An Enhanced Co-Scheduling Method using Reduced MS-State Diagrams

R. Govindarajan
Supercomputer Edn. & Res. Centre
Computer Science & Automation
Indian Institute of Science
Bangalore 560 012, India
govind@{serc,csa}.iisc.ernet.in

N.S.S. Narasimha Rao
Novell Software
Development India Ltd.
Garvephavipalya
Bangalore 560 068, India
nnarasimharao@novell.com

E.R. Altman
IBM T.J. Watson
Research Center
Yorktown Heights
NY 10598, U.S.A.
erik@watson.ibm.com

Guang R. Gao
Electrical & Computer Engg.
University o f Delaware
Newark
DE 19716, U.S.A
ggao@eecis.udel.edu

Abstract

Instruction scheduling methods based on the construction of state diagrams (or automata) have been used for architectures involving deeply pipelined function units. However, the size of the state diagram is prohibitively large, resulting in high execution time and space requirement.

In this paper, we present a simple method for reducing the size of the state diagram by recognizing *unique* paths of a state diagram. Our experiments show that the number of paths in the reduced state diagram is significantly lower — by 1 to 3 orders of magnitude — compared to the number of paths in the original state diagram.

Using the reduced MS-state diagrams, we develop an efficient software pipelining method. The proposed software pipelining algorithm produced efficient schedules and performed better than Huff's *Slack Scheduling* method, and the original *Co-scheduling* method, in terms of both the initiation interval (II) and the time taken to construct the schedule.

1 Introduction

Function units in modern architectures involve increasingly more complex resource usage. Deeper pipelines and wide-word instructions are staging a resurgence. Processors capable of issuing 8 instructions per cycle are on the horizon, and structural hazard resolution is expected to be more complex. With such complex resource usage, the instruction scheduler must check and avoid any *structural hazard*, e.g., contention for hardware resources by instructions. Conventional instruction scheduling and software pipelining methods [4, 6, 7, 10, 12, 13, 17, 18] accomplish this by maintaining a *global resource table* (or *Modulo Reservation Table (MRT)* in the case of software pipelining) to model the resource usage. In these methods instructions are scheduled using a naive method, attempting to schedule operations at each time slot until a conflict-free slot is found. Further, the approach followed here is greedy and involves several tries and retries.

Recently, a finite state automaton (FSA)-based instruction scheduling technique [2, 14, 16], which uses the notion of forbidden/permissible latencies, and legal initiation sequences [3, 11, 15] to efficiently detect resource conflict has been proposed. The FSA-based methods have effectively reduced the problem of checking structural hazards to a fast table lookup. The *Co-scheduling* framework proposed by us in [8] is a software pipelining method that makes use of classical pipeline theory and state diagram construction. The constructed state diagram, called a Modulo-Scheduled (MS)-state diagram, represents valid latency sequences. Each path in the MS-state diagram corresponds to a set of time steps at which different instructions can be initiated in the given pipeline under modulo scheduling without incurring any structural hazard. In the Co-scheduling method, a single path in the MS-state diagram, and the time steps corresponding to it are used to guide the software pipelining method.

Unfortunately, the practical use of state diagram based instruction scheduling methods [2, 8, 14, 16] is restricted due to the large size of the state diagram. In the context of Co-scheduling, the MS-state diagram may consist of a large number of paths especially for large values of the loop *initiation interval* (**II**). One basic observation that we make in this paper is that a significant number of paths in a MS-state diagram are redundant. A simple method that detects unique paths, known as *primary paths*, has been presented in this paper. A state diagram consisting only primary paths is termed as a *reduced state diagram*. Our experiments reveal a significant reduction — by 1 to 3 orders of magnitude — in the number of paths in the reduced MS-state diagram, Further, we have proposed an alternative **direct** method to derive the scheduling information represented by the reduced state diagram. The direct method proposed in this paper is faster, by a factor of 2 to 3, in terms of ex-

ecution time compared to the construction of the reduced MS-state diagram.

In this paper, the original Co-scheduling method is also enhanced by considering multiple paths of the MS-state diagram and choosing the most appropriate one based on the data dependences in the loop. The enhanced Co-scheduling method has been tested on a large number of loop kernels taken from several benchmark programs. The use of multiple offset sets and the reduced state diagrams facilitate achieving better software pipelined schedules with smaller **II** and shortens the scheduling time. Further, comparison with Huff's Slack Scheduling [10] reveals a 4-fold (or 400%) improvement in the time taken to construct the schedule. Further, the enhanced Co-scheduling method results in a moderate improvement in terms of **II**, particularly in loops which are resource-critical. Lastly, the enhanced Co-scheduling method also produces better schedules compared to original Co-scheduling [8] in terms of both **II** and the time to construct the schedule.

In the following section we motivate the need for reduced state diagram and enhanced Co-scheduling. In Section 3, we present the underlying theory of reduced MS-state diagrams. Section 4 deals with the construction of reduced MS-state diagrams and the enhanced Co-scheduling algorithm. In Section 5, we report the results of our experiments. Related works are compared in Section 6 and concluding remarks are presented in Section 7.

2 Background and Motivation

In this section we present the necessary background on Co-scheduling. Throughout this paper, we use the terms, "operations", "instructions", and "initiations" synonymously. We assume that our architecture consists of multiple functional units (FUs), where each FU is capable of executing instructions of a specific instruction class. Further, each FU is a *static* pipeline whose resource usage pattern is described by a single reservation table [11].[1]

2.1 Modulo-Scheduled State Diagram

The latency sequences that do not cause structural hazards in a pipeline operating under modulo scheduling are represented in a Modulo-Scheduled (MS) state diagram [8, 9]. The reservation table of an FU and its MS-state diagram for an $\mathbf{II} = 6$ are shown in Figure 1. A path $S_0 \xrightarrow{p_1} S_1 \xrightarrow{p_2} S_2 \cdots \xrightarrow{p_k} S_k$ in the MS-state diagram corresponds to **a latency sequence** $\{p_1, p_2, \cdots, p_k\}$. The latency sequence represents $k+1$ initiations that are made at time steps 0, p_1, $(p_1 + p_2)$, $\cdots$, $(p_1 + p_2 + \cdots + p_k)$. In modulo scheduling, these time steps correspond to the **offset values**

$$0, p_1, (p_1 \oplus p_2), \ \cdots \ , (p_1 \oplus p_2 \oplus \cdots \oplus p_k),$$

where $\oplus$ corresponds to addition modulo **II**,

As an example, the latency sequence $\{3\}$ corresponding to the path $S_0 \xrightarrow{3} S_3$ represents only 2 initiations made at time steps 0 and 3. Thus analyzing the MS-state diagram reveals valid latency sequences and the corresponding offset sets that do not cause collision. Further, in this state diagram paths $S_0 \xrightarrow{2} S_1 \xrightarrow{2} S_3$ and $S_0 \xrightarrow{4} S_2 \xrightarrow{4} S_3$ result in maximum number of initiations in the pipeline. We refer to $\{2, 2\}$ and $\{4, 4\}$ as maximum initiation latency sequences.

2.2 Co-scheduling Method

Co-scheduling was based on Huff's Slack Scheduling algorithm [10]. Co-scheduling starts with a Minimum Initiation Interval (**MII**) and attempts to schedule the loop for values of $\mathbf{II} \geq \mathbf{MII}$ until a schedule is found. For a given **II** it computes the MS-state diagram for all FUs that have structural hazards. To make the discussion concise and address only the relevant points, we will assume that there is only one type of pipeline[2]. From the MS-state diagram for the pipeline, a maximum initiation latency sequence is derived.

The basic notion of Huff's original Slack Scheduling [10] was to schedule instructions in increasing order of their *slackness*: the difference between the earliest time and the latest time at which an instruction may be scheduled. *Slack* is a dynamic measure and is updated after each instruction is scheduled. These points remain in Co-scheduling. The difference lies in how a time slot is chosen within the slack range. The original Slack Scheduling method attempts to schedule an instruction in every time step it its slack range until it finds a conflict-free slot. The Co-scheduling method, however, attempts to schedule an instruction only at pre-determined offset values (in the slack range) given by the maximum initiation latency sequence.

To clarify matters, consider the motivating example (Figure 1). Suppose we have chosen the maximum initiation latency sequence $\{2, 2\}$ and the corresponding offset values are 0, 2, and 4. Further let us assume an instruction i_1 is scheduled at time step, say, 1 and two more instructions i_2, and i_3 are to be scheduled in the same pipeline. If the slack of i_2 is $(4, 6)$, i.e., instruction i_2 can be placed at any time step from 4 to 6, the Co-scheduling method will

[1] It is possible to extend the work presented in this paper to architectures in which the pipelines share certain resources, such as the result write bus. A discussion on this is beyond the scope of this paper.

[2] This assumption (only one type of pipeline) is made only to simplify the discussion. Neither the original Co-scheduling method nor the enhanced Co-scheduling method presented in this paper is restricted by this.

Stage	Time Steps					
	0	1	2	3	4	5
1	x					
2		x	x			
3				x		

(a) Cyclic Reservation Table

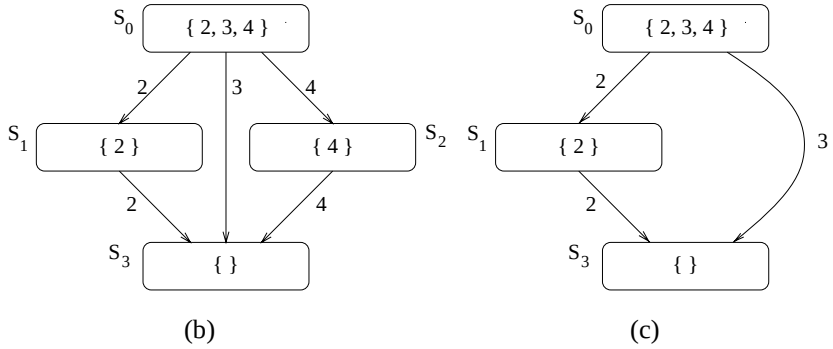

Figure 1: Full and Reduced MS-State Diagrams for Example Reservation Table

choose time step 5, so that the offset between i_1 and i_2 is 4, as governed by the chosen latency sequence.

The Co-scheduling method goes by the chosen maximum initiation latency sequence, even if this means avoiding some other latency values that would yield a legal partial schedule. Note that, in the considered example, even though i_2 can be scheduled at time step 4, with a (permissible) latency 3 with i_1, the Co-scheduling method does not attempt this. This is because if i_1 and i_2 are scheduled at times 1 and 4 (offsets 0 and 3) respectively, then instruction i_3 cannot be scheduled in the same pipe. Thus the Co-scheduling method takes a global view in scheduling instructions and deals with structural hazards much more efficiently, by using a maximum permissible latency sequence.

2.3 Reduced State Diagram: Motivation

The use of a single offset set, as done in Co-scheduling, may not be sufficient to obtain a schedule with the lowest value of **II**. We illustrate this with our example discussed in Section 2.2. Suppose there are only two instructions i_1 and i_2. Further assume that instructions i_1 and i_2 have a tight slack, requiring them to be scheduled at time steps 1 and 4 respectively. If we use a single set of *offset* values, e.g., 0, 2, and 4, as suggested by the original Co-scheduling, then one of the two instructions cannot be scheduled in the pipeline, even though the chosen latency sequence can accommodate 3 operations in the pipeline. In this case, eventually the Co-scheduling algorithm will result in an **II** of 7. However, if we had used the latency sequence $\{3\}$, or the corresponding offset values 0 and 3, then we could have scheduled the loop with $\mathbf{II} = 6$. Thus, it is beneficial, if Co-scheduling can consider all offset sets.

The number of offset sets for an MS-state diagram can be quite large (in the order of several 100,000s). Further, the number of paths in an MS-state diagram increases drastically for large values of **II**. As a consequence, the construction of the state diagram is expensive in terms of both space and time complexity, especially for large **II** values.

Consider the state diagram shown in Figure 1(b). Clearly, the states S_0, S_1, S_2, and S_3 are all distinct. However, the information represented by the paths $S_0 \xrightarrow{2} S_1 \xrightarrow{2} S_3$ and $S_0 \xrightarrow{4} S_2 \xrightarrow{4} S_3$ are not. Though the latency sequences corresponding to the above paths are different, it can be seen that the *offset* values for both of them are $0, 2$, and 4. Thus the latter path (actually any one of the two paths) is redundant. This raises the question, that even though state S_2 is distinct, can we avoid generating the state if all the paths that go through S_2 are redundant. Equivalently, can we list only the paths that lead to *distinct* offset sets? In our example, removing state S_2 and the arcs that are connected to it, does not result in any loss of information. The reduced state diagram is shown in Figure 1(c).

In this paper, we develop a method to construct the reduced MS-state diagram which consists of only unique paths. In Section 4, an enhanced Co-scheduling algorithm that uses many latency sequences, not just a maximum initiation latency sequence.

3 Reduced MS-State Diagrams

We begin with a few definitions. States S_0 and S_f correspond to start and final states in the MS-state diagram.

Definition 3.1 *A path* $S_0 \xrightarrow{p_1} S_1 \xrightarrow{p_2} S_2 \cdots \xrightarrow{p_k} S_f$ *in the MS-state diagram is called* **primary** *if the sum of the latency values does not exceed* **II**. *That is,* $p_1 + p_2 + \cdots + p_k < \mathbf{II}$. *A path is called* **secondary** *if* $p_1 + p_2 + \cdots + p_k > \mathbf{II}$.

Note that the sum of the latencies along any path in the MS-state diagram will not be equal to **II**. Otherwise, the initiation representing state S_f corresponds to the offset value 0 which causes a collision with the initiation at S_0 (the initial state).

Definition 3.2 *The set of* **permissible offsets** $\mathcal{O}$ *equals the set of all initial offset values at which an initiation is permitted.*

For the FU discussed in Section 2 (refer to Figure 1), the set of permissible offsets is $\{0, 2, 3, 4\}$. Next, we adapt the following definitions from [11, 15].

Definition 3.3 *Two offsets* o_1 *and* o_2 *belonging to* $\mathcal{O}$ *are* **compatible** *if* $(o_1 - o_2)$ `mod` $\mathbf{II}$ *is in* $\mathcal{O}$.

Definition 3.4 *A* **compatibility class** *with respect to* $\mathcal{O}$ *is a set in which all pairs of elements are compatible.*

Definition 3.5 *A* **maximal compatibility class** *is a compatibility class that is not a proper subset of any other compatible class.*

Two compatible classes for $\{0, 2, 3, 4\}$ are $\{0, 2, 4\}$ and $\{0, 3\}$. These compatibility classes are maximal, while $\{0, 2\}$ is not.

The compatibility classes of $\mathcal{O}$ are related to the offset sets of different paths in the MS-state diagram. The following lemmas establish that. The reader is referred to [9] for proofs of lemmas and theorems.

Lemma 3.1 *The offset set of any path from the start state* S_0 *to the final state* S_f *in the MS-state diagram forms a maximal compatibility class of* $\mathcal{O}$.

Lemma 3.2 *For each maximal compatibility class* C *of permissible offsets, there exists a* **primary path** *in the MS-state diagrams whose offset set* O *is equal to* C.

Using the above two lemmas, we can establish that:

Theorem 3.1 *For each secondary path from* S_0 *to* S_f *in the MS-state diagram there exists a primary path such that their offset sets are equal.*

Theorem 3.2 *A reduced MS-state diagram consisting only of primary paths contains the set of all valid offset sets that are permissible in the original state diagram.*

From the above theorems, it can be seen that though the number of paths in the original state diagram for either a fully pipelined or non-pipelined function unit is large, there is only one primary path (or one distinct offset set) in the reduced state diagram.

4 Enhanced Co-Scheduling using Reduced MS-State Diagram

In this section, we present two methods to obtain the unique offset sets. Section 4.3 deals with the enhanced Co-scheduling method.

4.1 Generation of Offset Sets using Reduced MS-State diagram

As demonstrated in Section 3 (Theorem 3.2), it is sufficient to obtain a reduced MS-state diagram consisting only of primary paths. Thus, one method to obtain a reduced state diagram is by identifying secondary paths and eliminating them. This is done two phases. In the first phase, at the time of creation of a new state, the state diagram construction method checks whether the path leading to the newly created state is secondary. If so, it marks such a state as *Redundant* and the construction of the subtrees of the state is stopped. In the second phase, the algorithm checks each state for redundant children and removes them. If all the children of a state are redundant, then the state itself is marked redundant. Subsequently, this state gets eliminated in the *recursive ascend*, that is, when its parent is checked for redundant children.

4.2 Generation of Offset Sets using Maximal Compatible Classes

Lemmas 3.1 and 3.2 establish the correspondence between the offset sets of primary paths and maximal compatible classes of the permissible offset set. Hence obtaining all maximal compatible classes is an alternative way of obtaining the offset sets. An approach to obtain the compatible classes is presented in [11] (page 99)[3]. This approach is a direct way of obtaining the set of all offset sets of the reduced MS-state diagram. We compare the performance of the offset set generation using the two methods, viz., reduced MS-state diagrams and maximal compatible classes, in Section 5.2. It should, however, be noted that the software pipelining algorithm presented in the following section is independent of the method used to obtain the set of offsets.

4.3 Enhanced Co-Scheduling Algorithm

Like Slack Scheduling algorithm, Co-scheduling uses (dynamic) slack-based priority, and *stretchability measure* to determine whether to the schedule an operation early or late in order to reduce the register pressure [10]. The main difference between the two methods is the way a conflict-free slot is chosen for scheduling an operation. For this, we consider all offset sets that have a cardinality greater than or equal to the number of operations in the loop that are mapped on to that function unit[4]. Sets with smaller

[3] It is important to note here that the generation of compatible classes was discussed in classical pipeline theory in the context of reconfiguring pipelines rather than for analyzing pipelines.

[4] For simplicity, we consider only a single instance of FU in each FU type.

cardinalities, need not be considered as they do not support the required number of instructions. The way scheduling proceeds is better explained with the help of an example. Though the example concentrates on how resource constraints are met in a single FU, the method is general enough to handle multiple FU types. The reader is referred to [9] for the detailed algorithm.

Consider, a function unit with four instructions i_1, i_2, i_3, and i_4 mapped on to it. Consider the following offset sets which support four or more initiations:

$$O_1 = \{0, 1, 7, 10\};\ O_2 = \{0, 1, 6, 7\};\ O_3 = \{0, 3, 6, 9, 12\};$$

Initially, the sets O_1, O_2, and O_3 are the *active* offset sets. Let us start with the scheduling of instruction i_1 with a slack *(3,5)*. Since this is the first instruction to be scheduled in the pipe, it has no structural hazards and can be placed in any cycle in its slack. To simplify the discussion, it is assumed that all instructions are placed as early as possible. Hence i_1 is placed at time 3.

Since offset values are relative, the first instruction is always assumed to be scheduled at an offset 0. In our example, instruction i_1 which is scheduled at time step 3 corresponds to an offset 0. Now, suppose i_2 has a slack *(10,15)*. The *Estart* time of i_2 corresponds to an offset $10 - 3 = 7$ with respect to i_1. A look at the offset sets reveals that $7, 9, 10$, and 12 are permissible offsets. Hence i_2 is scheduled at time step 10 with an offset 7 with respect to i_1. Since O_3 does not support an offset 7, it is marked *inactive*; the offset sets O_1 and O_2 are currently *active*.

Now, if instruction i_3 has a tight slack *(12,12)* with an offset 9, neither of the offset sets O_1 and O_2 can support the scheduling of i_3 at time 12. In such a case, the most recently placed operation is ejected. This may increase the number of active offset sets and hence the possibility of a placement. In our example, when i_2 is unscheduled, the offset set O_3 becomes active, and i_3 can be scheduled at time 12 (with offset 9). However, this makes offset sets O_1 and O_2 inactive, leaving O_3 as the only active set. Subsequently, when i_2 is chosen for scheduling[5], it is scheduled at time step 15, with offset 12. In a similar manner, if instruction i_4 has a slack *(19,26)*, it can be placed at one of the remaining offsets, 3 or 6. If no valid schedule is found even after ejecting a number of operations (greater than a *threshold* value), the current (partial) schedule is aborted and successive values of **II** are tried until a valid schedule is obtained.

In addition to the difference in selecting the schedule slot for an operation, enhanced Co-scheduling differs from Huff's Slack Scheduling in one other way. In forcing the placement of an operation, Huff's method ejects only conflicting operations; whereas in our approach, the operations scheduled in an FU are ejected in the reverse order in which they are scheduled. This is in order to ensure that our method does not get stuck with an active offset set. Enhanced Co-scheduling extends the original Co-scheduling method to consider multiple offset sets, and thus providing more opportunities for scheduling the loop.

[5]For simplicity, assume that i_2's slack does not change.

5 Experimental Results

In this section, we present quantitative results for reduced MS-state diagrams and the enhanced Co-scheduling method.

5.1 Reduced MS-state Diagram

First we compare the number of paths in the reduced MS-state diagram with that in the original state diagram. In order to exercise our state diagram-based method fully, for the 6 functional units considered in our experiments, we chose long reservation tables (representing deeper pipelines) with arbitrary structural hazards.[6]

Average Reduction in # Paths	FU-1	FU-2	FU-3	FU-4	FU-5	FU-6
	$8 \leq$ **II** ≤ 15					
Geo. Mean	7.5	20.8	2.5	2.6	26.1	2.2
Arith. Mean	13.5	60.7	3.0	3.4	64.7	2.5
	$16 \leq$ **II** ≤ 24					
Geo. Mean	1037.3	9084.3	54.3	52.3	2273.9	32.7
Arith. Mean	3882.5	27543.7	99.5	95.3	22490.7	55.5

Table 1: Average Reduction in the Number of **Paths**.

Table 1 shows the average reduction in the number of paths achieved by considering only the primary paths, the average being taken over different values of **II** from 8 to 24.[7] It can be seen that especially for large values of **II**, between 16 to 24, the reduction in number of paths is very significant (the geometric mean varies from 32 to 9,084). This can be further verified from the rate of increase in the number of paths for the original and reduced state diagrams plotted for FU-1 and FU-3 in Figure 2. Note that the y-axis (number of paths) in Figure 2 is on a log-scale. Lastly, for functional units FU-3, FU-4, and FU-6, the reduction in the number of paths is not so significant for the values of **II** considered in our experiments. This is because, at small **II** values, many of the latencies were forbidden for these FUs.

[6]The reservation tables of the functional units used in this experiment and in the enhanced Co-scheduling method are listed in [9] which is available over the internet.

[7]For values of **II** greater than 24, the number of paths in the original state diagram exceeds 10 Millions and all paths could not be enumerated within 30 minutes of CPU time. However, the reduced MS-state diagram has been constructed even for large values of **II** (upto 85).

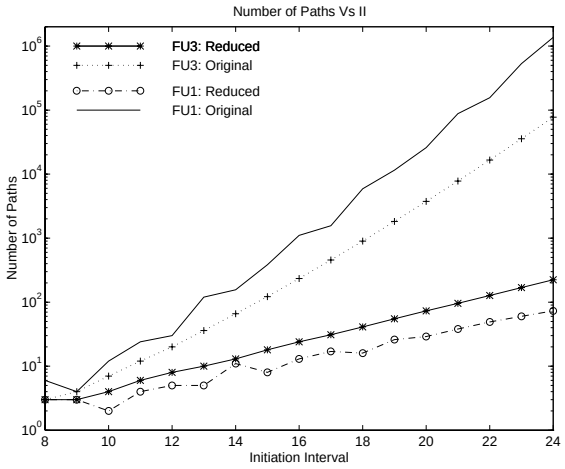

Figure 2: Number of Paths vs. **II**

5.2 Maximal Compatible Set Generation

Next we compare the reduced state-diagram approach for generating the offset sets with the direct approach of enumerating maximal compatibility classes. For the same set of reservation tables used in the previous section, and for the same **II** values, we compared the execution time, on an UltraSparc 170E, of the two approaches. The average execution time, averaged over runs for different values of **II**, for each FU are shown in Table 2. Obtaining the offset sets from the maximal compatible classes results in a 3-fold improvement in execution time for the values of **II** considered in our experiments.

Approach	**Avg. Exec. Time**					
	FU-1	FU-2	FU-3	FU-4	FU-5	FU-6
	(in milliseconds)					
Red. State Diag. (Arith. Mean)	277.7	1025.2	80.8	2933.8	138.2	17.6
Comp. Class (Arith. Mean)	70.7	146.1	93.1	519.4	74.5	7.0
Improvement (Geo. Mean)	6.9	5.3	1.9	3.8	3.1	3.7

Table 2: Execution Time of Two Methods to Generate Offset Sets

5.3 Performance of Enhanced Co-Scheduling Method

We experimented the enhanced Co-scheduling method on 1153 loops taken from a variety of scientific benchmarks. We assumed a target architecture with 7 function units: 2 Integer, 1 Load, 1 Store, 1 FP Add, 1 FP Multiply, and 1 FP Divide Units, each having complex resource usage. The reservation tables for the FP Multiply and FP Add Units correspond to those of FU-5 and FU-6, respectively, used in Sections 5.1 and 5.2.

	II – MII						
	0	1	2	3	4	5	≥ 6
No. of Cases	880	144	20	17	15	61	16
% Cases	76.3	12.5	1.7	1.5	1.3	5.3	1.4

Table 3: Performance of Enhanced Co-Scheduling

Table 3 gives a break up of the total benchmark programs in terms of how far the **II** of the constructed schedule is from the minimum initiation interval (**MII**). Our enhanced Co-scheduling found schedules at the minimum initiation interval in 880 cases. In the remaining cases, **II** of the resulting schedules was 2.58 time steps away, on an average, from the minimum initiation interval. The (arithmetic) mean time to compute a schedule is 2.9 milliseconds, while the median for this is 1.1 milliseconds. The time to construct a schedule does not include the time to construct reduced MS-state diagrams for different function units. This is because, the generation of the offset sets can be done off-line, and stored in a database. Lastly, even though enhanced Co-scheduling uses the set of all offset sets, typically several hundreds in number, it still was successful in finding a schedule within a few milliseconds.

5.4 Comparison with Slack Scheduling

The enhanced Co-scheduling method is compared with our implementation of Huff's Slack Scheduling method[8]. The results of our comparison are presented in Table 4. The term percentage improvement for various quantities, such as **II**, is calculated for each loop as follows:

$$\text{\% Impr in } \mathbf{II} \text{ for Method A} = \frac{II_{Method\ B} - II_{Method\ A}}{II_{Method\ A}} * 100$$

The numbers reported in columns 4 and 7 in Tables 4 and 5 are averaged over all loops reported in columns 2 and 5 respectively.

As seen from Table 4 our enhanced Co-scheduling results in better **II** in 114 benchmarks; the average improvement in **II** is 13.5%. In a large number of cases (993 benchmarks) both methods achieved the same **II**. This is because, for the target architecture considered for the scheduling, only one-fourth (24%) of the loops are *resource-critical* — i.e., resource MII (**ResMII**) dominates recurrence MII

[8]To the best of our knowledge, our implementation of Huff's Slack Scheduling method faithfully follows the implementation details presented in [10].

Measure	Enhanced Co-Scheduling Better			Huff Better			Both Same	
	No. of Benchmarks	% Cases	% Improvement	No. of Benchmarks	% Cases	% Improvement	No. of Benchmarks	% Cases
II	114	10	13.5	40	3	1.6	993	87
Avg. Trials per instrn.	662	58	560.1	5	0.1	322.4	480	42
Avg. Ejections per instrn.	338	29	858.7	79	7	858.0	730	63
Exec. Time	988	86	457.9	158	14	427.6	1	0.0

Table 4: Comparison of Enhanced Co-Scheduling with Slack Scheduling

(**RecMII**). Since Co-scheduling is basically Slack scheduling, fine-tuned for better selection of offset values, it is not surprising that the improvement, in terms of **II**, happens only in resource-critical loops. Also, it is possible that in some of the resource-critical loops both methods have achieved the **MII**. In a small number of benchmarks, Huff's Slack Scheduling achieves a better **II**, though the percentage improvement is only minor (1.6%). This could be due to the order in which instructions are ejected in enhanced Co-scheduling.

The results shown in Table 4 exhibit a significant reduction in execution time (time to construct the schedules). In 988 of the benchmarks programs, the execution time of our Co-scheduling method is lower than that of Slack Scheduling. The average improvement in execution time is roughly by a factor of 4.5. As mentioned earlier, the execution time of our enhanced Co-scheduling method did not include the time to construct the MS-state diagram since these diagrams can be precomputed and stored in the compiler.

Apart from **II** and the time to construct schedule, we compare the two methods in terms of two other measures, namely (i) the average number of schedule slots tried (trials) per operation, and (ii) the average number of ejections per instruction. From Table 4, we observe that our enhanced Co-scheduling method performs better by a factor of 5 in 30% to 60% of the benchmarks. Lastly, even though Huff's Slack Scheduling achieves comparable improvement in performance, in terms of average ejections, trials, and scheduling time, the number of benchmarks where the improvement is observed is significantly lower (respectively 79, 5, and 158 benchmark programs).

5.5 Comparison with Co-Scheduling

Lastly, we compare the performance of the enhanced Co-scheduling method with the original method [8] in Table 5. Since the original Co-scheduling method considered only a single offset set, in more than 23% of benchmarks the enhanced method obtained lower **II**. The average improvement in **II** is 16.2%. Both methods show improvement, in terms of the average number of trials, average number of ejected operations, and execution time, in more or less equal number of benchmarks.

6 Related Work

Several software pipelining methods have been proposed in the literature [1, 4, 10, 12, 13, 18]. These methods are based on a global modulo reservation table. A comprehensive survey of these works is provided by Rau and Fisher in [17]. As explained in the Introduction, one major drawback of these methods is their inefficiency. These methods make several trials before successfully placing an operation in the modulo reservation table. They do not make effective use of the well-developed classical pipeline theory [3, 11, 15].

The approach proposed in [2, 14, 16] uses a finite state automaton (FSA)-based instruction scheduling technique. These methods use ideas from the classical pipeline theory, especially the notion of forbidden and permissible latency sequences. However these methods deal with general instruction scheduling, and do not handle software pipelining, where each scheduled instruction is initiated once every **II** cycles. Co-scheduling [8] is a state diagram-based software pipelining method. In the Co-scheduling method, a single permissible latency sequence chosen from the MS-state diagram and the corresponding offset values were used to guide the software pipelining method.

Lastly, Eichenberger and Davidson proposed a method, which still relies on the use of global resource table, but reduces the cost of structural-hazard checking by reducing the machine description [5]. Their approach uses forbidden latency information to obtain a minimal representation for reservation tables of different function units. Their method is applicable to both instruction scheduling and modulo scheduling.

7 Conclusions

In this paper we present an enhanced Co-scheduling method that makes use of the state diagram based approach

Measure	Enhanced Co-Scheduling Better			Original Co-Scheduling Better			Both Same	
	No. of Benchmarks	% Cases	% Improvement	No. of Benchmarks	% Cases	% Improment	No. of Benchmarks	% Cases
II	255	23	16.2	4	0.03	1.4	871	77
Avg. Ejections per Instrn.	88	8	526.5	149	13	525.2	893	79
Avg. Trials per Instrn.	144	13	24.6	97	9	24.7	889	79
Exec. Time	651	58	50.3	476	42	50.3	5	0

Table 5: Comparison of Enhanced Co-scheduling with Original Co-scheduling

for software pipelining. This work identifies and eliminates redundant information in the original MS-state diagram. Based on the reduced MS-state diagram, we extend the original Co-scheduling method by considering multiple latency sequences. Experimental results reveal that the reduced states results in significant reduction (2 to 3 orders of magnitudes) in the number of paths. The enhanced Co-scheduling was successful in constructing efficient schedules: they were better in terms of both **II** and the time to construct the schedule compared to the schedules constructed either by Huff's Slack scheduling method or by the original Co-scheduling method.

Acknowledgments

We wish to thank Sean Ryan, Chihong Zhang, members of CAPSL, University of Delaware, and the anonymous referees for their helpful suggestions. The last author acknowledges the support of National Science Foundation, U.S.A.

References

[1] E. R. Altman, R. Govindarajan, and G. R. Gao. Scheduling and mapping: Software pipelining in the presence of structural hazards. In *Proc. of the ACM SIGPLAN '95 Conf. on Programming Language Design and Implementation*, pages 139–150, La Jolla, CA, June 1995.

[2] V. Bala and N. Rubin. Efficient instruction scheduling using finite state automata. In *Proc. of the 28th Ann. Intl. Symp. on Microarchitecture*, pages 46–56, Ann Arbor, MI, Dec. 1995.

[3] E.S. Davidson, L.E. Shar, A.T. Thomas, and J.H. Patel. Effective control for pipelined computers. In *Digest of Papers, 15th IEEE Computer Society Intl. Conf., COMPCON Spring '75*. Feb. 1975.

[4] J. C. Dehnert and R. A. Towle. Compiling for Cydra 5. *Journal of Supercomputing*, 7:181–227, May 1993.

[5] A.E. Eichenberger and E.S. Davidson. A reduced multipipeline machine description that preserves scheduling constraints. In *Proc. of the ACM SIGPLAN '96 Conf. on Programming Language Design and Implementation*, Philadelphia, PA, May 1996.

[6] J. A. Fisher. Trace scheduling: A technique for global microcode compaction. *IEEE Transactions on Computers*, 7(30):478–490, July 1981.

[7] P. B. Gibbons and S. S. Muchnick. Efficient instruction scheduling for a pipelined architecture. In *Proc. of the SIGPLAN '86 Symp. on Compiler Construction*, pages 11–16, Palo Alto, CA, June 25–27, 1986.

[8] R. Govindarajan, E. R. Altman, and G. R. Gao. Co-scheduling hardware and software pipelines. In *Proc. of the Second Intl. Symp. on High-Performance Computer Architecture*, pages 52–61, San Jose, CA, Feb. 1996.

[9] R. Govindarajan, N.S.S. Narasimha Rao, E. R. Altman, and G. R. Gao. Software Pipelining using Reduced MS-State Diagrams. Technical Memo, Dept. of Computer Science & Automation, Indian Institute of Science, Bangalore, India, Feb. 1997. *(available via http://www.csa.iisc.ernet.in/∼govind/papers/TR-97-2.ps.gz)*

[10] R. A. Huff. Lifetime-sensitive modulo scheduling. In *Proc. of the ACM SIGPLAN '93 Conf. on Programming Language Design and Implementation*, pages 258–267, Albuquerque, NM, June 1993.

[11] P. M. Kogge. *The Architecture of Pipelined Computers*. McGraw-Hill Book Company, New York, New York, 1981.

[12] M. Lam. Software pipelining: An effective scheduling technique for VLIW machines. In *Proc. of the SIGPLAN '88 Conf. on Programming Language Design and Implementation*, pages 318–328, Atlanta, GA, June 1988.

[13] J. Llosa, M. Valero, E. Ayguadé, and A. González. Hypernode reduction modulo scheduling. In *Proc. of the 28th Ann. Intl. Symp. on Microarchitecture*, pages 350–360, Ann Arbor, MI, Dec. 1995.

[14] T. Muller. Employing finite state automata for resource scheduling. In *Proc. of the 26th Ann. Intl. Symp. on Microarchitecture*, Austin, TX, Dec. 1993.

[15] J. H. Patel and E. S. Davidson. Improving the throughput of a pipeline by insertion of delays. In *Proc. of the 3rd Ann. Symp. on Computer Architecture*, pages 159–164, Clearwater, FL, January 19–21, 1976.

[16] T. A. Proebsting and C. W. Fraser. Detecting pipeline structural hazards quickly. In *Conf. Rec. of the 21st ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages*, pages 280–286, Portland, OR, Jan. 1994.

[17] B. R. Rau and J. A. Fisher. Instruction-level parallel processing: History, overview and perspective. *Journal of Supercomputing*, 7:9–50, May 1993.

[18] B. R. Rau. Iterative modulo scheduling: An algorithm for software pipelining loops. In *Proc. of the 27th Ann. Intl. Symp. on Microarchitecture*, pages 63–74, San Jose, CA, Dec. 1994.