only operation 3 and operation 10 will be used in subsequent iterations of the loop beginning at line 2 of the code to create new states.

It can be shown that the following pseudo-code, as briefly explained above, implements the ideas in Section IV-A and Section IV-B. Detailed pseudo-code of the algorithm can be found in [10].

```
LDS(CFG, constraints){
1. root_ops ← {op0 | op0 is start operation of CFG};
2. while (∃opi ∈ root_ops for which a state
   in the schedule has not been created){
3.    (state, new_root_opns) ←
      create_state(CFG, constraints, opi);
4.    add_state_to_sched(state);
5.    root_ops ← root_ops ∪ new_root_ops;
   }
}
```

V. Experimental Results

To evaluate the proposed performance analysis technique and the new scheduling algorithm, we schedule a number of descriptions using the path-based (PB) scheduling algorithm [2] and the proposed LDS algorithm. We synthesize the following behavioral descriptions: GCD, Traffic Controller [9], and the *send* process of the X.25 communications protocol [8]. The synthesis results are reported in Tables 1 and 2. The two schedules derived for each specification are given under column *Sch*, and they satisfy the constraints specified under column *Constraints*.

The three step process for estimating the performance of a schedule, described in Section III-A, has been incorporated in a program PERSIS (PERformance analySIS). First, we calculate the branch probabilities for the behavioral description by simulation using a specified input distribution. For the experiments reported, the inputs for the description were generated with the UNIX uniform random number generator *rand()*. The number of input vectors applied for simulation is user-specified. We fixed the number of input vectors to be used by increasing the number till the branch probabilities after simulation stopped changing in the first two places of the decimal. Simulating the GCD description required 10,000 input vectors to achieve the required level of accuracy. However, the *send* and the Traffic descriptions required just 100.

The next two steps use numeric methods as detailed in Section III-A. The set of equations, (4), is solved using the linear equations solver available in the Mathematica software. Results of our estimation procedure for the expected number of clock cycles required by the schedules is given under column PERSIS in Table 1.

We also simulated the schedules using the same set of inputs which were used to determine the behavioral description branch probabilities. The average number of clock cycles required by the schedule

Design	Constraints	Sch	Clock Cycles				
			LP/ SP	Expected		Std Dev	Worst Case
				PERSIS	Sim		
GCD	No data dependent operations in a state	PB	4/2	144.3	142.8	3.45	39785
		LDS	4/2	78.3	76.6	1.72	19894
X.25 (send proc)	1 alu (+/−) 2 comp (>)	PB	4/1	191.9	190.4	10.67	381
		LDS	4/1	160.9	159.3	8.89	318
Traffic	1 incr (++)	PB	4/4	71.8	69.8	1.67	115
		LDS	4/4	39.4	38.4	0.84	61

Table 1: Comparing performance of path-based and loop-directed schedules using various metrics.

to execute the description for the set of inputs, as obtained by simulation, is given under the column *Sim*. This is an exact value of the expected number of clock cycles required by the schedules to execute the behavioral description for the given set of inputs. For example, the expected number of clock cycles required by the LDS schedule for the GCD example as calculated by our performance analysis program, PERSIS, is 78.3. The value obtained by simulation is 76.6. Thus our procedure is within 2% of the simulation result for this example, and also for all the other examples. The results establish the validity of our estimation procedure.

The results in Table 1 show that the longest path/shortest path (LP/SP) metrics predict the performance of the two schedules to be the same for each of the three benchmarks used. However, the proposed performance metric and also actual simulation of the schedules revealed a large variance in the expected number of clock cycles required by the path-based and LDS schedules (column *PERSIS/Sim*). For the GCD benchmark, both the schedules require 4 clock cycles and 2 clock cycles to execute the longest and shortest paths of the specification. However, the expected number of clock cycles required by the LDS schedule, as estimated by PERSIS, is 78.3 clock cycles as opposed to 144.3 clock cycles needed by the path-based schedule. For the GCD benchmark, the loop-directed schedule requires 87% less clock cycles than the path-based schedule. For *send* and Traffic, LDS requires 20% and 82% less clock cycles than the corresponding path-based schedule respectively.

The standard deviation (*Std Dev*) and the *Worst Case* was obtained by simulation of the schedule. For all three cases, the standard deviation from the expected number of clock cycles, and the worst case clock cycles for the LDS schedule is less than the corresponding figures for the path-based schedule.

Design	Sch	S	ST	Area [mm^2]	Exp CC	CP [ns]	Exp SP[ns]
GCD	PB	7	14	4.80	142.8	29.44	4204
	LDS	6	12	4.47	76.6	29.38	2251
X.25 (send proc)	PB	6	9	132.7	190.4	34.42	6554
	LDS	7	10	133.1	159.3	35.23	5612
Traffic	PB	8	13	0.693	69.8	19.27	1345
	LDS	10	22	0.716	38.4	19.50	748

Table 2: Effect of improved performance on area/delay statistics.

Table 2 shows the effect of minimization of the expected number of clock cycles on some important circuit parameters. The total number of states of the schedules is reported under column *S* and state transitions under column *ST*. Each RT-level circuit generated from the schedules is subjected to technology-dependent delay optimization, including fanout optimization, using the SIS technology mapper [11] and the *lib2.genlib* standard cell SCMOS 2.0 library [12]. Subsequently, OASIS [13] place and route tools are used to obtain the standard cell layout. Columns *CP* and *Area* report the clock period and area of the final implementations. In all the cases, the minimization of the expected number of clock cycles could be achieved without significantly affecting the area or delay of the circuits, even when the number of states and state transitions increased. The expected sampling period (*Exp SP*) is the product of the expected clock cycles (*Exp CC*) and clock period. Since the clock period is nominally the same for the circuits produced from the path-based and loop-directed schedules, the expected sampling period could be reduced by approximately the same percentages as the clock cycles for all the benchmarks.

VI. Conclusions

This paper proposed a new approach to estimate the performance of a given schedule and a new scheduling algorithm. The performance measure calculated is the expected number of clock cycles required by a schedule to execute *any* valid sequence of operations in the specification. The proposed loop-directed scheduling algorithm (LDS) produces schedules such that the performance of the schedule is optimized for execution of any valid sequence of operations in the specification, including sequences that have repeated operations.

Acknowledgments: We thank T. Misawa, K. Watanabe, and A. Merchant for helpful discussions.

References

[1] K. Wakabayashi and T. Yoshimura. A Resource Sharing and Control Synthesis Method for Conditional Branches . In *Proc. of the IEEE ICCAD*, 1989.

[2] R. Camposano and R. A. Bergamaschi. Synthesis using Path-Based Scheduling: Algorithms and Exercises. In *Proc. of the 27th ACM/IEEE DAC*, June 1990.

[3] R. Camposano. Path Based Scheduling for Synthesis. *IEEE Trans. on CAD*, Jan 1991.

[4] T. Kim, J. W. S. Liu, and C. L. Liu. A Scheduling Algorithm For Conditional Resource Sharing . In *Proc. of the IEEE ICCAD*, 1991.

[5] K. Wakabayashi and H. Tanaka. Global Scheduling Independent of Control Dependencies Based On ConditionVectors . In *Proc. of the 29th ACM/IEEE DAC*, 1992.

[6] S.H. Huang, Y.L. Jeang, C.T. Hwang, Y.C. Hsu, and J.F. Wang. A Tree-Based Scheduling Algorithm For Control-Dominated Circuits. In *Proc. of the 30th ACM/IEEE DAC*, 1993.

[7] K. S. Trivedi. *Probability and Statistics with Reliability, Queuing, and Computer Science Applications.* Prentice Hall, Englewood Cliffs, N.J., 1982.

[8] A. S. Tanenbaum. *Computer Networks.* Prentice Hall, Englewood Cliffs, N.J., 1989.

[9] 1992 High-Level Synthesis Workshop. Benchmarks available in the HLSW92 directory via anonymous ftp from mcnc.mcnc.org.

[10] S. Bhattacharya, S. Dey, and F. Brglez. Performance Analysis and Optimization of Schedules for Conditional and Loop-Intensive Specifications. Technical Report 93-C049, C&C Research Labs, NEC USA, November 1993.

[11] E.M. Sentovich, K.J. Singh, C. Moon, H. Savoj, R.K. Brayton, and A. Sangiovanni-Vincentelli. Sequential Circuit Design using Synthesis and Optimization. In *Proc. of the ICCD*, October 1992.

[12] S. Yang. Logic Synthesis and Optimization Benchmarks, User Guide 3.0. In *Intl. Workshop on Logic Synthesis*, MCNC, Research Triangle Park, NC, May 1991.

[13] K. Kozminski (ed.). *OASIS Users Guide*. MCNC, MCNC, Research Triangle Park, N.C. 27709, 1992.