A Reduced Multipipeline Machine Description that Preserves Scheduling Constraints[*]

Alexandre E. Eichenberger and Edward S. Davidson

Advanced Computer Architecture Laboratory
EECS Department, University of Michigan
1301 Beal Ave, Ann Arbor, MI 48109-2122
alexe,davidson@eecs.umich.edu
(313) 936-2917

Abstract

High performance compilers increasingly rely on accurate modeling of the machine resources to efficiently exploit the instruction level parallelism of an application. In this paper, we propose a reduced machine description that results in faster detection of resource contentions while preserving the scheduling constraints present in the original machine description. The proposed approach reduces a machine description in an automated, error-free, and efficient fashion. Moreover, it fully supports schedulers that backtrack and process operations in arbitrary order. Reduced descriptions for the DEC Alpha 21064, MIPS R3000/R3010, and Cydra 5 result in 4 to 7 times faster detection of resource contentions and require 22 to 90% of the memory storage used by the original machine descriptions.

1 Introduction

Current compilers for VLIW and superscalar machines focus on exploiting more of the inherent parallelism in an application in order to obtain higher performance. Fine grain schedulers are a critical element in efficiently exploiting instruction level parallelism and a significant body of research has sought more effective scheduling algorithms. Several new directions have been explored: schedulers may not schedule operations in cycle order, focusing initially on operations along critical paths [1][2][3][4][5][6], they may backtrack to reverse poor scheduling decisions [1][2][3][4][7], and they may hide long latencies by speculating operations across branches and basic blocks [1][6][7][8][9][10].

High performance compilers have also used precisely detailed machine models [1][3][7][11][12][13] to better utilize the machine resources of current processors with increasingly wider issue mechanisms, deeper pipelines, and more heterogeneous functional units. Precise modeling of machine resources is critical to avoid resource contentions that may stall some of the pipelines or, in the absence of hardware interlocks, corrupt some of the results. Resource modeling has to cope with rapidly changing processor models while controlling development cost by reusing existing compiler technology.

To meet these challenges, compilers have increasingly relied on a resource modeling utility, separated from the rest of the compiler, that can quickly answer the following query: "Given a target machine and a

[*]Appeared as Technical Report CSE-TR-266-95, University of Michigan, Ann Arbor, MI.

partial schedule, can I place this additional operation in this cycle without resource contention?" Typically, this functionality has been provided by a *contention query module* that processes the machine description of a target machine, generates an internal representation of the resource requirements, and provides for a querying mechanism [1][3][7][11][12][13]. The IMPACT compiler, for example, implemented such a module [12] to produce high performance schedules for a wide range of machines, from existing architectures such as X86, PA-RISC, and Sparc to research architectures such as PlayDoh [14].

With the recent emphasis on exploiting instruction level parallelism, compile time is increasingly spent in the contention query module as several cycles of a schedule, possibly in several basic blocks [9][10], are queried per operation in order to achieve good schedules. Optimizing contention query modules therefore has a significant impact on the overall performance of a compiler, as queries are issued in the innermost loop of the scheduler. This optimizing issue has recently been addressed in several papers [15][16][17], but these either over restrict the manner in which operations are placed or approximate the resource requirements of a schedule.

In this paper, we propose a reduced machine description that results in significantly faster detection of resource contentions while exactly preserving the scheduling constraints present in the original machine description. The reduced machine description is expressed using reservation tables that determine the resource usage for each operation. We demonstrate how to derive a reduced machine description for a given target machine and present several examples illustrating the effectiveness of our approach.

The proposed approach fully supports *unrestricted scheduling models* where operations can be scheduled in arbitrary order and prior scheduling decisions can be reversed. Unrestricted scheduling is essential to accommodate the elaborate scheduling techniques used by today's high performance compilers. The Cydra 5 compiler, for example, uses an operation-driven scheduler that reduces the schedule length of a basic block by scheduling operations along the critical path first [1]. Operation-driven schedulers consider operations in topological order, not in order of monotonically increasing (or decreasing) schedule time. Also, both the Cydra 5 and the IMPACT compilers use software pipelining techniques to achieve loop schedules with high throughput [1][2]. Software pipelining schedulers do not consider operations in topological order as, in general, no topological order is defined in dependence graphs with loop-carried dependences. Moreover, experimental results indicate that software pipelined loops can achieve higher throughput in less compilation time when some limited number of scheduling decisions can be reversed, as shown by Rau [3], and used in numerous compilers [1][2][3][4][18]. The Multifow compiler also uses a backtracking mechanism to improve scalar code schedules [7].

The proposed approach also precisely handles *basic block boundary conditions*, i.e. the dangling resource requirements from predecessor basic blocks. In general, the resource requirements at the beginning of a basic block consists of the union of all the resource requirements dangling from predecessor basic blocks. Handling boundary conditions is even more important for high performance compilers that hide operation latencies by (speculatively) moving operations across branches and basic blocks [1][6][7][8][9][10]. Both the Cydra 5 and the Multifow compilers, for example, use scheduling algorithms that handle dangling resource requirements [1][7].

Currently, most compilers rely on machine descriptions that have been manually reduced using ad-hoc methods. This process is error prone and, to avoid errors or reduce the machine description, conservative assumptions may be employed. Thus, the reduced machine description may prohibit certain operation sequences that cause no contentions on the target machine. Furthermore, high performance compilers are often developed in parallel with micro-architecture development during which resource requirements often change. Manually reducing the machine description must then be carried out several times, introducing more potential for errors, reduced optimization, and increased maintenance cost. Using our approach, the resource requirements can be expressed in terms close to the actual hardware structure of the target machine and the reduced machine description used by the compiler is generated in an error-free and automated fashion.

Experiments with the DEC Alpha 21064 [19], the MIPS R3000/R3010 [20], the and Cydra 5 [21] machines indicate 4 to 7 times faster contention queries and require 22 to 90% of the memory storage used by the original machine descriptions. These improvements are obtained by using highly reduced machine descriptions instead of the original or manually optimized machine descriptions. Using the operation frequency found in a benchmark suite of 1327 loops from the Perfect Club, SPEC-89, and the Livermore Fortran Kernels compiled for the Cydra 5, and modeling all operations in that benchmark suite, our results indicate that the average contention query is performed by testing only 1.03 words, where a single 64 bit word encodes the state of 7 consecutive schedule cycles.

In this paper, we present related work in Section 2 and an introductory example in Section 3. Algorithms to construct reduced machines are developed in Sections 4 and 5. Reduced machine and benchmark examples are presented in Section 6 and conclusions in Section 7.

2 Related Work

Resource contention in multipipeline scheduling may be based directly on reservation tables, or on the forbidden latency sets or contention-recognizing state machines derived from them, as introduced by Davidson *et al* [22]. Traditionally, reservation tables contain much redundant information that consumes memory and increases query response time. As a result, recent advances favor finite-state automata approaches. In this paper, however, we propose a reduced reservation table approach that eliminates much of the redundancy and does not suffer from the weaknesses of the automata approaches, as detailed below.

Proebsting and Frazer [15] as well as Müller [16] proposed a contention query module using a finite state automaton that recognizes all contention-free schedules. This approach was recently extended for unrestricted scheduling models by Bala and Rubin using a forward and reverse pair of automata [17]. In their approach, operations considered in order of monotonically increasing (or decreasing) schedule time are quickly scheduled using a forward automaton. Additional operations are then inserted in the schedule in cycles recognized as contention-free by the forward and reverse automata. Because an inserted operation introduces additional resource requirements, these additional requirements must be propagated in adjacent cycles, i.e. the state of scheduled operations in adjacent cycles must be updated in both the forward and reverse automata. Their approach also addresses the handling of basic block boundary conditions at the cost of introducing up to O(s^2) new states in the automata, where s is the number of cycle-advancing states in the original automata [17].

The principal advantage of automaton-based approaches is that the next contention-free cycle can be determined in a single table lookup. A potential problem of this approach, however, is the size of these automata, especially when up to O(s^2) additional states are introduced to handle basic block boundary conditions. This issue is addressed in the literature in three ways. First, operations of a target machine can be combined into classes of operations that have compatible resource contentions [15]. Second, large automata can be factored into sets of smaller ones [16][17], reducing the size of the automata, but increasing the number of table lookups necessary to process a contention query. Third, the number of additional states introduced in the automata to handle boundary conditions can be reduced [17], at the cost of making conservative approximations, potentially resulting in schedules with lower performance.

Another problem arises when supporting unrestricted scheduling models, as the state of the forward and reverse automaton must be saved after each scheduled operation. As a result, two states per operation must be stored in addition to the two automata [17], which may result in a large memory overhead per cycle of the schedule, especially for wide-issue machines. Supporting unrestricted scheduling models also requires the consistency of the stored state to be maintained when scheduling additional operations [17], as inserted operations introduce additional resource requirements. Thus, handling unrestricted scheduling models introduces both memory and computation overhead.

3 Reducing a Machine Description

In this section, we illustrate the three-step process of constructing a synthesized machine, resulting in reduced numbers of resources and resource usages while exactly preserving the scheduling constraints due to resource contentions in the target machine.

We begin with a given machine description consisting of a set of *reservation tables*, one per operation, that expresses the resource requirements of each operation in terms close to the actual hardware structure of the target machine. The rows of a reservation table correspond to distinct resources of the target machine and its columns correspond to the cycles in which resources are used, relative to the issue time of the corresponding operation. An X entry in row i/column j is made in the reservation table associated with operation X if there is a *usage* of resource i in cycle j by operation X, i.e. if resource i is reserved for exclusive use during cycle j by operation X. In this paper, we restrict our focus to machines with a single copy of each distinct resource. This restriction does not preclude machines with replicated resources; it merely requires that the replicated copies be distinguished in the reservation tables.

Figure 1a illustrates the reservation tables of a hypothetical machine with 2 operations (A and B) and 5 distinct resources $(0, \ldots, 4)$. Operation A is representative of the resource requirements of a fully pipelined functional unit. Operation B is representative the resource requirements of a partially-pipelined functional unit, where resource q_3 may correspond to a multiply stage used for 4 consecutive cycles and resource q_4 may correspond to a rounding mode stage used for 2 consecutive cycles. Although this hypothetical machine was constructed to concisely illustrate our methodology, it is representative of some of the resource usage patterns found in our benchmark examples (see Figure 4 for example).

Step 1: We extract from the reservation tables of the target machine the set of *forbidden latencies*, i.e. the set of initiation intervals for which a resource contention occurs between two operations. Visually, the set of forbidden latencies between operations X and Y is obtained by overlapping their reservation tables, and searching for all initiation intervals that result in simultaneous use of one or more shared resources.

To formalize the definition of forbidden latencies, we define the *usage set* X_i as the set of cycles in which operation X reserves resource i for exclusive use. Figure 1a illustrates the usage sets of our example machine. For example, usage set B_3 is equal to $\{2, 3, 4, 5\}$, as operation B uses resource 3 in cycle 2, 3, 4,

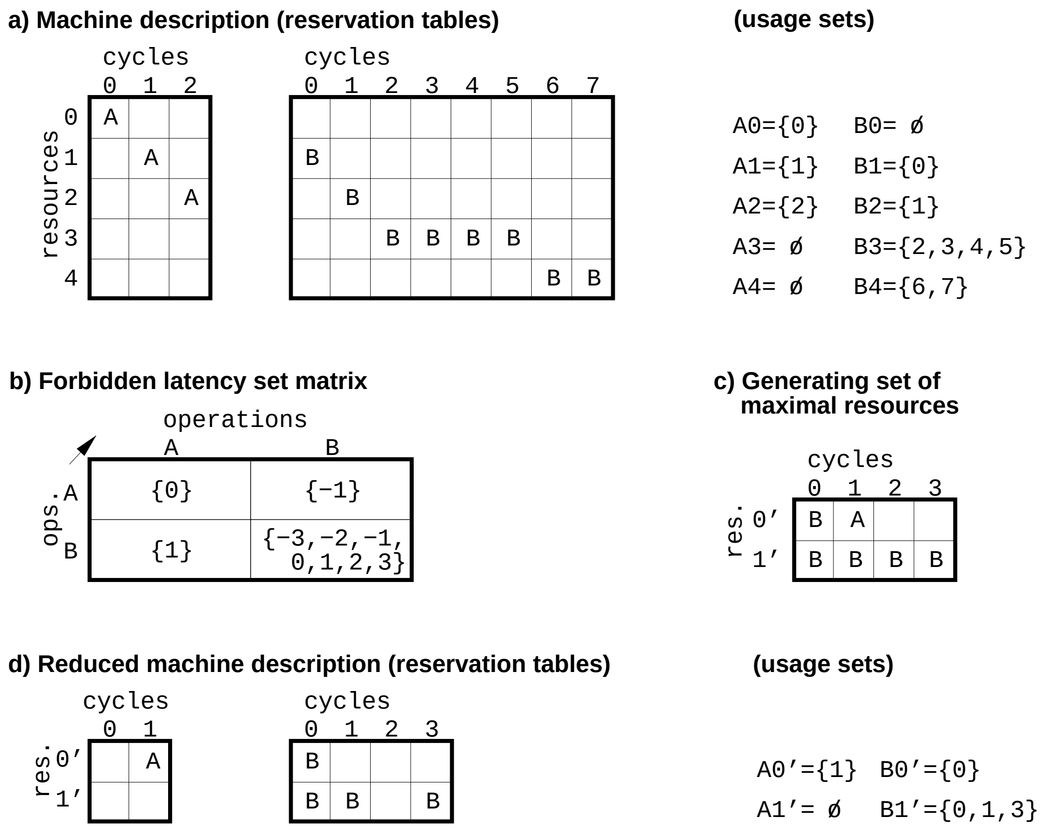


Figure 1: Reducing a machine description.

and 5. Consider the usages $x \in X_i$ and $y \in Y_i$ of resource i by operation X and Y, scheduled at time t_X and t_Y, respectively. A resource contention occurs when both operations use the shared resource simultaneously, i.e. when $t_X + x = t_Y + y$. Consequently, we know that operation X cannot be scheduled $(y - x)$ cycles after operation Y. Generalizing this relation to all resources shared by operations X and Y, we obtain $F_{X,Y} = \{f \mid \text{operation } X \text{ cannot be scheduled } f \text{ cycles after operation } Y\}$, i.e.

$$F_{X,Y} \quad = \quad \{(y - x) \mid \text{for all } i \in Q,\ x \in X_i,\ y \in Y_i\ \} \tag{1}$$

Equation (1) defines a matrix of forbidden latency sets for all pairs of operations, where $F_{X,Y}$ is the set in row X, column Y of the matrix. Figure 1b illustrates this matrix computed for our example machine. While these sets are computed for each operation of the target machine, we need list these sets only for each operation class, as presented by Proebsting and Fraser [15], where two operations belong to the same *operation class* if they have compatible[1] sets of forbidden latencies. Note two properties of the forbidden latency matrix. First, if operation X uses any resources, it necessarily conflicts with itself for an initiation interval of 0 (i.e. $0 \in F_{X,X}$). Second, operation X cannot be scheduled f cycles *after* operation Y if and only if Y cannot be scheduled f cycles *before* X (i.e. $f \in F_{X,Y} \Leftrightarrow -f \in F_{Y,X}$).

Formal Problem Definition: Generate a reduced machine description that precisely produces the forbidden latencies of the target machine with a reduced number of synthesized resources and resource usages.

[1] Operations X and Y belong to the same class if $F_{X,Z} = F_{Y,Z}$ and $F_{Z,X} = F_{Z,Y}$ for each operation Z of the target machine.

One of several objective functions (e.g. the number of resources or resource usages) may be minimized, depending on the desired internal representation. Querying for resource contentions using either the original or reduced machine descriptions yields the same answer, as both descriptions enforces the same forbidden latencies.

Step 2: We build the *generating set of maximal resources* which contains all maximal resources associated with the target machine [23]. A *maximal resource* is defined as a synthesized resource such that (a) every forbidden latency generated by this resource is forbidden in the target machine and (b) no additional usage by any operation can be added to this resource without generating a forbidden latency that is not forbidden in the target machine. There are two maximal resources of our example machine, shown in Figure 1c. The first resource, resource $0'$, is a maximal resource that generates $1 \in F_{B,A}$; it also includes forbidden latencies: $0 \in F_{A,A}$, $0 \in F_{B,B}$, and $-1 \in F_{A,B}$. Note that no other usages of A or B can be added to resource $0'$, as they would necessarily introduce forbidden latencies not present in the forbidden latency matrix of our example machine. Similarly, the second maximal resource, resource $1'$, generates the remaining forbidden latencies, i.e. $\{0, \pm 1, \pm 2, \pm 3\} \in F_{B,B}$, and no other usages can be added. Since resources $0'$ and $1'$ cover all forbidden latencies of our example machine, we know that there are no other maximal resources.

The maximal resources are interesting for several reasons. First, any reservation table that generates the same forbidden latency matrix can be constructed from subsets of maximal resources, possibly translated by some number of cycles. Second, no additional forbidden latency can be added to the set of forbidden latencies generated by a maximum resource because it would otherwise violate part (a) of the maximum resource definition. As a result, we can use a subset of the maximal resources to cover all the forbidden latencies of a target machine with the fewest number of synthesized resources. Third, the maximal resources have the largest feasible number of usages per resources. This property can be exploited to compute a lower bound on the minimum schedule length of a scalar code or an upper bound on the maximal throughput of a modulo schedule.

Step 3: We select a subset of the maximal resources and their resource usages which covers all the forbidden latencies in the forbidden latency matrix. The selection heuristic minimizes an objective function that varies as a function of the desired internal representation. For our example machine, if the objective is to minimize the number of synthesized resources, we must select both resources $0'$ and $1'$, as resource $0'$ is the only resource covering $0 \in F_{A,A}$ and resource $1'$ is the only resource covering $3 \in F_{B,B}$. However, if the objective function is to minimize the number of resource usages, we may also remove the second or third usage of B in resource $1'$, shown Figure 1c, as the three remaining usages of B are sufficient to generate all the forbidden latencies in $F_{B,B}$.

Comparing Figure 1a to Figure 1d, we can appreciate the benefit of reducing the reservation tables of a target machine. First, the reduced machine description reduces the number of resources from 5 to 2, thus potentially decreasing the memory requirements needed to store the reserved resources of a schedule. Second, the number of resource usages decreases from 3 to 1 for operation A, and from 8 to 4 for operation B. If detecting resource contentions is linear in the number of usages, the reduced machine description results in significantly faster queries. We will refine this view in Section 5.

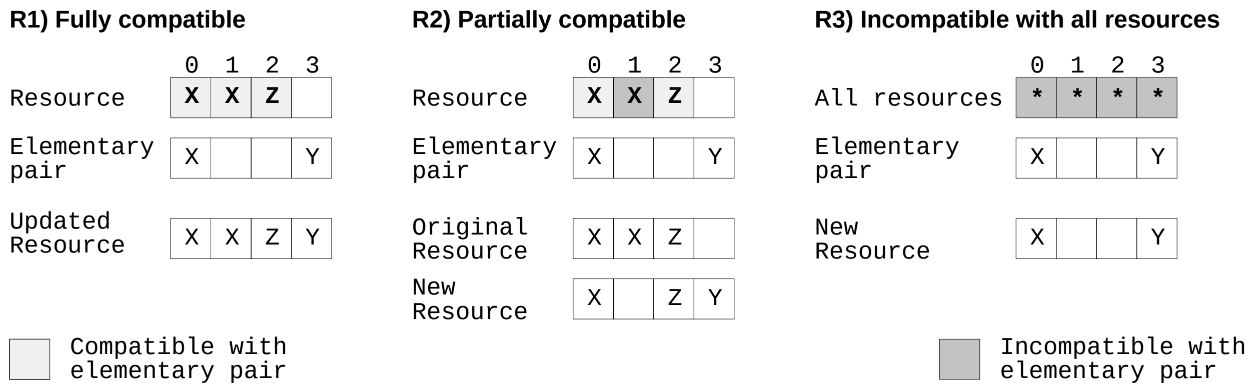


Figure 2: Three situations when adding an elementary pair to a resource (Rules 1, 2, and 3).

4 Building Generating Sets

In this section, we present an algorithm that constructs the generating set of maximal resources, a set that contains all the maximal resources of a target machine. The algorithm builds the maximal resources incrementally, adding usages to resources and creating new resources when appropriate. It is an efficient algorithm that does not backtrack; however, it may produce sub-maximal resources in addition to all maximal resources. A mechanism to remove sub-maximal resources as well as redundant maximal resources will be discussed in Section 5.

This algorithm handles each forbidden latency in turn, in order of increasing forbidden latency. Note that we consider only the nonnegative forbidden latencies of the forbidden latency matrix, as positive and negative latencies are redundant ($f \in F_{X,Y} \Leftrightarrow -f \in F_{Y,X}$). The zero self-contention latencies (i.e. $0 \in F_{X,X}$) are handled as a special case, after completion of our algorithm.[2]

When handling a forbidden latency, $f \in F_{X,Y}$, the algorithm attempts to add that forbidden latency to each of the resources currently in the generating set. Since resources deal with usages rather than with forbidden latencies, the first step is to convert a forbidden latency to an elementary pair of usages. The *elementary pair* associated with the forbidden latency f is defined as a usage by operation X in cycle 0 and a usage by operation Y in cycle f. We make an ordered list of elementary pairs and start with an empty current generating set.

The second step attempts to add the first elementary pair on the current list to each of the resources of the current generating set. Three situations may occur when adding an elementary pair p to a resource q:

R1: In the first situation, elementary pair p is *fully compatible* with resource q, i.e. the forbidden latencies generated by each pair of usages in p and q do exist in the forbidden latency matrix. Since elementary pair p and resource q are fully compatible, we add the usages of p to resource q. This process is referred to as *Rule 1*, and is illustrated in Figure 2-R1 for the elementary pair associated with the forbidden latency $3 \in F_{X,Y}$.

R2: In the second situation, elementary pair p is *partially compatible* with resource q, i.e. the usages of p and some, but not all, usages of q generate forbidden latencies that are present in the forbidden

[2]After completion of our algorithm, we simply create additional resources with a single usage in cycle 0 for each operation X with $0 \in F_{X,X}$ as unique forbidden latency.

latency matrix. Since elementary pair p is not fully compatible with resource q, we may not simply add the usages of p to resource q, as otherwise forbidden latencies not present in the forbidden latency matrix would be introduced by a resource in the current generating set. Instead, we leave resource q unchanged and create a new resource in the current generating set, a resource that includes the usages of elementary pair p and each of the usages of resource q that are compatible with p. This process is referred to as *Rule 2*, and is depicted in Figure 2-R2.

R3: In the third situation, elementary pair p is *incompatible* with every usage in resource q. No action is performed unless elementary pair p is incompatible with every usage of every resource in the current generating set. In that case, a new resource is created with the usages of elementary pair p. This process is referred to as *Rule 3*, and is illustrated in Figure 2-R3.

Theorem 1 (Generating Set of Maximal Resources) *Building the generating set of maximal resources by applying Rules 1, 2 and 3 to each resource in the current generating set for each elementary pair (in any order) produces resources that forbid only those latencies that are forbidden in the target machine. Furthermore, the final generating set includes all maximal resources of the target machine.*

Proof: Rules 1, 2, and 3 never place a usage in a resource unless it is compatible with each other usage in that resource, i.e. no resource in the current (and hence final) generating set forbids any latency not forbidden in the target machine.

We prove the second part of Theorem 1 by contradiction. Suppose there is a maximal resource not in the generating set. We shift its resource usages so that its earliest usage occurs in cycle 0 and call that maximum resource q. Let q have n usages, u_1 to u_n, with u_1 in cycle 0. We refer to the forbidden latencies generated by $u1$ with each other $n-1$ usages as f_2 to f_n. Without loss of generality, we may assume that the forbidden latencies f_2 to f_n are numbered in the order in which they are processed by our algorithm.

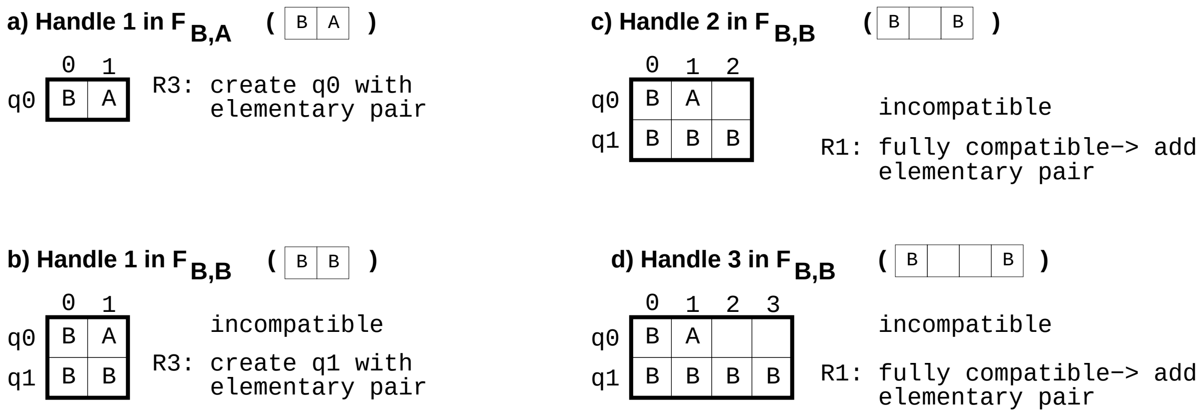
After forbidden latency f_2 is processed (by Rules 1, 2, and 3), its corresponding elementary pair (u_1, u_2) is present in at least one resource, q_{12}, of the current generating set. Other forbidden latencies are then processed, possibly adding usages to q_{12}, but never removing any.

Eventually, the algorithm will process forbidden latency f_3. From resource q, we know that f_2 and f_3 are compatible. Hence Rules 1 and 2 will result in a resource q_{123} containing usages u_1, u_2, u_3, and possibly others; Rule 3 will not apply. Repeating this processes with all the remaining forbidden latencies of the target machine, including f_4 through f_n, we obtain resource $q_{12\ldots n}$ containing all usages u_1 through u_n, and possibly others. Thus resource q is either not maximal or is included in the final generating set, contradicting the initial assumption. $\Box$

Figure 3 illustrates the algorithm, step by step, for our example machine. The algorithm handles the four nonnegative non-zero self-contention forbidden latencies $1 \in F_{B,A}$, $1 \in F_{B,B}$, $2 \in F_{B,B}$, and $3 \in F_{B,B}$, in that order. The generating sets are shown at each step, in Figures 3a, 3b, 3c, and 3d, respectively. The rule applied to each resource is also indicated to the right of each resource.

5 Selecting Synthesized Resources and Resources Usages

Once the generating set of maximal resources has been computed, we select a subset of these resources and their usages that covers all the forbidden latencies in the forbidden latency matrix. The selection heuristic



a) Handle 1 in $F_{B,A}$ (B A)
0 1
q0 B A
R3: create q0 with elementary pair
b) Handle 1 in $F_{B,B}$ (B B)
0 1
q0 B A
q1 B B
incompatible
R3: create q1 with elementary pair
c) Handle 2 in $F_{B,B}$ (B B)
0 1 2
q0 B A
q1 B B B
incompatible
R1: fully compatible-> add elementary pair
d) Handle 3 in $F_{B,B}$ (B B)
0 1 2 3
q0 B A
q1 B B B B
incompatible
R1: fully compatible-> add elementary pair

Figure 3: Building the generating set for our example machine.

attempts to minimize an objective function that varies as a function of the desired internal representation for partial schedules. In this paper, we consider the two following internal representations.

Discrete-Representation: This representation uses a *reserved table* with one row per resource and one column per schedule cycle. Each entry contains a flag indicating whether the corresponding resource has been reserved by an operation in the current partial schedule. Entries may contain additional fields, such as the predicate under which a resource is reserved, as proposed in the Enhanced Modulo Scheduling scheme [2]. The contention query module checks whether operation X can be scheduled in cycle j by checking each usage in the reduced reservation table for X (offset by j) against the corresponding entry in the reserved table. Because the number of entries tested is proportional to the number of resource usages overall reduced reservation tables, the primary objective of the selection heuristic is to reduce the number of resource usages in the reduced machine description. This objective function is referred to as *min-usages*.

Bitvector-representation: This representation extracts the flag bits of the discrete representation and packs them into one bitvector per schedule cycle (and reduced reservation tables are represented likewise). Checking for one query now amounts to simply "anding" each nonempty bitvector in the reduced reservation table with the corresponding column bitvector in the reserved table and checking for a 0 result.

If k bitvectors can be packed per word, the number of "and" operations per query and the memory requirements for storing the reserved table are reduced. However, we need to store or generate k representations of each reduced reservation table, word aligned on k successive cycles. The selection heuristic must now reduce the number of words that need to be tested, i.e. the number of nonempty groups of k consecutive cycles in the reduced reservation tables. A secondary objective is to *maximize* the numbers of resource usages in these nonempty words, as more resource usages per word permit faster (early out) detection of resource contentions. This objective function is referred to as *min-k-cycles*.

Selection Heuristic: Although integer programming can solve these minimum cover problems, we have found a fast and effective heuristic. First, we prune the resources of the generating set by successively removing each resource that produces a set of forbidden latencies that is generated or covered by a remaining resource. Second, for each nonnegative forbidden latency, we locate all usage pairs that generate it in the pruned generating set. Third, we choose one of the forbidden latencies with the shortest list of usage pairs.

The heuristic selects from the list the usage pair that covers the largest number of forbidden latencies not yet covered by currently selected resource usages. In case of ties, the heuristic selects the usage pair whose newly covered forbidden latencies have a larger sum. Once a usage pair is selected, the pair of usages and the corresponding resource are marked as selected. When using the bitvector-representation, the heuristic also marks every other usage of marked resources within the same word. The selection heuristic then proceeds with the next forbidden latency until the marked resource usages cover every forbidden latency in the target machine.

6 Reduced Machine and Benchmark Examples

In this section, we present experimental results for three machines, the DEC Alpha 21064, the MIPS R3000/R3010, and the Cydra 5. For each machine and internal representation, we present three data points. First, we present the total number of resources in the machine description. Second, we present the average number of resource usages per operation type in the machine description. For all but one benchmark, we assume that each operation types has the same frequency, a pessimistic assumption because complex operations are usually less frequent than simple operations. Third, we present the number of words of bitvectors that need to be tested to answer a query i.e. the number of nonempty groups of k consecutive cycles. This number, referred to as *word usages*, is averaged over all operation types and possible alignments.

As a proof of concept, we investigated our technique on the Cydra 5 machine [21] which has the most complex resource requirements of the three machines. The machine configuration investigated has 7 functional units: 2 memory port units, 2 address generation units, 1 FP adder unit, 1 FP multiplier unit, and 1 branch unit. The original machine description used by the Cydra 5 Fortran77 compiler models 56 resources and 152 distinct patterns of resource usages [1]. Note that the original machine description was already manually optimized, i.e. some of the physical resources of the Cydra 5 were eliminated from the machine description as they did not introduce any new forbidden latencies [24]. The Cydra 5 machine description results in 52 distinct operation types and 10223 forbidden latencies. Compared to the most complex machine description handled in papers on finite state automaton [15], the Cydra 5 has 3.5 times more operation types and 2.4 times more forbidden latencies. For each internal representation, our algorithm reduces the Cydra 5 machine description in less than 11 minutes on a Sparc-20.

Representation:	original	discrete	bitvectors	(32 bits)	(64 bits)
Objective function:	–	*min-uses*	*min-1-cycles*	*min-2-cycles*	*min-4-cycles*
tot. resources	56	15	15	15	15
avg. resource usages/ops.	18.2	8.3	8.8	10.1	11.4
avg. word usages/ops.	13.2	6.7	6.2	4.7	3.3

Table 1: Results for the Cydra 5: 52 operation types, 10223 forbidden latencies (all ≤ 41).

Table 1 presents data for 4 reduced machine descriptions of the Cydra 5 machine and the original description used by the Cydra 5 Fortran77 compiler[1]. The second column corresponds to a reduced discrete-representation machine description that attempts to minimize *min-uses*. The three remaining columns correspond to reduced bitvector-representation machine descriptions that attempts to minimize *min-k-cycles* and maximize *min-uses*. Underlined numbers correspond to the entries minimized by the respective objective functions.

Compared to the original machine description, the results for the discrete representation indicate a decrease in the average number of resource usages by a factor of 2.2 (from 18.2 to 8.3 resource usages) and a decrease in discrete entries by a factor of 3.7 (from 56 to 15 resources). Similarly, the results for the 64 bit word bitvector representation indicate a decrease in the average number of words tested by a factor of 4.0 (from 13.2 to 3.3 words) and require only 25% of the storage used by the original machine description to store the reserved tables (from 1 to 4 cycles per word). Note the successive increases in the number of resource usages for the three reduced bitvector representations. These increases permit faster detection of resource contentions and do not increase memory space for state storage.

Representation:	original	discrete	bitvectors	(32 bits)	(64 bits)
Objective function:	–	*min-uses*	*min-1-cycles*	*min-4-cycles*	*min-9-cycles*
tot. resources	8	7	7	7	7
avg. resource usages/ops.	12.8	5.8	5.9	8.1	10.9
avg. word usages/ops.	11.6	5.0	4.8	2.9	2.0

Table 2: Results for the DEC Alpha 21064: 12 operation types, 293 forbidden latencies (all ≤ 58).

Table 2 shows the results of our technique for the DEC Alpha 21064 [19], using the machine description in Bala and Rubin [17]. Comparing the original description to the 64 bit word bitvector representation, the reduced machine description decreases the average number of words to test by a factor of 5.8 (from 11.6 to 2.0 words). This reduced representation can detect all resource contentions by testing on average two 64 bit words even though the largest forbidden latency is 58 cycles. Bala and Rubin presented a factored finite state automaton for this processor with 469 states ignoring boundary conditions and with 3666 states when boundary conditions with forbidden latencies dangling by up to 7 cycles are precisely modeled and approximated otherwise (between 8 and 58 cycles) [17]. Encoding the 3666 states in 12 bits, the state automata requires 4x12 bits of memory per schedule cycle, when caching the states of the forward and reverse automata for this dual issue microprocessor, compared to 7 bits per schedule cycle for bitvector representation.

Representation:	original	discrete	bitvectors	(32 bits)	(64 bits)
Objective function:	–	*min-uses*	*min-1-cycles*	*min-4-cycles*	*min-9-cycles*
tot. resources	22	7	7	7	7
avg. resource usages/ops.	17.3	7.3	8.1	8.3	8.5
avg. word usages/ops.	11.0	5.6	5.6	2.4	1.6

Table 3: Results for the MIPS R3000/R3010: 15 operation types, 428 forbidden latencies (all ≤ 34).

Table 3 shows the results of our technique for the MIPS R3000/R3010 [20], using the machine description in Proebsting and Frazer [15]. Comparing the original description to the 64 bit word bitvector representation, the reduced machine description decreases the average number of words to test by a factor of 6.9 (from 11.0 to 1.6 words). Proebsting and Frazer reported for this processor a finite state automaton of 6175 states, an automaton that does not handle the boundary conditions [15].

Table 4 presents data for a subset of Cydra 5, namely the 12 operation types actually used in a benchmark

Representation:	original	discrete	bitvectors	(32 bits)	(64 bits)
Objective function:	–	*min-uses*	*min-1-cycles*	*min-3-cycles*	*min-7-cycles*
tot. resources	39	9	9	9	9
avg. resource usages/ops.	9.4 (5.03)	2.9 (1.27)	2.9 (1.27)	3.6 (1.32)	4.2 (1.35)
avg. word usages/ops.	7.5 (4.02)	2.6 (1.09)	2.6 (1.09)	2.0 (1.06)	1.5 (1.03)

Table 4: Results for a subset of the Cydra 5: 12 operation types, 166 forbidden latencies (all ≤ 21).

suite of 1327 loops obtained from the Perfect Club, SPEC-89, and the Livermore Fortran Kernels [3][25][26]. The numbers in parenthesis correspond to the average number of usages, weighted by the relative frequency of each operation type in the benchmark suite. We see that the weighted average number of resource usages for the discrete representation is as low as 1.27, 4.0 times better than the original machine description. Similarly the weighted average number of words to test for bitvector representation is as low as 1.03, 3.9 times better than the original machine description. The reservation tables associated with the machine descriptions of the original model, the discrete representation, and the 64 bit word bitvector representation are shown in Figures 4a, 4b, and 4c, respectively.

We will report the dynamic number of usages and words tested for this benchmark in the final submission.

7 Conclusions

In this paper, we present an efficient contention query module that supports the elaborate scheduling techniques used by today's high performance compilers. In particular, we support unrestricted scheduling models, where the operation currently being scheduled may be placed before some already scheduled operations and backtracking is performed to produced highly optimized software-pipelined and critical-path sensitive schedules. We also support precise boundary conditions where resource requirements may dangle from predecessor basic blocks to permit effective latency-hiding techniques.

Our contention query module is based on a reduced machine description that results in significantly faster detection of resource contentions while exactly preserving the scheduling constraints present in the original machine description. This approach achieves two goals. First, it handles queries significantly faster which is increasingly important as queries for contentions are made in the innermost loop of a scheduler. Second, it reduces machine descriptions in an error-free and automated fashion, thus, simplifying the interface between the actual hardware structure of the target machine and the compiler representation of the scheduling constraints due to resource contentions.

Experiments with three machine descriptions indicate that our approach addresses the perceived weakness of resource modeling approaches based on reservation tables. Because the machine descriptions are reduced, all resource contentions are detected on average by 1.6 (MIPS R3000/R3010), 2.0 (DEC Alpha 21064), and 3.3 (Cydra 5) "and" operations when using the 64 bit word bitvector representation. Results taking into account the relative frequency of each operation type indicate that the number of "and" operations is even lower, as simpler operations are more frequent than more complex ones. Moreover, the memory requirements needed to store the reserved resources of a schedule are small, as a 64 bit word may encode the bitvector of 4 (Cydra 5), 9 (MIPS R3000/R3010), or 9 (DEC Alpha 21064) schedule cycles.

Acknowledgements

This work was supported in part by the Office of Naval Research under grant number N00014-93-1-0163 and by Hewlett-Packard. The authors would like to thank Santosh Abraham, Scott Mahlke, Bob Rau, and Michael Schlansker for many useful suggestions and for providing the Cydra 5 machine model. The help of Todd Proebsting for explaining the MIPS R3000/R3010 machine model and Vasanth Bala for explaining the Alpha 21064 machine model is also acknowledged.

References

[1] J. C. Dehnert and R. A. Towle. Compiling for the Cydra 5. In *The Journal of Supercomputing*, volume 7, pages 181–227, 1993.

[2] N. J. Warter, G. E. Haab, and J. W. Bockhaus. Enhanced Modulo Scheduling for loops with conditional branches. *Proceedings of the 25th Annual International Symposium on Microarchitecture*, pages 170–179, December 1992.

[3] B. R. Rau. Iterative Modulo Scheduling: An algorithm for software pipelining loops. *Proceedings of the 27th Annual International Symposium on Microarchitecture*, pages 63–74, November 1994.

[4] R. A. Huff. Lifetime-sensitive modulo scheduling. *Proceedings of the ACM SIGPLAN'93 Conference on Programming Language Design and Implementation*, pages 258–267, June 1993.

[5] J. R. Goodman and W.-C. Hsu. Code scheduling and register allocation in large basic blocs. *Proceedings of the International Conference on Supercomputing*, pages 442–452, 1988.

[6] K. Ebcioglu, R. D. Groves, K.-C. Kim, G. M. Silberman, and I. Ziv. Vliw compilation techniques in a superscalar environment. In *Proceedings of the ACM SIGPLAN'94 Conference on Programming Language Design and Implementation*, pages 36–48. 1994.

[7] G. P. Lowney et al. The multiflow trace scheduling compiler. In *The Journal of Supercomputing*, volume 7, pages 51–142, 1993.

[8] P. P. Chang, N. J. Warter, S. A. Mahlke, W. Y. Chen, and W. W. Hwu. Three architectural models for compiler-controlled speculative execution. *IEEE Transactions on Computers*, 44(4):481–494, April 1995.

[9] D. Bernstein and M. Rodeh. Global instruction scheduling for superscalar machines. In *Proceedings of the ACM SIGPLAN'91 Conference on Programming Language Design and Implementation*, pages 241–255, June 1991.

[10] S.-M. Moon and K. Ebcioglu. An efficient resource-constrained global scheduling technique for superscalar and vliw processors. *Proceedings of the 25th Annual International Symposium on Microarchitecture*, pages 55–71, September 1992.

[11] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Warter, and W. W. Hwu. Impact: An architectural framework for multiple-instruction-issue processors. In *Proceedings of the Eighteenth Annual International Symposium on Computer Architecture*, pages 266–275, May 1991.

[12] J. C. Gyllenhaal. A machine description language for compilation. Master's thesis, Department of Electrical and Computer Engineering, University of Illinois, Urbana, IL, 1994.

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

[14] V. Kathail, M. S. Schlansker, and B. R. Rau. HPL PlayDoh architecture specification: Version 1.0. Technical Report HPL-93-80, HP Laboratories, February 1994.

[15] T. A. Proebsting and C. W. Fraser. Detecting pipeline structural hazards quickly. *Twenty-First Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 280–286, January 1994.

[16] T. Müller. Employing finite automata for resource scheduling. *Proceedings of the 26th Annual International Symposium on Microarchitecture*, pages 12–20, 1993.

[17] V. Bala and N. Rubin. Efficient instruction scheduling using finite state automata. *To appear in the Proceedings of the 28th Annual International Symposium on Microarchitecture*, November 1995.

[18] M. Lam. Software pipelining: An effective scheduling technique for VLIW machines. *Proceedings of the ACM SIGPLAN'88 Conference on Programming Language Design and Implementation*, pages 318–328, June 1988.

[19] Digital Equipment Corp., Maynard, MA. *DecChip 21064 Microprocessor Hardware Reference Manual EC-N0079-72.*

[20] G. Kane and J. Heinrich. *MIPS RISC Architecture.* Prentice Hall, 1992.

[21] G. R. Beck, D. W. L. Yen, and T. L. Anderson. The Cydra 5 mini-supercomputer: Architecture and implementation. In *The Journal of Supercomputing,* volume 7, pages 143–180, 1993.

[22] E. S. Davidson, L. E. Shar, A. T. Thomas, and J. H. Patel. Effective control for pipelined computers. *Spring COMPCON-75 digest of papers,* pages 181–184, February 1975.

[23] J. H. Patel and E. S. Davidson. Improving the throughput of a pipeline by insertion of delays. *Proceedings of the Third Annual International Symposium on Computer Architecture,* pages 159–164, 1976.

[24] M. S. Schlansker. Personal communication. June 1995.

[25] A. E. Eichenberger, E. S. Davidson, and S. G. Abraham. Optimum modulo schedules for minimum register requirements. *Proceedings of the International Conference on Supercomputing,* pages 31–40, July 1995.

[26] A. E. Eichenberger and E. S. Davidson. Stage scheduling: A technique to reduce the register requirements of a modulo schedule. *To appear in the Proceedings of the 28th Annual International Symposium on Microarchitecture,* December 1995.

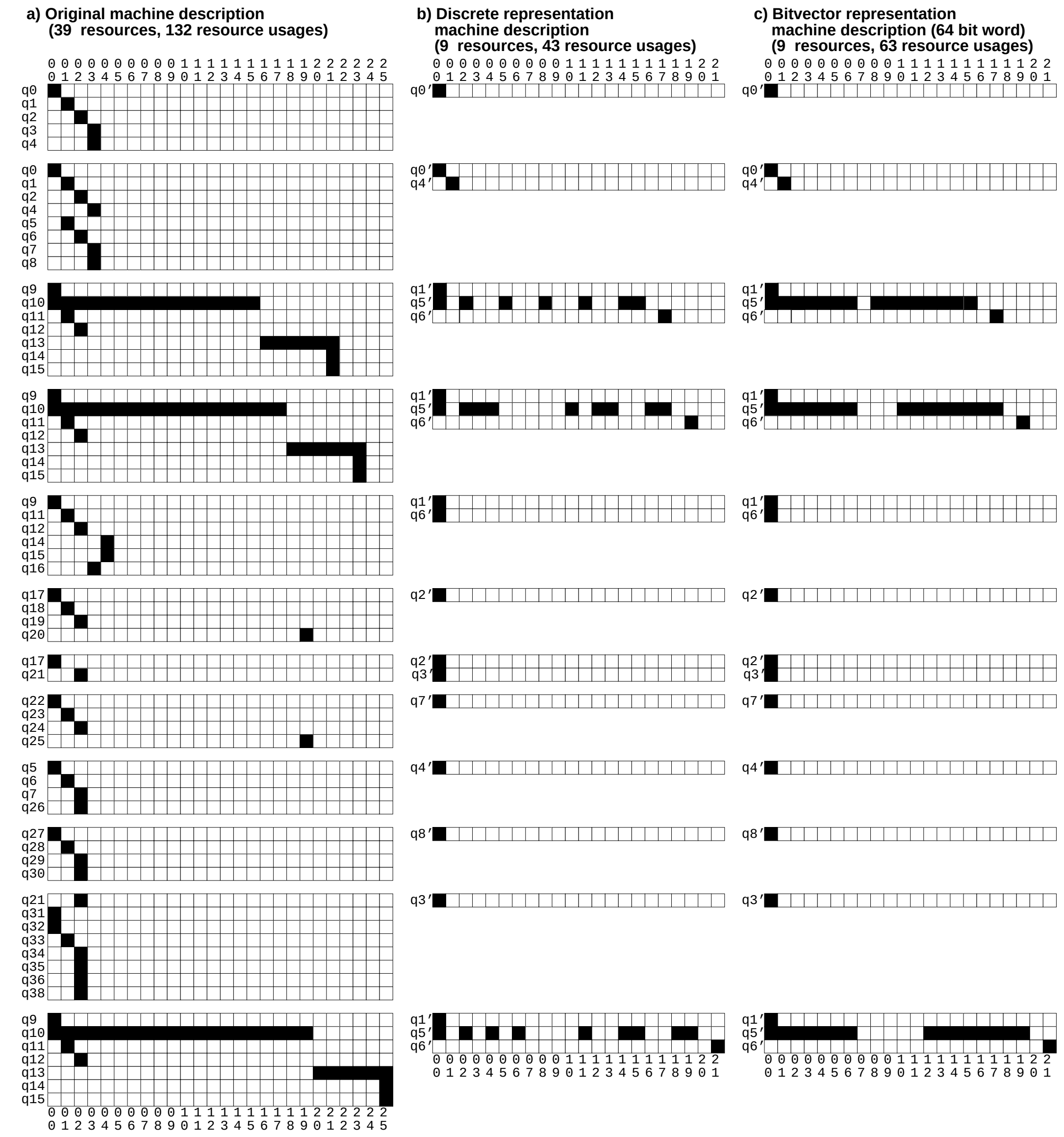

a) Original machine description
(39 resources, 132 resource usages)
b) Discrete representation machine description
(9 resources, 43 resource usages)
c) Bitvector representation machine description (64 bit word)
(9 resources, 63 resource usages)

Figure 4: Reservation tables for a subset of the Cydra 5.