Optimization of Machine Descriptions for Efficient Use

John C. Gyllenhaal Wen-mei W. Hwu
Center for Reliable and High-Performance Computing
University of Illinois, Urbana-Champaign, IL 61801
gyllen, hwu@crhc.uiuc.edu

B. Ramakrishna Rau
Hewlett Packard Laboratories
1501 Page Mill Road, Palo Alto, CA 94304
rau@hpl.hp.com

Abstract

A machine description facility allows compiler writers to specify machine execution constraints to the optimization and scheduling phases of an instruction-level parallelism (ILP) optimizing compiler. The machine description (MDES) facility should support quick development and easy maintenance of machine execution constraint descriptions by compiler writers. However, the facility should also allow compact representation and efficient usage of the MDES during compilation. This paper advocates a model that allows compiler writers to develop the MDES in a high-level language, which is then translated into a low-level representation for efficient use by the compiler. The discrepancy between the requirements of the high-level language and the low-level representation is reconciled with a collection of transformations that derive efficient low-level representations from the easy-to-understand high-level descriptions. In order to support these transformations, a novel approach to representing machine execution constraints has been developed. Detailed and precise descriptions of the execution constraints for the HP PA7100, Intel Pentium, Sun SuperSPARC, and AMD-K5 processors are analyzed to show the advantage of using this new representation. The results show that performing these transformations and utilizing the new representation allow easy-to-maintain detailed descriptions written in high-level languages to be efficiently used by ILP-optimizing compilers.

1. Introduction

Machine descriptions (MDES) have been used to specify execution constraints for several high-performance compilers [1][2]. These machine descriptions are primarily used to drive the instruction scheduler, which uses this information to avoid resource conflicts and data dependence interlocks. However, in future compilers more compiler modules will also need to use MDES information. As compilers push to increase the performance of processors by exploiting instruction-level parallelism (ILP), transformations such as predication and height reduction also need to use execution constraints to avoid over-subscription of processor resources[1]. Currently most compiler modules, including some schedulers, forgo the use of a machine description altogether and instead rely on heuristic "knobs" and algorithmic changes to tune for a specific processor. This is due, in part, to the difficulties involved in providing these modules with access to an accurate machine description, in a form they can efficiently use.

Since each scheduling decision, and potentially each optimization decision, for every operation involves checking execution constraints, the efficiency of such checks can significantly impact the compile time. As a result, compiler writers have faced the choice between two undesirable alternatives. One alternative is to sacrifice portability for accuracy. A compiler designed for a particular processor often uses an accurate, very low-level representation of the machine's description (commonly coded directly into the compiler), that must be tediously modified in order to be effective for subsequent processors. This approach is not desirable in the highly competitive microprocessor industry, where complex new processors are being rapidly designed and brought to the market. Timely development of effective compilers for these new processors is critical to the realization of their full performance potential.

The other alternative is to sacrifice accuracy in favor of portability. Compilers designed to support a wide range of processors, such as gcc, usually describe the machine to their instruction schedulers with easy-to-modify metrics, such as the function unit mix and operation latencies, but these metrics can only approximately model the complex execution constraints in today's superscalar processors. Inaccurate modeling of execution constraints during compilation makes it difficult for the compiler to properly address run-time issues such as resource conflicts and data dependence interlocks. As a result, unexpected execution cycles arise during run time. In processors that exploit high degrees of instruction-level parallelism (ILP), these extra execution cycles can have a significant effect on the overall performance. Accurate modeling of execution constraints is therefore necessary in order to properly utilize these complex processors.

The possibility of using a generic, high-quality scheduler and ILP optimizer driven by an MDES that can be quickly targeted to a new processor is attractive. This paper advocates a model which allows writers to develop an MDES in a high-level language, which is then translated into a low-level representation for efficient use by the compiler. The high-level language should be designed to allow the specification of detailed execution constraints in an easy-to-understand, maintainable, and retargetable manner. The low-level representation should be designed to allow the compiler to check execution constraints with high efficiency in both space and time. The discrepancy

[1] Although this paper will focus on modeling resource constraints, machine descriptions also contain other important information such as operation latencies, the modeling of bypassing and forwarding effects, and the mapping of this information to specific operations based on their opcode and the type of operands expected.

between the requirements of the high-level language and the low-level representation should then be reconciled with a collection of transformations that derive efficient low-level representations from the easy-to-understand high-level descriptions.

The two-tier model is analogous to using high-level programming languages now that contemporary compiler technology has eliminated the benefits of using assembly language for general purpose programs. The user of a high-level machine description language is not required to be intimately familiar with modules using the machine description, and does not need to manually optimize the description for those modules. In fact, a few of the transformations described in this paper are adapted from the classical compiler techniques that helped make high-level programming languages so well accepted. There are, however, important transformations introduced in this paper that have no direct correspondence in the optimizing compiler domain. These transformations take advantage of the unique characteristics of an MDES to increase the efficiency of the resulting low-level representation.

In addition to describing and evaluating the transformations, a novel representation of resource constraints for complex processors is presented. This representation exposes critical information that can be profitably exploited by both these transformations and the compiler modules. This approach to representing machine execution constraints is based on the AND/OR-tree concept used in search algorithms. In order to show the effectiveness of the new representation and transformations, detailed and precise descriptions of the execution constraints for the HP PA7100, Intel Pentium, Sun SuperSPARC, and AMD-K5 processors are constructed in a high-level language. The low-level representation of these descriptions is then generated and used to drive a multi-platform list scheduler. Using a scheduler for the concrete evaluation of these transformations allows the rationale behind and the effect of each transformation to be clearly shown. Similar benefits should be seen by other compiler modules that can benefit from using an accurate machine description.

Although a specific high-level machine description language [3], low-level representation [4], multi-platform compiler [5], and multi-platform list scheduler [6] are being used to validate these techniques, the aim of this paper is to show the general applicability of these techniques, not to proselytize the specific components used. Concepts key to understanding the example and results will be briefly explained, but these explanations will only describe some of the above components' capabilities, and the interested reader is referred to the papers and reports that deal directly with these components.

A description and analysis of a common mechanism used to model resource constraints, which is used by this paper's machine descriptions, follows this section. A new representation of these resource constraints is introduced in Section 3, and is shown to be well suited for describing today's complex processors. An analysis of each MDES, before any transformations are performed, is presented in Section 4. In Section 5, the importance of adapting common-subexpression elimination, copy propagation, and dead-code removal to clean up machine descriptions is shown, as well as how the new representation facilitates these transformations. Section 6 provides a brief overview of the implications of using bit-vectors in the low-level

Option 1

Cycle	Decoder 0	Decoder 1	Decoder 2	M 0	Wr Pt 0	Wr Pt 1
-1	X					
0				X		
1					X	

Option 2

Cycle	Decoder 0	Decoder 1	Decoder 2	M 0	Wr Pt 0	Wr Pt 1
-1		X				
0				X		
1					X	

Option 3

Cycle	Decoder 0	Decoder 1	Decoder 2	M 0	Wr Pt 0	Wr Pt 1
-1			X			
0				X		
1					X	

Option 4

Cycle	Decoder 0	Decoder 1	Decoder 2	M 0	Wr Pt 0	Wr Pt 1
-1	X					
0				X		
1						X

Option 5

Cycle	Decoder 0	Decoder 1	Decoder 2	M 0	Wr Pt 0	Wr Pt 1
-1		X				
0				X		
1						X

Option 6

Cycle	Decoder 0	Decoder 1	Decoder 2	M 0	Wr Pt 0	Wr Pt 1
-1			X			
0				X		
1						X

Figure 1: The six reservation tables that represent the resources used by the SuperSPARC's integer load operation.

representation of resource constraints in order to set the stage for the Section 7, which describes transformations that make the bit-vector representations more effective. Section 8 describes and analyzes a set of transformations which makes checking resource constraints more efficient. Section 9 summarizes the aggregate effect of all these transformations, with and without the new representation. In Section 10, a brief summary of related work is presented and is followed by some concluding remarks.

2. Modeling Resource Constraints

This paper's machine descriptions model the processor's resource constraints through the use of a set of *reservation tables* [7], an approach used by several high-performance MDES-driven compilers [1][2]. In particular, this paper's machine descriptions are based on the approach used by the Cydra 5 [1]. Each reservation table specifies a particular way an operation may use a processor's resources as that operation executes. For example, the resources used by the execution of a SuperSPARC's one-cycle integer load can be modeled with the six reservation tables (each called a *reservation table option*, or simply an *option*) that are shown in Figure 1. An integer load must use the SuperSPARC's only memory unit (M), but may use any of the three decoders (Decoder) and two register write ports (Wr Pt). The register read ports for the address generation unit are dedicated and do not need to be modeled. All option lists are prioritized (option 1 having highest priority), so for the order shown in this figure, the first available (lowest numbered) decoder and register write port will be used by the integer load. The "Cycle" column of these tables indicates the *usage time*, which indicates when each of these resources is used, relative to some chosen time point in the processor's pipeline. For all the examples and machine descriptions used in this paper, the point chosen to be time "zero" is the first stage of the execution pipeline. Therefore resources used during decoder stages have negative usage times, and resources used after execution completes, such as result buses and register write ports, have usage times around the operation's latency. A resource used at a particular usage time will be referred to as a *resource usage*. It should be noted that the resources modeled often do not represent actual processor resources, but are abstractions used to model the processor's scheduling rules. This approach was used in the construction of the machine descriptions used in this paper. These machine descriptions were designed to accurately

Number of Options	% of Scheduling Attempts	Operations Modeled
1	13.41%	Branches and serial ops
3	0.72%	Floating-point ops
6	14.37%	Load ops
12	4.92%	Store ops
24	9.24%	Shifts and cascaded IALU ops that use 1 read port
36	3.00%	Shifts and cascaded IALU ops that use 2 read ports
48	50.29%	IALU ops that use 1 read port
72	4.05%	IALU ops that use 2 read ports

Table 1: Option breakdown and scheduling characteristics for the SuperSPARC MDES.

and precisely model the processor's scheduling rules, and intuitive resource names were used solely to enhance the clarity of the machine descriptions.

Many of this paper's examples are drawn from the SuperSPARC MDES, so a brief and somewhat simplified overview of the SuperSPARC's execution constraints is necessary before continuing. The SuperSPARC [8] is an in-order superscalar processor that has three full decoders, four integer register read ports (RP), two integer register write ports, two integer ALU (IALU) units, one barrel shifter, one memory unit with a dedicated address generation unit, one branch unit, and support for one floating-point operation per cycle. The address generation unit and the floating-point function units have dedicated register ports which do not need to be modeled, but it is important to model the usage of the SuperSPARC's integer register read and write ports. All of the common integer operations have a one cycle latency, and the load operations also have a one cycle latency. However, load and store operations can cause address generation interlocks if they are not scheduled properly.

The SuperSPARC's design also allows the execution of two flow-dependent IALU operations in the same cycle. The second IALU operation, which utilizes this feature to execute a cycle early, is referred to as a *cascaded* IALU operation. The number of reservation table options required to model an IALU operation depends on the number of register source operands that the operation has. A non-cascaded IALU operation with one register source may use any one of the decoders, read ports, IALU units, and write ports, yielding $\binom{3}{1}\binom{4}{1}\binom{2}{1}\binom{2}{1}=48$ distinct combinations of these resource usages. The same non-cascaded IALU operation with two register sources, requiring two read ports, has $\binom{3}{1}\binom{4}{2}\binom{2}{1}\binom{2}{1}=72$ distinct combinations of resource usages. Each of these distinct combinations of resource usages is modeled by a reservation table option. There is only one IALU available to execute cascaded IALU operations, so cascaded IALU operations only have half the reservation table options of non-cascaded IALU operations. The appropriate set of reservation table options is chosen based on an operation's incoming dependence distances.

A breakdown of the number of reservation table options used to model the various operations in the SuperSPARC MDES is shown in Table 1. This first column specifies the number of scheduling options. The second column indicates the percentage of the time that a multi-platform list scheduler, driven by

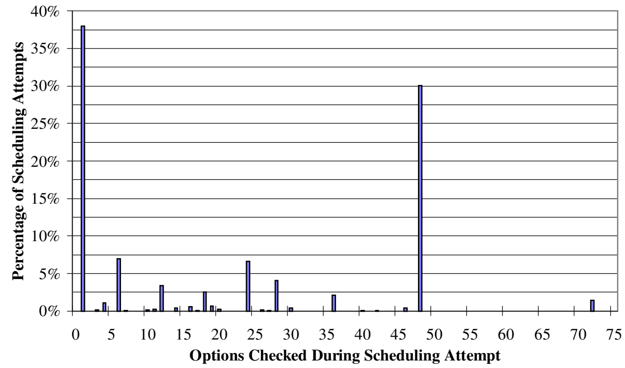


Figure 2: Distribution of options checked during each scheduling attempt using the SuperSPARC MDES.

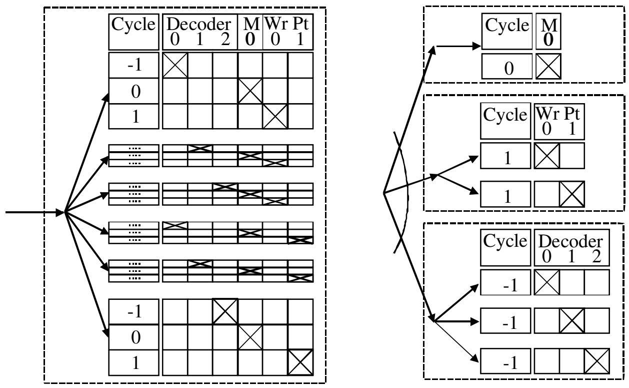
this MDES and scheduling SuperSPARC SPEC CINT92 assembly code, attempted to schedule an operation with that many options during prepass scheduling. The last column gives a brief summary of the types of operation that have that many options. Note that for the SuperSPARC and the PA7100 MDES, branches are modeled as always using the last decoder in order to maximize scheduling freedom (since nothing may issue after a branch on these machines).

Although all of the reservation table options must be tested in order to determine that an operation cannot be scheduled, a variable number of the options need to be tested in order to determine that the operation can be scheduled. Figure 2 shows the distribution of options actually checked while scheduling for the SuperSPARC. On average, 2.05 scheduling attempts were required per operation, so roughly half of the time a scheduling attempt fails. The 30.05% peak at 48 options checked is primarily due to the fact that 58.50% of unsuccessful scheduling attempts were on operations with 48 options. Overall, 45.52% of the scheduling attempts required between 24 and 72 options to be checked. The 38.02% peak for one scheduling option checked is due mainly to scheduling attempts that succeed with the first option attempted. For successful scheduling attempts, 73.75% succeed with the first option tested, 8.23% tested between 2 and 16 options, 16.71% tested between 17 and 32 options, and 1.31% tested more than 33 options.

In the following sections, transformations will be presented that make testing each option nearly as efficient as possible. However, unless the number of options checked can be reduced, modeling complex machines exactly will remain expensive in terms of compile time. A new representation, presented in the next section, can dramatically reduce the number of options checked in complex machine descriptions.

3. A New Representation: AND/OR-Trees

The primary reason so many options need to be checked for complex processor descriptions is that the traditional representation for resource constraints hides useful information from the compiler. By exposing this useful information with the new representation presented below, the compiler can more efficiently check the resource constraints. This representation can also inherently reduce the MDES size (Section 4), facilitate size-reducing transformations (Section 5), and facilitate transformations to further optimize for resource conflict detection



a) b)

Figure 3: Two methods of modeling the resource constraints of a SuperSPARC integer load operation. a) The traditional OR-Tree representation. b) The proposed AND/OR-Tree representation.

(Section 8). Before describing this new representation, a brief review of the traditional representation is in order.

The traditional representation can be viewed as an OR-Tree, as shown in Figure 3a. This figure shows the six reservation table options for the SuperSPARC integer load (the same options that are shown in Figure 1). The options are in priority order (with the highest priority first), and if the resources for any of the options are available, the operation can be scheduled. The advantage of this representation is that for OR-trees with a small number of options, the OR-tree's resource constraints can be quickly and efficiently checked. For processors which have execution constraints that can be modeled with just a few reservation table options, it is difficult to improve upon the efficiency of this OR-tree representation.

The disadvantage of this OR-tree representation is that it does not allow an easy or efficient way of using information about why an option was not available. For example, if Option 1 (the top option) in Figure 3a is unavailable because write port 0 (Wr Pt 0) is unavailable, then Options 2 and 3 are also guaranteed to be unavailable. Although an inference engine could be programmed into the resource constraint check algorithm to eliminate these options, the overhead would more than negate the benefit.

The solution proposed in this paper is to use a new representation that is based on the AND/OR-tree concept used in search algorithms. This new representation is, in essence, an AND-tree of OR-trees, allowing multiple OR-trees to be used together in order to represent the resource constraints. An example of this new AND/OR-tree representation is shown in Figure 3b. The AND/OR-tree shown specifies the resource requirements for the SuperSPARC's integer load as requiring the memory unit(M), one of the two write ports (Wr Pt), and one of the three decoders. By utilizing the short-circuit properties of AND and OR, the resource constraint check algorithm can quickly determine which of the required resources are available (or if they are not available), without performing any unnecessary checks.

The algorithm overhead incurred by using this new representation is minimal, since it is built upon the OR-tree representation and does not require any new information from the OR-tree resource constraint checker (i.e. which option, if any, is available). In Figure 3, each of the OR-trees, which are enclosed in dotted boxes, can have the same internal representation and may have the same resource constraint checker algorithm applied to them. The compiler used for this paper's experiments does so and, for implementation efficiency, adds an outer loop around the OR-tree's algorithm that processes the array of OR-trees associated with an AND/OR-tree. Although some additional space is required to represent the AND-level of the tree, the use of AND/OR-trees can significantly reduce the size of the resource constraint description in the MDES, as shown in Section 4.

4. Original MDES Characteristics

In this paper, detailed and precise descriptions of the execution constraints for the HP PA7100, Intel Pentium, Sun SuperSPARC, and AMD-K5 are analyzed to show the rationale behind the transformations presented in the following sections and the advantages of using the AND/OR-tree representation. For this analysis, each of these machine descriptions is used to drive a multi-platform list scheduler, which is then used to schedule SPEC CINT92 assembly code for that platform. Each platform's assembly code (between 201011 and 282219 static operations) was generated using the level of profile-driven inlining, classical optimization, ILP optimization, and peephole optimization that had been found, through extensive tuning, to yield the highest possible execution-time performance for that platform[2]. The analysis also focused on prepass scheduling for the SuperSPARC and PA7100, and on postpass scheduling for the Pentium and K5. Prepass scheduling was not performed for the X86 processors due to the limited number of registers available. The SuperSPARC was described in Section 2, and the reservation table option breakdown of its machine description was shown in Table 1. A brief description of the other processors modeled, and their reservation table option breakdowns, is in order before analyzing the original characteristics of each MDES.

The PA7100 [9] is an in-order superscalar processor that has two decoders, and supports executing one floating-point operation in parallel to an integer or memory operation. The relative order of these two operations does not matter, so there are two options for most operations, as shown in Table 2. The Pentium [10] is an in-order superscalar X86 processor that has two execution pipelines. A detailed set of pairing rules is used to specify the operation combinations that may execute in parallel. Each operation has one or two reservation table options, as shown in Table 3. The K5 [11] is a four-issue, out-of-order, superscalar X86 processor that the MDES models as an in-order processor that can buffer operations between decode and execution. This processor converts X86 operations into one or more *Rops* (internal RISC operations), which may be dispatched in different cycles if the required resources are not available. Accurate modeling allows the scheduler to take advantage of this dynamic behavior, potentially increasing decoder and execution

[2] The relative performance level for each platform and benchmark varied, but each benchmark's performance was either close to, or better than, the published peak SPEC CINT92 numbers for that platform.

Number of Options	% of Scheduling Attempts	Operations Modeled
1	18.81%	Branch ops
2	81.19%	Ops that can use either decoder

Table 2: Option breakdown and scheduling characteristics of the PA7100 MDES.

Number of Options	% of Scheduling Attempts	Operations Modeled
1	45.42%	Ops that can execute in only 1 pipe
2	54.58%	Ops that can execute in either pipe

Table 3: Option breakdown and scheduling characteristics of the Pentium MDES.

Number of Options	% of Scheduling Attempts	Operations Modeled
16	14.72%	1 Rop ops with 1 unit choice
24	0.14%	2 Rop ops dispatched in 1 cycle (1 unit choice)
32	74.72%	1 Rop ops with 2 unit choices
48	5.91%	2 Rop bundled cmp+br dispatched in 1 cycle
64	2.56%	3 Rop bundled cmp+br dispatched in 1 cycle
96	0.19%	2 Rop ops dispatched in 1 cycle (2 unit choices)
128	0.66%	2 Rop bundled cmp+br dispatched over 2 cycles
192	0.15%	3 Rop ops dispatched over 2 cycles (subset of)
256	0.37%	2 Rop ops dispatched over 2 cycles (2 unit choices)
384	0.43%	3 Rop bundled cmp+br dispatched over 2 cycles
768	0.15%	3 Rop ops dispatched over 2 cycles (subset of)

Table 4: Option breakdown and scheduling characteristics of the K5 MDES.

			OR-Trees		AND/OR-Trees		
MDES	Total Ops Sched.	Avg. Sched. Attempts Per Op	Avg. Options/ Attempt	Avg. Checks/ Attempt	Avg. Options/ Attempt	Avg. Checks/ Attempt	Percent Checks Reduced
PA7100	201011	1.94	1.56	2.47	1.45	1.96	20.6%
Pentium	207341	1.47	1.49	3.99	1.49	3.99	0.0%
S-SPARC	282219	2.05	21.48	31.09	4.38	4.83	84.5%
K5	203094	1.62	19.59	35.49	5.20	5.73	83.9%

Table 5: Original scheduling characteristics of the machine descriptions for each target machine.

		OR-Trees		AND/OR-Trees		
MDES	Number of Trees	Table Options	Size (bytes)	Table Options	Size (bytes)	%Size Reduced
PA7100	15	40	2504	16	1384	44.7%
Pentium	37	34	14824	34	15416	-4.0%
SuperSPARC	24	333	17120	40	2624	84.7%
K5	33	4424	312640	64	4376	98.6%

Table 6: Original MDES memory requirements.

utilization. The K5's design allows up to four X86 operations to be decoded each cycle and up to four Rops to be dispatched each cycle. It also has up to two execution units available for each type of Rop. As shown in Table 4, 89.44% of scheduling attempts are for one-Rop X86 operations that have 16 or 32 reservation table options. However, up to 768 reservation table options are required to accurately model a multi-Rop X86 operation, which can be dispatched over multiple cycles. Failure to model these operations correctly can impact performance in critical loops.

To clarify the notation in the tables, it should be noted that for both of the X86 processors, the compiler *bundles* each branch together with an appropriate condition-code-setting operation, in order to maximize scheduling freedom. The reservation tables for these bundled operations model the resources required by all the operations in the bundle. After scheduling, these bundled operations are converted back into individual operations.

Although it is not necessary for the high-level MDES language to support AND/OR-trees, AND/OR-trees provide a concise way of specifying complex resource usages. The machine descriptions used in this paper's experiments are written in a high-level MDES language that supports the specification of both OR-trees and AND/OR-trees. Every MDES, except the Pentium MDES, uses AND/OR trees extensively. (The Pentium's execution constraints do not have the flexibility that benefits from the use of AND/OR-trees.) In order to generate the OR-tree MDES representations for this paper's experiments, each MDES that uses AND/OR-trees was run through an MDES preprocessor that expanded out each AND/OR-tree specification into the corresponding OR-tree specification.

The scheduling characteristics for these machine descriptions, before any transformations are performed, are shown in Table 5. The second and third columns indicate how many operations were scheduled, and how many scheduling attempts were required, on average, before an operation was successfully scheduled. These two numbers will remain constant throughout all of the transformations (with either representation), and the exact same schedule is produced in each case, since all the execution constraints described in the machine descriptions are being preserved. It should be noted that the number of scheduling attempts required per operation can increase significantly with the use of more advanced scheduling techniques such as iterative modulo scheduling [12] and operation scheduling, and with the application of more ILP optimizations to the assembly code. However, the selected experimental setup closely models current compiler practices, and the benefit of this paper's AND/OR-tree representation and MDES transformations should only increase as more scheduling attempts are required, since they speed up detection of resource-constraint conflicts.

The fourth and sixth columns of Table 5 show the average number of reservation table options checked for each scheduling attempt, for the OR-tree and AND/OR-tree representations, respectively. The fifth and seventh columns show the average number of resource checks that were required for each scheduling attempts. The last column shows that for complex machine descriptions, before any transformations are performed, the use of the AND/OR-tree representation can reduce the number of resource checks per reservation table option by up to 84.5%.

The memory required to internally represent the resource constraints in the compiler used for this paper's evaluation is shown in Table 6. Although this internal representation has been extensively tuned to maximize the performance of the resource constraint checking algorithm, it also was designed to minimize memory requirements in ways that incur no performance penalty. To this end, the internal representation allows common information to be shared among AND/OR-trees and OR-trees, but in some cases a small amount of header informa-

MDES	Num. of Trees	OR-Tree Rep. Table Options	Size (bytes)	% Size Reduced	AND/OR-Tree Rep. Table Options	Size (bytes)	% Size Reduced
PA7100	13	24	1712	31.6%	14	1232	11.0%
Pentium	30	28	10816	27.0%	28	11296	26.7%
S-SPARC	19	277	14752	13.8%	30	1896	27.7%
K5	27	3704	266032	14.9%	55	3592	17.9%

Table 7: MDES memory requirements after eliminating redundant and unused information.

tion per item is duplicated to prevent performance degradation. Both the OR-tree and the AND/OR-tree internal representations have the same number of trees. However, the table shows that the AND/OR-tree representation, because it does not require the explicit enumeration of all the resource usage combinations (OR-tree options), can significantly reduce the memory required (a 98.6% reduction for the K5). Thus, before any MDES optimizations, the AND/OR-tree significantly reduces both the internal representation size and the number of checks required per attempt for complex resource constraint descriptions (i.e. the SuperSPARC and K5). This advantage will remain definitive after both representations are fully optimized.

The sizes shown in this paper for the AND/OR-tree representation reflect the extra memory required to store the AND level of the tree. In the Pentium MDES, the AND level always points to one OR-tree, so the AND/OR-tree representation will always be slightly larger. It should also be noted that the common information to be shared is entirely specified by the external MDES representation, in order to minimize the time required to load the MDES into memory. The number of trees and reservation table options shown in the table reflects only what the writer of the MDES specified as being shared. It is easy and natural to specify shared information in the high-level MDES language used, so most of the common information is shared in these machine descriptions. However, common information is often not shared in order to make the machine description more readable or easier to modify. In fact, some of the information in the MDES may not even be used. The transformations presented in the Section 5 will deal with these issues.

5. Eliminating Redundant or Unused MDES Information

Machine descriptions tend to evolve as a processor's execution constraints become more thoroughly understood, as the compiler's vocabulary of operations increases, and as these machine descriptions are ported to different or experimental processors in the family. As the machine descriptions evolve, the amount of redundant and unused information in the MDES tends to grow because, for an MDES writer, it is typically easier to just make a local copy of the information to be changed than to do the careful analysis required to safely modify or delete existing information. In fact, this was experienced both at Cydrome Inc., with creation and maintenance the Cydra 5 MDES, and by the authors of this paper's machine descriptions.

This redundant and unused information can be eliminated from the MDES by adapting the classical compiler optimizations (common sub-expression elimination, copy propagation, and dead-code removal [13]) to the MDES domain. In this paper's implementation, common sub-expression elimination and copy propagation were combined into one step that finds redundant MDES information and points all various references to that information to only one particular copy, and an adaptation of dead-code removal eliminates unreferenced information. These techniques greatly reduced the size required to represent all of the aspects of the MDES, such as resource constraints, operation latency, and operation format. Their effect on the resource-constraint description size in particular is shown in Table 7.

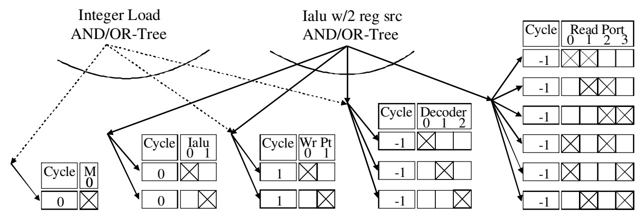


Figure 4: An example of how the AND/OR-tree representation can facilitate the sharing of OR-trees.

MDES	Total Ops Sched.	Avg. Sched. Attempts Per Op.	OR-Tree Avg. Options/ Attempt	Avg. Checks/ Attempt	AND/OR-Tree Average Options/ Attempt	Avg. Checks/ Attempt	Percent Checks Reduced
PA7100	201011	1.94	1.42	2.32	1.38	1.89	18.5%

Table 8: PA7100 scheduling characteristics after removing unnecessary options for memory operations.

It is interesting to note that the AND/OR-tree representation for the SuperSPARC and K5 machine descriptions benefited more from eliminating redundant information than the OR-tree representation. This is because the reservation table options in the AND/OR-tree representation typically specify the resource usages at a finer granularity (less usages per option) than the OR-tree options, allowing them to be shared more aggressively. In addition, the OR-trees in an AND/OR-tree tend to be more general-purpose, allowing entire OR-trees to be shared by several AND/OR-trees. An example of this second case is shown in Figure 4, where the OR-trees for decoder and register write port resource usages are shared by the SuperSPARC's integer load AND/OR-tree and the SuperSPARC's integer ALU (with two register sources) AND/OR-tree. In this way the AND/OR-tree representation facilitates further reduction of the MDES size.

The transformations for removing redundant information can also be adapted to more MDES specific circumstances, such as removing options from an OR-tree that can be determined to be impossible to satisfy. An option can be removed from an OR-tree if its resource usages are identical to, or a superset of, the resource usages for a higher-priority option, since the higher-priority option will always be selected if these resources are available. This case can arise when the use of preprocessor directives enumerates the various OR-tree options, and it can also arise as a machine description evolves, which is the case for the PA7100 MDES used in this paper. The PA7100 MDES was derived from the MDES for an earlier HP PA processor. During the retargeting, two of the reservation table options for the PA7100's memory operations became identical, but the MDES author never realized this since correct output was still generated. The effect this has on the PA7100's scheduling characteristics is shown in Table 8.

MDES	Memory Requirements (in bytes)					
	OR-Tree Rep.			AND/OR-Tree Rep.		
	Before	After	Diff.	Before	After	Diff.
PA7100	1712	1408	17.8%	1232	1128	8.4%
Pentium	10816	3224	70.2%	11296	3704	67.2%
SuperSPARC	14752	11152	24.4%	1896	1640	13.5%
K5	266032	183280	31.1%	3592	3136	12.7%

Table 9: MDES size characteristics before and after a bit-vector representation is used (one cycle/word).

MDES	Average Checks Per Scheduling Attempt					
	OR-Tree Rep.			AND/OR-Tree Rep.		
	Before	After	Diff.	Before	After	Diff.
PA7100	2.32	2.18	6.0%	1.89	1.76	6.9%
Pentium	3.99	2.31	42.1%	3.99	2.31	42.1%
SuperSPARC	31.09	26.69	14.2%	4.83	4.62	4.3%
K5	35.49	34.35	3.2%	5.73	5.30	7.5%

Table 10: Scheduling characteristics before and after a bit-vector representation is used (one cycle/word).

6. Utilizing Bit-Vector Representations

The results presented so far have not taken advantage of the fact that most resource-constraint checking algorithms, including the one used in this paper, use bit-vectors [14][15][16] to keep track of the resources used each cycle in what is referred to as a *resource usage map* (RU map). This design allows the RU map size to be minimized and efficiently initialized, and allows multiple resource usages to be checked (and reserved) with a single AND (OR) operation. In addition, using bits in the MDES to represent multiple resource usages can significantly decrease the MDES size. Although it is possible to pack more than one cycle's resource usages into a single memory word, it is not necessary to do so for the machine descriptions in this paper. The resource usage time transformation presented in the next section will reduce the number of checks to almost the minimum of one resource check per reservation table option.

The incremental effect of packing each cycle's resource usages into one memory word is shown in Tables 9 and 10. Before using bit-vectors, each resource usage was represented as a cycle/resource pair (one resource usage per check). After using bit-vectors, the resource usages were represented as a cycle/resource-vector pair (multiple resource usages per check possible, if the usages are in the same cycle). Although both representations require two words to represent each pair, the bit-vector representation typically requires less pairs per table. The Pentium MDES shows the most benefit, since modeling the Pentium's resource constraints required checking several resource usages in every cycle. The other machine descriptions didn't benefit as much, since their resource usage did not always fall within the same cycle. For example, the reservation table options shown in Figure 3a do not benefit from packing a cycle's resource usage into a single memory word, since there is only one usage per cycle. However, the resource usage time transformation presented in the next section will resolve this issue.

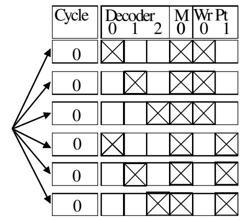


Figure 5: The OR-tree modeling the resource constraints for a SuperSPARC integer load operation, after transforming the resource usage times in order to better utilize the bit-vector representation.

7. Optimizing for Bit-Vector Representations

The use of the actual resource usage times, as in Figure 3a, can significantly reduce the effectiveness of using a bit-vector representation that packs one cycle's worth of resource usages into a single memory word. We address this problem by making use of the theory of pipelined, multi-function unit design [7][17]. For any ordered pair of reservation table options (A, B), t is a *forbidden latency* (i.e., an operation using reservation table option B cannot be initiated t cycles after an operation that uses reservation table option A) if and only if A and B have resource usages for some common resource at times i and j, respectively, such that i is greater than or equal to j and i-j = t. The set of all forbidden latencies between A and B is termed the *collision vector* for the ordered pair (A, B). A given schedule results in no resource conflicts if and only if, for every pair of operations, the difference in their scheduled times never violates the collision vector for the corresponding pair of reservation tables. Note that the actual reservation table options A and B are not directly important; only the collision vector for (A, B) is. Consequently, we could substitute any reservation table options A' and B' for A and B, respectively, as long as the collision vector for (A', B') is the same as that for (A, B). Further note that, in computing a forbidden latency, only the difference between the resource usage times i and j matters, not their actual values. In particular, we could add a common constant to both resource usage times without altering the forbidden latency.

With this in mind, the optimization that we use, for each resource, is to subtract a strategically selected constant from the originally specified resource usage times for that resource in every reservation table option, with a view to concentrating resource usages into as few time slots as possible. The constant may be different for each resource. This optimization is related to the one used by Eichenberger and Davidson [18]. Although minimization techniques can be used to find those constants that maximize the benefit, a simple heuristic was found to be highly effective for the forward-scheduling list scheduler and the processors considered in this paper. The heuristic is, for each resource, to pick the constant to be the earliest resource usage time for that resource (across all reservation table options). The result of this heuristic is to concentrate a far larger number of resource usages than before at time zero, thereby making the bit-vector approach more effective. For a backward-scheduling list scheduler, the constants should be chosen to make the latest usage time to be zero (or some constant). Applying this transformation to Figure 3a yields the OR-tree shown in Figure 5.

MDES	Memory Requirements (in bytes)					
	OR-Tree Rep.			AND/OR Tree Rep.		
	Before	After	Diff.	Before	After	Diff.
PA7100	1408	1168	17.0%	1128	1032	8.5%
Pentium	3224	3080	4.5%	3704	3560	3.9%
SuperSPARC	11152	7016	37.1%	1640	1584	3.4%
K5	183280	125488	31.5%	3136	3096	1.3%

Table 11: MDES memory requirements before and after transforming resource usage times (one cycle per word).

MDES	OR-Tree Rep.				AND/OR-Tree Rep.			
	Avg. Checks/Attempt			Checks/	Avg. Checks/Attempt			Checks/
	Before	After	Diff	Option	Before	After	Diff.	Option
PA7100	2.18	1.59	37.1%	1.12	1.76	1.55	11.9%	1.12
Pentium	2.31	1.57	32.0%	1.05	2.31	1.57	32.0%	1.05
S-SPARC	26.69	21.59	19.1%	1.01	4.62	4.49	2.8%	1.03
K5	34.35	19.87	42.2%	1.01	5.30	5.25	0.9%	1.01

Table 12: Scheduling characteristics before and after transforming resource usage times and sorting the resulting usages to check time zero first (one cycle per word).

In addition to making the bit-vector representation more effective, this transformation also has a subtle effect on the characteristics of the resource usage checks that can be taken advantage of. The resource usages that cause most of the resource conflicts now tend to be concentrated at time zero. The resource usages with times greater than zero are usually conflict free and are primarily there to delay the execution of later operations. For example, the non-zero-time divide-unit usages for a divide operation rarely prevent the operation from being scheduled (if the divide unit is available at time zero). However, while scheduling the next divide operation, the divide unit will not be available at time zero until the previous divide completes. Thus for a forward-scheduling list scheduler, the average number of checks before a conflict is detected is minimized by sorting the resulting usage checks so that time zero is checked first. In this manner, the same machine descriptions can be automatically tuned for other types of schedulers by adjusting the heuristic for picking the resource usage time shift constants and for the sorting of the resulting usage checks.

The MDES memory requirements after transforming the resource usage times are shown in Table 11. The size of the OR-tree representation is reduced up to 37.1% by using this transformation. There is less reduction for the AND/OR-tree representation since this representation tends to have fewer resource usages per option. The transformations presented in the next section do not change the MDES size, so these sizes are the final MDES sizes after full optimization.

The MDES scheduling characteristics after transforming the resource usage times and sorting the resulting usages to check time zero first are shown in Table 12. This transformation reduced the average number of resource checks per option to between 1.01 and 1.12, which is close to the ideal case of one check per option. As a result, the average number of options checked per attempt is what is truly dictating the number of checks required. Although the AND/OR-tree already has a clear advantage in this regard for the SuperSPARC and K5, the number of options checked is further reduced by the transformations in Section 8.

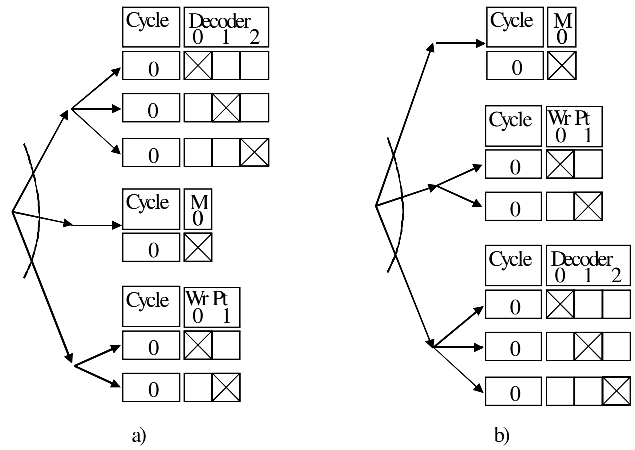


Figure 6: An example of optimizing the order of the OR-trees in an AND/OR-trees for resource conflict detection. a) Original order specified. b) After optimizing the order.

8. Optimizing AND/OR-Trees for Resource Conflict Detection

The structure of the AND/OR-tree representation allows additional transformations to be performed that can increase the chance of detecting resource conflicts early. The first transformation is to sort the sub OR-trees in the AND/OR-tree so that the OR-tree most likely to have a resource conflict (heuristically determined) is checked first. The following heuristic-based sort criteria were found to produce the most consistent results. The OR-trees are first sorted by the earliest usage time in each tree, since after the resource usage time transformation, most conflicts occur at usage time zero. For OR-trees with the same earliest usage time, sort by the number of options in each OR-tree, so that OR-tree with the fewest options is checked first. To break ties at this point, preference is given to the OR-trees that are shared by the most number of AND/OR-trees, since this gives an indication of which OR-trees have resources that are heavily used. Finally the original order specified is used to break any remaining ties. Figure 6a shows the OR-tree order originally specified in the MDES (and used for all previous analysis), and Figure 6b shows the order after sorting the OR-trees using the above criteria (only the second criterion applies).

A second transformation that can be applied is to remove resource usages that are common to all of the OR-tree options and place them in an OR-tree with just one option (creating one if necessary). This transformation works well when a resource common to all options is likely to cause a resource conflict. By pulling it out, this resource conflict can be detected earlier. This transformation can also be used to create some simple AND/OR-trees from OR-tree descriptions. Application of this transformation can actually increase the number of resource checks required, but the following application heuristics were found to yield good results. First, if there is already a one-option OR-tree that has a resource usage with the same usage time as the common usage, apply the transformation. (With bit-vectors, this transformation cannot hurt performance.) Also,

MDES	AND/OR-Tree Representation					
	Options Per Attempt			Checks Per Attempt		
	Before	After	Diff.	Before	After	Diff.
PA7100	1.38	1.38	0.0%	1.55	1.55	0.0%
Pentium	1.49	1.49	0.0%	1.57	1.57	0.0%
SuperSPARC	4.38	2.97	32.2%	4.49	3.08	31.4%
K5	5.20	4.32	16.9%	5.25	4.38	16.6%

Table 13: Scheduling characteristics before and after optimizing AND/OR-trees for resource conflict detection.

apply the transformation if the common usage is the only usage in the OR-tree with that usage time (each option in the OR-tree then has one less check, and in exchange only one check is added). Otherwise, the transformation should not be applied. In the machines descriptions used in this paper, all the applications of this transformation occurred due to the first application rule. After the usage time transformation the second case becomes rare or, for these descriptions, nonexistent.

The incremental effect of these transformations on the AND/OR-tree scheduling characteristics is shown in Table 13. Most of the AND/OR-trees in the SuperSPARC and K5 MDES are reordered, which significantly reduces the average number of options checked before a resource conflict is detected. The MDES sizes did not change due to these transformations.

9. Aggregate Effect of All Transformations

There are two important machine description aspects that are optimized by the transformations presented in this paper. The first one is the amount of memory needed by the compiler to represent the processor's resource constraints. Minimizing this size allows more MDES information to fit within the first-level cache during compilation and also reduces the overall memory requirements of the compiler. The aggregate effect on required memory of all the transformations presented in this paper is shown in Table 14. When applied to an OR-tree representation, these transformations reduce representations by as much as a factor of five. When these transformations are further combined with the AND/OR-trees, representations up to a hundred times smaller than the unoptimized OR-tree representation are produced[3]. For processors with flexible execution constraints (i.e. the SuperSPARC and K5), combining these transformations with the AND/OR-tree representation is especially effective.

The second aspect of the machine descriptions to be optimized is the number of resource checks per scheduling attempt. Minimizing this number reduces the time required to check resource constraints, making room in the compiler's time budget for more advanced scheduling or optimization techniques. The aggregate effect of all the transformations presented in this paper on the average number of resource checks required per scheduling attempt is shown in Table 15. As described in Section 4, these check-per-attempt statistics were generated using an MDES-driven multi-platform list scheduler to schedule SPEC CINT92 assembly code for each platform. When compared to the checks-per-attempt of the unoptimized OR-tree representation, these transformations reduced the number of checks required by the OR-tree representation by up to a factor

[3] As described in Section 4, the Pentium MDES does not take advantage of AND/OR-trees. The size increase is due to representation overhead.

MDES	Memory Requirements (in bytes)				
	Unoptimized	Fully Optimized With Bit-Vector Rep.			
	OR-Trees	OR-Trees	Reduction	AND/OR-Trees	Reduction
PA7100	2504	1168	53.4%	1032	58.8%
Pentium	14824	3080	79.2%	3560	76.0%
SuperSPARC	17120	7016	59.0%	1584	90.7%
K5	312640	125488	59.9%	3096	99.0%

Table 14: Aggregate effect of all transformations on MDES resource-constraint representation size.

MDES	Average Checks Per Scheduling Attempt				
	Unoptimized	Fully Optimized With Bit-Vector Rep.			
	OR-Trees	OR-Trees	Reduction	AND/OR-Trees	Reduction
PA7100	2.47	1.59	35.6%	1.55	37.2%
Pentium	3.99	1.57	60.7%	1.57	60.7%
SuperSPARC	31.09	21.59	30.6%	3.08	90.1%
K5	35.49	19.87	44.0%	4.38	87.7%

Table 15: Aggregate effect of all transformations on MDES scheduling characteristics.

of 2.6. When these transformations are combined with the AND/OR-trees, the number of checks were reduced by as much as a factor of ten. As was seen with the MDES-size aspect, combining these transformations with the AND/OR-tree representation is especially effective at reducing the number of checks required for processors with flexible execution constraints (i.e. the SuperSPARC and K5).

The trend that these tables show is that as the processors become more powerful and flexible, the AND/OR-tree representation, combined with the described transformations, becomes crucial for keeping under control both the MDES size and the number of checks per scheduling attempt. We expect the K5 MDES results to be representative of the latest generation of microprocessors, such as the Intel Pentium Pro and the HP PA8000.

10. Related Work

Eichenberger and Davidson [18] recently proposed a minimization algorithm which, for each reservation table option, generates an equivalent reservation table option with a minimum[4] number of resource usages. The total number of resources used to model the processor is also minimized, which facilitates packing multiple cycles of resource usages into a bit-vector. This algorithm, combined with a bit-vector representation, was shown to minimize both the memory required to represent each option and the number of resource checks per option. However, Eichenberger and Davidson do not address the problem of reducing the number of option checks per scheduling attempt.

The transformations presented in this paper reduce the number of resource checks and memory required per option to a level close to that obtained using the Eichenberger and Davidson algorithm, although a different approach is used. In addition, when combined with our proposed AND/OR-tree representation, our transformations simultaneously optimize the number of options checks per scheduling attempt, the number of

[4] This algorithm uses heuristics to avoid exhaustive searches. Although true minimums may not always be found, the results are near optimal.

resource checks per option, and the memory required to represent the processor's resource constraints.

Proebsting and Fraser [19], Müller [20], and Bala and Rubin [21] have proposed approaches that use finite-state automata, instead of resource reservation tables, to determine if an operation may be scheduled without a resource conflict. These techniques, when compared with the use of unoptimized reservation tables and representations, have shown significant reductions in the number of checks per scheduling attempt and in representation size. However, the combination of our transformations and AND/OR-tree representation appear to mitigate these advantages, even for complex resource constraints. In addition, as described in the literature, finite-state automata do not appear to support some advanced scheduling techniques, such as iterative modulo scheduling [12], which strategically unschedule operations in order to remove the resource conflicts that are preventing an operation from being scheduled. This requires the ability to identify and unschedule the operations that are causing the resource conflicts. This type of unscheduling is straightforward with reservation tables (implemented in [12]), but it is unclear how to do this type of unscheduling with finite-state automata.

11. Conclusions

This paper advocates a model which allows compiler writers to develop easy-to-understand, maintainable machine descriptions in a high-level language, which is then translated into a low-level representation for efficient use by the compiler. To reconcile the discrepancy between the requirements of the high-level language and the low-level representation, a collection of transformations was presented that derives efficient low-level representations from descriptions written in a high-level MDES language. In addition, a new resource constraint representation, AND/OR-trees, was introduced that facilitates the efficient description of complex execution constraints. Experiments showed that this AND/OR-tree representation, combined with the proposed transformations, produces small and efficient low-level representations requiring less than 3.5k bytes of compiler memory. This combination was also shown to greatly reduce, by up to a factor of ten, the number of resource checks per scheduling attempt required to model complex execution constraints. These results strongly support the assertion that precise and accurate machine descriptions, designed to be easy-to-maintain and written in a high-level language, can be translated into a low-level representation that can be efficiently used by an ILP compiler.

Acknowledgments

The authors would like to thank Rick Hank for the use of the HP-PA codegenerator and PA7100 machine description, Sabrina Hwu for the use of the Sparc codegenerator and the SuperSPARC machine description, Dan Lavery, Dave Gallagher, and Andrew Hsieh for the use of the X86 codegenerator and machine descriptions, Roger Bringmann for the use of the multiplatform list scheduler, the anonymous reviewers for the detailed reviews, and the IMPACT compiler team in general. The authors would also like to thank Mike Schlansker, Vinod Kathail, and the rest of the CAR group at HP Labs for valuable insight and discussion concerning countless machine description issues.

This research has been supported by the National Science Foundation (NSF) under grant MIP-9308013, Intel Corporation, Advanced Micro Devices, Hewlett-Packard, SUN Microsystems, NCR, and the National Aeronautics and Space Administration (NASA) under Contract NASA NAG 1-613 in cooperation with the Illinois Computer Laboratory for Aerospace Systems and Software (ICLASS).

References

[1] J. C. Dehnert and R. A. Towle, "Compiling for the Cydra 5," *The Journal of Supercomputing*, vol. 7, pp. 181-227, January 1993.

[2] G. P. Lowney et al., "The Multiflow trace scheduling compiler," The Journal of Supercomputing, vol. 7, pp. 51-142, January 1993.

[3] J. C. Gyllenhaal, B. R. Rau, and W. W. Hwu, "Hmdes Version 2.0 Specification," Technical Report IMPACT-96-3, The IMPACT Research Group, University of Illinois, Urbana, IL, 1996, *http://www.crhc.uiuc.edu/Impact/*.

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

[5] 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," Proc. 18th Ann. Int'l Symp. Computer Architecture, (Toronto, Canada), pp. 266-275, Jun 1991.

[6] R. A. Bringmann, *Compiler-Controlled Speculation*, Ph.D. thesis, Dept. of Computer Science, University of Illinois, Urbana, IL, 1995.

[7] E. S. Davidson, L. E. Shar, A. T. Thomas, and J. H. Patel, "Effective control for pipelined computers," Spring COMPCON'75 digests of papers, pp. 181-184, Feb. 1975

[8] G. Blanck and S. Krueger, "The SuperSPARC microprocessor," COMPCON Spring 1992, pp. 136-141.

[9] T. Asprey et al., "Performance features of the PA7100 microprocessor", IEEE Micro, pp. 22-35, June 1993.

[10] Intel, *The Pentium Microprocessor*. Santa Clara, CA, 1993.

[11] Dave Christie, "Developing the AMD-K5 architecture", IEEE Micro, pp. 16-26, April 1996.

[12] B. R. Rau, "Iterative Modulo Scheduling: An algorithm for software pipelining loops," Proceedings of the 27th Annual International Symposium on Microarchitecture, pp. 63-74, Nov. 1994.

[13] A. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques, and Tools*. Reading, MA: Addison-Wesley, 1986.

[14] R. L. Kleir, "A representation for the analysis of microprogram operation," Proceedings of the 7th Annual Workshop on Microprogramming, Sept. 1974.

[15] D. J. DeWitt, "A control-word model for detecting conflicts between microprograms," Proceedings of the 8th Annual Workshop on Microprogramming, Sept. 1975.

[16] J. A. Fisher, *The Optimization of Horizontal MicroCode Within and Beyond Basic Blocks; An Application of Processor Scheduling with Resources*. Ph.D. thesis, New York University, 1979.

[17] P. M. Kogge, *The Architecture of Pipelined Computers*. McGraw-Hill, New York, 1991.

[18] A. E. Eichenberger and E. S. Davidson, "A reduced multipipeline machine description that preserves scheduling constraints," Proceedings of the Conference on Programming Language Design and Implementation, pp. 12-20, May 1996.

[19] T. A. Proebsting and C. W. Fraser, "Detecting pipeline structural hazards quickly," The 21st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 280-286, Jan. 1994.

[20] T. Müller, "Employing finite automata for resource scheduling," Proceedings of the 26th Annual International Symposium on Microarchitecture, pp. 12-20, Dec. 1993.

[21] V. Bala and N. Rubin, "Efficient instruction scheduling using finite state automata," Proceedings of the 28th Annual International Symposium on Microarchitecture, pp. 46-56, Nov. 1995.